Streamlit Documentation: Integrating Machine Learning Models

Introduction

Streamlit is a Python library, open-source and user-friendly, empowering data scientists and developers to swiftly develop interactive web applications for data visualization, machine learning, and beyond. In this extensive beginner's guide, our focus lies in demonstrating the seamless integration of machine learning models with Streamlit, enabling users to engage with these models through interactive interfaces.

Official Streamlit Website: https://streamlit.io/

Prerequisites

Before we begin, ensure you have the following prerequisites installed on your system:

1. Python (version 3.6 or higher)
2. Pip (package installer for Python)

You can check if you have Python installed by running the following command in your terminal or command prompt:

```
python --version
```

To install Pip, you can follow the instructions on the official Python website (https://www.python.org/) for your specific operating system.

Step 1: Install Streamlit

To install Streamlit, use Pip by running the following command:

```
pip install streamlit
```

Streamlit requires various other libraries to work correctly, and the above command will install all the necessary dependencies.

Official Streamlit GitHub Repository: [https://github.com/streamlit/streamlit]

Step 2: Create a New Python Script

Start by creating a new Python script (e.g., `ml_app.py`) using your preferred code editor or IDE. This script will be the entry point of your Streamlit web application.

Step 3: Import Streamlit and Machine Learning Libraries

At the beginning of your Python script, import the necessary libraries:

```
import streamlit as st
import pandas as pd
import numpy as np
import joblib
```

Official Streamlit Documentation: [https://docs.streamlit.io/]

Step 4: Load the Trained Machine Learning Model

In this step, you'll load the pre-trained machine learning model that you want to use in your Streamlit app. First, ensure you have a pre-trained model saved as a file (e.g., **model.pkl**).

Alternatively, you can use the **pickle** library to load the model. The joblib library is commonly used for large NumPy arrays, but pickle is a viable option for simpler models.

```
def load_model():
    # Load your pre-trained machine learning model here
    model = joblib.load('path/to/your/model.pkl')
    return model

model = load_model()
```

Step 5: Create the Streamlit App

Now you can start building your Streamlit application around the machine learning model. Let's create a simple example where users can input features and get predictions from the model:

```python
def main():
    st.title("Machine Learning Model Demo")
    st.write("Enter the following features:")

    # Add widgets for user input (e.g., text_input, selectbox, slider)
    feature1 = st.text_input("Feature 1:")
    feature2 = st.slider("Feature 2:", 0.0, 100.0, 50.0)

    if st.button("Predict"):
        # Process user input and make predictions using the loaded model
        prediction = make_prediction(model, feature1, feature2)
        st.write(f"Prediction: {prediction}")

def make_prediction(model, feature1, feature2):
    # Process user input and make predictions using the model
    # Replace this with your actual machine learning prediction code
    # For example:
    # prediction = model.predict([[feature1, feature2]])
    # return prediction[0]
    return np.random.randint(0, 2)

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

Step 6: Run Your App

Save your Python script and open a terminal or command prompt in the same directory where the script is located. Run the following command:

```
streamlit run ml_app.py
```

Streamlit will start a local development server and open the web application in your default web browser.

Step 7: Interact with Your App

Now that your app is running, you can interact with it directly from your web browser. Input values for the features and click the "Predict" button to see the model's predictions.

NEXT LEVEL:

Advanced ML Integrations

As you become more comfortable with Streamlit, you can explore more advanced integrations with machine learning models. Here are some ideas to consider:

Data Preprocessing and Visualization

- Allow users to upload CSV files and visualize data before making predictions.

- Implement data preprocessing steps and allow users to customize them.

Model Tuning and Hyperparameter Optimization

- Integrate hyperparameter tuning tools like GridSearchCV or RandomizedSearchCV.

- Display model performance metrics to users.

Model Interpretability

- Use SHAP (SHapley Additive exPlanations) or other interpretability libraries to explain model predictions.

- Provide visualizations to help users understand how the model works.

Multiple Models Selection

- Allow users to choose between different pre-trained models for predictions.

- Provide model comparison charts to evaluate the performance of different models.

You have successfully learned how to integrate a machine learning model with Streamlit to create interactive web applications. This comprehensive guide covered the essential steps to get you started, and you now have the tools to build sophisticated apps that empower users to interact with machine learning models in real-time.

Summary

With Streamlit, you can build powerful data-driven web applications with minimal code. It's designed to be beginner-friendly, yet versatile enough for experienced developers. By leveraging the integration with machine learning models, you can create interactive and user-friendly apps that showcase your data science projects and insights. Enjoy exploring the possibilities of Streamlit and take your data science work to the next level!

SOME NOTABLE FEATURES AND WIDGETS ON STREAMLIT.

we'll explore some of the most notable features and widgets that Streamlit has to offer. Whether you're a data scientist, developer, or someone new to web development, Streamlit empowers you to build stunning data-driven apps with minimal effort.

- **Markdown**: Streamlit allows you to use Markdown to format and stylize text in your app. You can use Markdown to create headings, bullet points, bold/italic text, links, and more. Markdown is handy for providing clear instructions, explanations, and descriptions in your documentation.

```
# Use Markdown to format text
st.markdown("# Welcome to My Streamlit App")
st.markdown("This is a *Streamlit* app to demonstrate various features.")
```

- **st.image**: This widget is used to display images in your app. You can use it to showcase screenshots, visualizations, or any other images relevant to your documentation.

```
# Display an image in the app
st.image("path/to/image.png", caption="Streamlit Logo", use_column_width=True)
```

- **st.code**: With this widget, you can display code snippets in your app. It is useful for demonstrating usage examples or sharing code snippets with users.

```
# Display a code snippet in the app
st.code("""
import streamlit as st

# Streamlit code example
st.title('Hello, Streamlit!')
""")
```

- **st.sidebar**: The sidebar is a special section where you can place widgets that persist across different pages of your app. You can use it for navigation, input controls, or displaying additional information.

```
# Adding widgets to the sidebar
st.sidebar.header("Navigation")
selected_page = st.sidebar.radio("Go to", ["Home", "About", "Contact"])
```

- **st.selectbox**: This widget allows users to select options from a dropdown menu. It is useful for providing choices and interactivity in your app.

```
# Create a selectbox widget
selected_option = st.selectbox("Choose an option", ["Option 1", "Option 2", "Option 3"])
```

- **st.button**: You can use this widget to add buttons to your app. It is useful for triggering actions or performing specific tasks when the button is clicked.

```
# Add a button to the app
if st.button("Click me!"):
    st.write("Button clicked!")
```

- **st.dataframe**: This widget displays Pandas DataFrames in your app. It is useful for presenting tabular data to users.

```
# Create a DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 22]}
df = pd.DataFrame(data)

# Display the DataFrame in the app
st.dataframe(df)
```