DS 5500: Client-Server Visualization and Modeling
Due Date: March 13th, by beginning of class

In this assignment you will:

- Run a simple ML model on a server
- Feed results from the model to an API
- Design visualization (using D3) which accesses data through API

You are going to create a visualization which explores performance of different classification algorithms on the Blood Transfusion Service Center Data Set (transfusion.data). Specifically, you will create a web-based visualization using D3 which displays the ROC curve for logistic regression with the selected preprocessing and hyperparameter choices.

Starter code is available here

flask_roc.py contains starter code for loading the data and getting your flask server up and running.

Main (same for everyone to ensure consistency):

- Reads in the dataset
- Splits into train and test
- Runs the app

Run the code with python flask_roc.py (don't change this and run it in a jupyter notebook)

Part 1) Complete the ROC class and add it as a resource

Specifically, it **must** contain functionality for users to to preprocess the data using either standardization or min-max scaling and any value of C they wish. Make sure to **properly standardize the data** (ie, don't standardize using mean of entire dataset, only training set).

Run your flask server with python flask_roc.py then use your browser and go to wherever the app is running (usually http://127.0.0.1:5000/) to make sure you can access the tpr, fpr and thresholds for a given model configuration.

Hint: return the data as a list of dictionaries in the form [{'tpr': tpr1, 'fpr': fpr1, 'threshold': threshold1}, …], one dictionary per threshold value. This will make it easier to work with in D3.

Make sure your flask server works and you can see the correct fpr, tpr and thresholds in browser. DO NOT PROCEED WITH THE FRONT END DEVELOPMENT UNTIL THIS IS WORKING PERFECTLY.

Part 2) Read data from the API with D3

1. **Keep your flask server running**
2. Start up a second server “python3 -m http.server” in the D3 folder. This server will serve up your visualizations. It is **separate** from your flask server. (if you wish to do a little extra work and have the flask server also serve up your visualizations, that’s fine)
3. In main.js, edit the D3.json() function to make sure you can replicate the data fetching from the API with D3 (test it with the same url you were using in your browser in part 1).
4. Write code (html, css, js) to retrieve from the user which preprocessing type and value for C they are interested in evaluating. There is no prescribed way we’ve set out for you to do this, so there’s freedom to do it how you’d like
    a. For instance there could be a field where users can enter values for C
    b. The UI must be clear and easy for users to understand
    c. We don't expect you to know how to do this off the top of your head, so feel free to Google to figure out how to implement your desired method. **As always, don't just copy the code -- please write and adapt code yourself using the ideas you read about. Cite your sources in a comment!**
5. Now that you can get the preprocessing step and hyperparameters the user wants to explore, create a URL string that you can use to fetch the data for the ROC curve from your flask server (d3.json(url))
6. Remember, every time a user selects a new model pipeline configuration, you need to make another call to your API

Checkpoint: by this step you should have a working python flask server that feeds fpr, tpr and threshold values for a given model configuration and a client which allows users to specify their desired model then logs the fpr, tpr and thresholds to the console.

Part 3) Line chart

Helpful links:
Update D3.js data with button press
Mike Bostok’s line chart

Now it’s time to put it all together and turn the data into a ROC curve line chart. You do not need lines for multiple models (just one model at a time), **but you do need to include the diagonal random choice line**.

Your line chart must have a title, axis labels, and axis ticks. Please choose appropriate colors, if necessary. Remember to get it right in black and white first, though.

Your visualization must properly update after a user has made a different choice of preprocessing step or C (you can have a little button “visualize” that gets data for a new model specification and fetches the results from python. Then update the line chart accordingly).

You can reuse some of your code from the previous D3 assignment(s).

Visualizations: your visualizations must

- Shows the data asked for in the prompt
- Have a meaningful title, axis labels, data labels, and if necessary a legend
- Uses reasonable, sensical visual encodings (grounded in theory from class/readings) that are appropriate for the given task(s).

Code: your code must

- be readable
- Be your own, original work
- have meaningful variable names
- be clearly commented so the TA can easily grade it

Submission directions:

Upload your code and data to a github repo (make it public).

Write a little README.md file which clearly explains how to run your code. If the TA cannot figure out how to get your code to run because of inadequate documentation, you will fail the assignment.

Copy the url to your repo, put it in a document, and submit on blackboard.