Carnegie Mellon University

04652 EAI Systems, Spring 2024

Mini 2, Assignment 3: Dashboards in Dash

Expected individual study hours: 12

Points: 100

Purpose

In labs 1 and 2 of this Mini, you have constructed a REST API to the Iris flower classification model. The API has been tested using direct HTTP requests generated with a tool such as Postman which also allows for viewing the request return content.

Having a visual interface to the model's API is obviously a more direct and usable approach to understanding the model operation and explaining its predictions to those unfamiliar with low-level API technologies.

The Dash framework is our tool for building graphical user interfaces (GUIs) in this course, and we have reviewed some of its core components in class. The prep learning activities run through the first 3 pages of the Dash tutorial.

The purpose of this lab and the following one is to put that background into practice to build a simple dashboard for the Iris model exercising the available REST API from your earlier work.

Target Learning Objectives

- The importance of user interface and dashboard prototyping and evolution in the agile process as a communication tool.
- To build a working knowledge of core features of Dash – components and their update callbacks.

Preparation/Background materials

- Complete the first 3 sections of the Dash tutorial at https://dash.plotly.com/tutorial through basic callbacks.

Instructions

1. The file app_ui_template.py contains a template for the dashboard you will create in this lab. It relies on the final version of the Iris API implementation from the last lab – which of course relies on the "back end" implementation of the model itself in your base model python code.

2. The iris_extended_encoded.csv file provides the data needed as usual. Both files will be posted to Canvas in the Files/Mini2 folder.

3. The file is runnable as is, though of course with mostly missing content. But the layout issues have been solved for you. Everything in the app.layout section of the code should be correct apart from the values to be filled in and marked inline. The HTML components and their nesting should not need to be modified – let me know if you find otherwise ☺

4. The purpose of this assignment is to focus on usage of the Dash components, setting their values, and importantly understanding the dynamic updating mechanism in the callbacks that links them together. Those aspects of the code are left for you to complete and are marked with indications like "<add function var mappings here>" or other instructions in comments.

5. Of course, remove the brackets and surrounding quotes as needed to form correct inputs to the python parser. The quotes are needed to assure the application runs minimally out-of-the-box so you can work incrementally and continue to test your updates in continually running code.

6. Feel free to play with the web page layout though the provided Div's and style directives are fine to get the layout we've seen in class.

7. Likewise, feel free to add more components are you might like to experiment with.

Lab Task

The template as provided supports four tabs – one for loading locally and uploading to the REST API a dataset and viewing its content in a histogram, scatter plot, and table. The second tab supports building an instance of the Tensorflow layers in the Iris model and training that model instance using a dataset instance uploaded previously using the first tab. The third tab is UI support for scoring individual Iris data samples and the fourth tab drives the batch test API endpoint.

1. First, there are missing values nested inside the app.layout components – mostly for the inputs required by the histogram and scatter plot components to control their selections. Be sure to code these inputs generally -- i.e. not hard coding the column names from this specific Iris data set but reading them from the dataframe metadata!

2. You'll need to provide the Dash annotations and callback functions to construct the HTTP requests for each of the API endpoints needed by the UI.

3. Your callback functions should also extract the return values and place them into the appropriate app layout elements as indicated in each tab.

4. Key warning/requirement: your feature histogram, scatter plot, and scrolling table should redraw themselves whenever:

    a. the input dropdown controls related to them change, or
    b. whenever the dataframe containing the currently loaded dataset is changed.

    Note that this means you will need to set up "chained" input triggers for the histogram, scatter plot, and scrolling table to redraw themselves whenever the "load" callback completes.

5. Upload to a lab3 folder in your GitHub repo your complete code listing along with screen shots of each running tab in operation. Include copies of the text output in your API/backend and Dash service console logs.

Upload your code into GitHUb along with screen shots of all 4 tabs of the Dashboard with working content shown. Answer the following discussion questions and upload your responses to GitHub along with your code files.

1. What did you find easy to do in Dash?
2. Likewise, what was hard to implement or you didn't wind up getting it to work?
3. What other components, or what revised dashboard design, would you suggest to better assist in explaining the behavior of the Iris model to a client?

4. Can you think of better ways to link the “back end” Iris model and its results with the front-end Dash functions?

Deadline

Tuesday, April 16 at 4:59 PM CAT