Documentation for Breast Cancer Diagnosis Streamlit App

Overview

This Streamlit application allows users to visualize and interact with cell nuclei measurements data. Users can adjust parameters using sliders, and the application will generate a radar chart based on the selected values.

1. **Importing Libraries**

The code begins by importing the necessary libraries:
Streamlit: For creating the web application interface.
Pickle: For loading serialized Python objects (not used in the provided code).
Pandas: For data manipulation and analysis.
NumPy: For numerical operations (not explicitly used in the provided code).
Plotly: For creating interactive visualizations, specifically radar charts.
Python:-

```
import streamlit as st
import pickle as pickle
import pandas as pd
import numpy as np
import plotly.graph_objects as go
```

2. **Data Cleaning Function**

get_clean_data()
This function loads and cleans the dataset:
Reads the CSV file located at data/data.csv.
Drops unnecessary columns: 'Unnamed: 32' and 'id'.
Maps the 'diagnosis' column from string labels ('M' for malignant and 'B' for benign) to numerical values (1 and 0).
Returns the cleaned DataFrame.
Python:-

```
def get_clean_data():
    data = pd.read_csv('data/data.csv')
    data = data.drop(['Unnamed: 32', 'id'], axis=1)
    data['diagnosis'] = data['diagnosis'].map({'M': 1, 'B': 0})
    return data
```

3. **Sidebar Configuration**

add_sidebar()
This function creates the sidebar in the Streamlit app, allowing users to adjust measurements:
Calls get_clean_data() to load the cleaned data.
Defines a list of measurement labels and their corresponding DataFrame columns.
Initializes a dictionary to store user inputs.Creates sliders for each measurement, setting their range based on the data's minimum and maximum values, with the default value set to the mean of each measurement.

Python:-

```
def add_sidebar():
    st.sidebar.title("Cell Nuclei Measurements")
    data = get_clean_data()
    slider_labels = [("Radius (mean)", "radius_mean"), ...]  # List of measurements
    input_dict = {}
    for label, key in slider_labels:
        input_dict[key] = st.sidebar.slider(
        label,
        min_value=data[key].min(),
        max_value=data[key].max(),
        value=data[key].mean(),
        )
    return input_dict
```

4. **Scaling Values**

get_scaled_values_dict(input_dict)

This function scales the user-selected values to a range of 0 to 1 based on the minimum and maximum values of the training data:

Loads the cleaned dataset and separates features from the 'diagnosis' column.

Initializes a dictionary to store scaled values.

For each input value, it calculates the scaled value using the formula:

scaled_value=value−min_valmax_val−min_valscaled_value=max_val−min_valvalue−min_val

Python:-

```
def get_scaled_values_dict(input_dict):
    data = get_clean_data()
    X = data.drop(['diagnosis'], axis=1)
    scaled_dict = {}
    for key, value in input_dict.items():
    max_val = X[key].max()
    min_val = X[key].min()
    scaled_value = (value - min_val) / (max_val - min_val)
    scaled_dict[key] = scaled_value
    return scaled_dict
```

5. **Radar Chart Visualization**

get_rader_chart(input_data)

This function creates a radar chart based on the scaled input values:

Calls get_scaled_values_dict(input_data) to obtain the scaled values.

Initializes a Plotly figure and adds traces for each measurement to create the radar chart.
(Note: The complete implementation for adding traces and displaying the chart is not fully provided; ensure to complete this part.)
Python:-

```
def get_rader_chart(input_data):
    input_data = get_scaled_values_dict(input_data)
    fig = go.Figure()      # Add the traces (incomplete in provided code)
    # Add the traces for Mean, Standard Error, and Worst measurements
    fig.add_trace(
        go.Scatterpolar(
            r=[input_data['radius_mean'], input_data['texture_mean'],
            input_data['perimeter_mean'],          input_data['area_mean'],
            input_data['smoothness_mean'], input_data['compactness_mean'],
            input_data['concavity_mean'], input_data['concave points_mean'],
            input_data['symmetry_mean'],          input_data['fractal_dimension_mean']],
            theta=['Radius', 'Texture', 'Perimeter', 'Area', 'Smoothness', 'Compactness',
            'Concavity', 'Concave Points','Symmetry', 'Fractal Dimension'],
            fill='toself',
            name='Mean'     )
        )

    fig.add_trace(
        go.Scatterpolar(
            r=[input_data['radius_se'], input_data['texture_se'], input_data['perimeter_se'],
            input_data['area_se'],input_data['smoothness_se'], input_data['compactness_se'],
            input_data['concavity_se'], input_data['concave points_se'],
            input_data['symmetry_se'], input_data['fractal_dimension_se']],
            theta=['Radius', 'Texture', 'Perimeter', 'Area', 'Smoothness', 'Compactness',
            'Concavity', 'Concave Points','Symmetry', 'Fractal Dimension'],
            fill='toself',
            name='Standard Error'     )
        )

    fig.add_trace(
        go.Scatterpolar(
            r=[input_data['radius_worst'], input_data['texture_worst'],
            input_data['perimeter_worst'],          input_data['area_worst'],
            input_data['smoothness_worst'], input_data['compactness_worst'],
            input_data['concavity_worst'], input_data['concave points_worst'],
            input_data['symmetry_worst'],input_data['fractal_dimension_worst']],
```

```
                theta=['Radius', 'Texture', 'Perimeter', 'Area', 'Smoothness', 'Compactness',
                'Concavity', 'Concave Points','Symmetry', 'Fractal Dimension'],
                fill='toself',
                name='Worst'     )
        )
```

Updating the Chart Layout
The layout of the radar chart is then updated to ensure that the radial axis is visible and that it has a defined range of 0 to 1. The legend is shown for clarity, and the chart is set to automatically size based on the container.
Python:-

```
# Update the layout of the radar chart
    fig.update_layout(
        polar=dict(  radialaxis=dict(
        visible=True,
        range=[0, 1] )),
        showlegend=True,
        autosize=True )
return fig
```

7. **Model Prediction Function**

add_predictions(input_data)

This function handles the prediction of the diagnosis based on user input. Key steps include:
Loading the Pre-trained Model: The model and the scaler are loaded using the pickle library. The model is expected to be a previously trained machine learning model for predicting benign or malignant tumors.
Python:-

```
# Load the model
model = pickle.load(open('model/model.pkl', 'rb'))
scaler = pickle.load(open('model/scaler.pkl', 'rb'))
```

Scaling the Input Values: The input data from the sidebar is transformed into a NumPy array and scaled using the previously loaded scaler to ensure consistency with the training data.
Python:-

```
# Scale the input values
input_array = np.array(list(input_data.values())).reshape(1, -1)
input_array_scaled = scaler.transform(input_array)
```

Making a Prediction: The scaled input values are passed to the model to predict the diagnosis.
The prediction result (0 for benign, 1 for malignant) is then displayed in the Streamlit app.
Python:-

```
# Predict the diagnosis
prediction = model.predict(input_array_scaled)
st.subheader("Cell Cluster Prediction")
```

```
if prediction[0] == 0:
        st.write("<span class='diagnosis benign'>Benign</span>", unsafe_allow_html=True)
else:
        st.write("<span class='diagnosis malicious'>Malicious</span>", unsafe_allow_html=True)
```

Displaying Probability: The code snippet ends with a placeholder for displaying the probability of the prediction. Ensure to add the necessary logic to calculate and display this probability.

Python:-

```
# Display the probabilities of being Benign and Malignant
st.write("Probability of being Benign: ", model.predict_proba(input_array_scaled)[0][0])
st.write("Probability of being Malignant: ", model.predict_proba(input_array_scaled)[0][1])
```

Important Disclaimer

The application concludes this section with a disclaimer, emphasizing that while the app can assist medical professionals in making diagnoses, it should not replace professional medical evaluations.

python

```
st.write("This app can assist medical professionals in making a diagnosis, but should not be used as a substitute for a professional diagnosis.")
```

9. **Main Application Structure**

main()

The main function orchestrates the entire Streamlit application. It sets the page configuration and manages the layout and content of the app.

Page Configuration: The function sets the title, icon, layout, and initial sidebar state for the application.

Python:-

```
def main():
        st.set_page_config(page_title="Breast Cancer Predictor",
        page_icon=":female-doctor:",
        layout='wide',
        initial_sidebar_state="expanded",
        )
```

Adding Custom CSS: The application can include custom styling by reading a CSS file and applying it to the app.

PYthon:-

```
# Adding CSS in the appwith open("assets/style.css") as f:
        st.markdown("<style>{}</style>".format(f.read()), unsafe_allow_html=True)
```

Sidebar Input: The function calls add_sidebar() to create the interactive sidebar for user inputs.

Python:-

```
        input_data = add_sidebar()
```

Container for Main Content: The function uses a container to hold the main content of the app, including the title and description.

Pytnon:-

```
with st.container():
    st.title("Breast Cancer Predictor")
    st.write("Please connect this app to your cytology lab to help diagnose breast cancer from tissue samples. This app predicts using a machine learning model whether a breast mass is benign or malignant based on measurements it receives from the cytology lab. You can also update the measurements by hand using the sliders in the sidebar.")
```

Layout with Columns: The layout is divided into two columns to display the radar chart and the prediction results side by side.

Python:-

```
    col1, col2 = st.columns([4, 1])
    with col1:
    rader_chart = get_rader_chart(input_data)
    st.plotly_chart(rader_chart, use_container_width=True)
    with col2:
    add_predictions(input_data)
```

10. Running the Application

The application is initiated by calling the main() function when the script is executed. This structure ensures that the app runs smoothly when launched.

Python:-

```
if __name__ == "__main__":
        main()
```

Conclusion

This Streamlit application is designed to assist in the diagnosis of breast cancer by utilizing machine learning models alongside user-friendly interfaces. By allowing users to adjust measurements through sliders, the app provides real-time visualizations and predictions, making it a valuable tool for medical professionals.