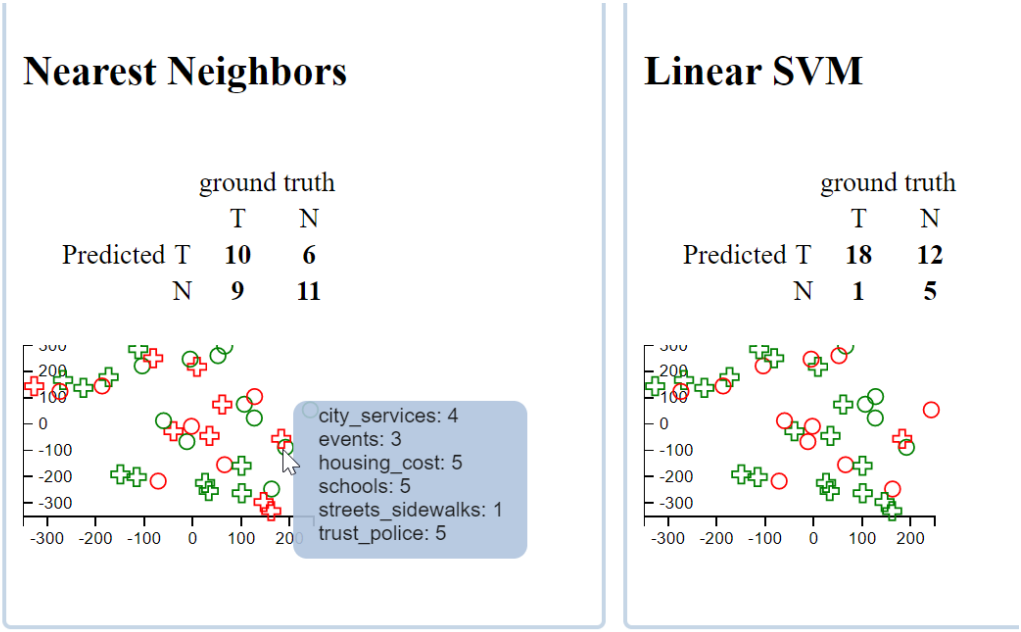
Projections for Visual Analytics

Due: Thursday, April 6th by 11:59am (before class)



Description:
In this lab, we're going to continue working on our visual analytics application from the previous lab. We're still looking at the Somerville Happiness dataset. This data has six dimensions, and it was impossible to see patterns in all six dimensions at once -- we used scatterplots to select individual pairs of dimensions to view a model's predictions in. This hampered our ability to both understand the dataset and accomplish our task: choosing a good machine learning model.

We learned in class that one way to get around the *curse of dimensionality* can be to use machine learning to project high dimensional data down to two dimensions (aka Dimensionality reduction). In this lab, you will be using the scikit-learn python library to try out a few different projection techniques. You will then update your visual analytics tool from the last lab to show the result of the projection to the user. Lastly, you will play with an online tool to better understand the limitations of *t-SNE*, a popular nonlinear projection technique.

Installation:
Try to install the packages with

pip install -r requirements.txt

If that proceeds without error, then you should be able to start this lab.

Workflow Overview:
There are a few moving parts to this lab. Here's a high-level workflow of what's happening:

1. Set up the server via Flask: establish the connection between the server (Flask) and the client (Browser).
2. Generate 2D projection(s) of the Somerville Happiness data: In the server, call the appropriate projection algorithms. These projection algorithms should produce a 2D position (x, y) for each of the data points in the input data (i.e. convert the data from 6D to 2D).
3. Visualize the projection(s): Send the output of the projection algorithms to front end. Visualize the projections with a scatterplot.

Your Tasks:
Your goal is to update the skeleton support code to generate projections of the dataset we've been studying. You will then update your front-end to visualize those projections. Lastly, you will answer a couple of questions about your understanding of *t-SNE*.

- First, you will want to copy your scatterplot and your confusion matrix code from last week to the code given to you this week.
- Set up the server and make sure that you can run the html file in a web browser.
- Calculate projection. **(25 points)**
    1. In project_somerville.py file, Fill in that code with the correct calls to each projection method. For t-SNE, try 4 different values for the "perplexity" argument: 2, 5, 10, and 20. Pick one that you think is best for your application. In your writeup, write a few sentences about why you decided to use that perplexity value. **Hint:** Look at the documentations for each function, which is linked in the comments of the Python file.
        - We will be doubly impressed if you can make a user-interface change so that a user can update the perplexity argument in the front-end, have that propagate to the server, and then update the visualization based on the resulting t-SNE projection -- thus completing an interactive, full-stack application.
        - Note that the perplexity argument is unique to t-SNE. So maybe don't show the argument in the front-end if the user chooses a projection that's not t-SNE…
    2. In server_solution.py file, call the right function to calculate the projection and pass the result to the frontend. You can choose different models in the web page to see the changes.
- Visualize the projections with a scatterplot. **(25 points)**
    1. In script.js file, fill in code to enable the "on change" event generated by the radio buttons.

2. Set up the domain when the current projection option is not axis_aligned.
3. The new projections have meaningless axes now because they don't represent any given dimension[1]. As a result, we'd like to be able to view the attributes of any data point when we mouse over them, as a tooltip. Add a tooltip to each point in the scatterplot that displays the attributes of that point (see Figure above for an example).

- One of the projections used in the first part of this lab is the t-SNE projection, or t-distributed stochastic neighborhood embedding. We discussed how it works in class, but there is a really nice tool online for playing around with it on different datasets. Skim through the article at https://distill.pub/2016/misread-tsne/ and play with the tool a little bit. Write up a paragraph about what analytical tasks you think t-SNE is a good tool for, and what it is a bad tool for. **(20 points)**
- Lastly, do a small writeup (at least one paragraph each) answering each of the following questions: **(30 points)**
    1. Did the addition of projections change your choice of which prediction model you would use for the Somerville Happiness dataset (e.g., AdaBoost, Decision Tree, etc.)? Explain your reasoning.
    2. Compare the four projections (PCA, MDS, t-SNE, and UMAP), which one is the most useful for your task (of picking the best model)? What about the original axis-aligned scatterplot? Explain your reasoning.
    3. What sorts of harm can be done by using the projections that you tried out in the lab?

Files Given:

- index.html – the HTML file
- script_starter.js – the Javascript file
- style.css – the CSS file
- model_prediction.json – a json file contains the prediction result of each model (same as the last lab)
- project_somerville.py – a python script that contains skeleton code for projecting the data. Should generate pca.json, mds.json, tsne.json and umap.json (for debugging purposes)
- server_starter.py a python script that sets up the server using Flask.
- requirements.txt – a list of python packages that are required for this lab. Can be run with `pip -R requirements.txt install`
- somerville_happiness_survey.csv – The raw data file that is loaded by the script in CSV format.
- somerville_happiness_survey.json – The raw data file that is loaded by the script in JSON format.

[1] This is not entirely true for PCA – the axes of PCA (that is to say, the principal components of PCA) are linear combinations of the original data dimensions. However, these axes are often still difficult to interpret.

Helpful Resources:

- Scikit-learn has some wonderful resources for both learning how to program using their library as well as understanding conceptually what the machine learning models are doing. In general, search "scikit-learn API PCA" or "scikit-learn API MDS" to find documentation and examples. You can also look at the user guide to find interesting articles, such as this one comparing various manifold methods.
  https://scikit-learn.org/stable/modules/manifold.html

How to submit your lab:

- Submit the support_code folder as a zip file and your writeup in a PDF file.
- Using Canvas, each team should make only one submission.