Process book

* Overview and Motivation:

One important application of information visualization is that it helps domain experts understand multivariate data, which is hard to visualize in conventional ways. Dimension reduction method such as PCA can help understand the data, but some features of the high dimension space are lost through the dimension reduction process. The main inspiration of this project is the previous work *Visual exploration of high dimensional scalar functions* by Samuel Gerber. In this paper, a method that involves computation of Morse Smale Complex is used to cluster high dimensional data by calculating the significance level of features shown in the data. Then an inverse regression method is used to show the behavior of the point clouds in different clusters. The existing tool that uses this algorithm is written in C++ and python, which does not provide powerful interaction as what javascript does in frontend. Designing such interface will help users do better analysis on multivariate data and understand the high dimension space.

The dataset used for this visualization is not limited to any single dataset. The first dataset that is going to be analyzed is related to nuclear simulation are obtained from Nuclear Energy University Program. Other multivariate datasets of users’ interests from different fields can also be used for this visualization. We got some good feedbacks from through the peer review session about the size of the data to visualize. When the size of the dataset is too large, we would have a function that only selects certain samples from the whole dataset to visualize.

* Questions:

Could we get some meaningful relationships between different attributes of the data? how is each dimension related to each other?

How is the data distributed in the space, how are the points clustered in the high dimensional space? Are the tree view we try to create consistent with the geometric location of data in high dimensional space?

Can our visualization help to answer the above questions clearly? Which part can be addressed well and which part is not?

The original method from the paper calculates the clusters and store the data into local directory. Can we update the tree structure during runtime?

* Data: Source, scraping method, cleanup, etc.

We start with nuclear simulation dataset of 10000 points. Each attribute is numeric. Preprocessing is not required. However, the current available algorithm that calculates the clusters is not in JavaScript. Since we do not want to spend a long time rewriting the algorithm in JavaScript or dealing with python server, we decide to treat this part as preprocessing and run the algorithm locally, which is also the advice from the TA. The result is stored in a json file that gives specification of how the clusters would merge or split as persistence changes. The front-end requires both raw data in the form of csv as well as the json file.

Although there is not tree structure mentioned in the original paper, we try to create the tree-view first since it should give some notion of the algorithm. Since what we want to show through other views are clusters existing in the current tree structure, the tree is the main view that all other views need to be linked to.

There are also other views such as scatter plot, histogram and boxplot in our proposal, which we will also put in our initial prototype as the first step.

Together with the treeview, we also think of adding a view that shows the clusters in high dimension space.

Currently we have the first dataset we would like to analyze together with its processed version that contains the information of how clusters merge or split. We believe the existing algorithm should be robust for other multivariate data.

We have several views that show different aspects of the dataset. However, the tree view does not fully function yet. After the preprocessing, we have the specification json file. We have written the algorithm that updates the clusters in python, which is It will read a user’s input of persistence level and update the number of clusters, with which a tree will be updated. We will convert it to javascript soon. After the tree is implemented, all other views will connected to the tree for interaction.

Initially we implemented tree structure on the processed data. It showed the tree structure at different resolution depending on the persistence level of user’s interest. The size of the node that represents the cluster will also depend on the number of levels of the tree. Right now the algorithm would compare the persistence value of each cluster with the persistence of user’s interest. Then it would build the tree based on the clusters that have persistence level above user’s interest. Since we do not want to remove and rebuild the tree each time, the next step would be using filter and class to just visualize certain levels of the tree depending on user’s interest.

Since one of the goals of the project would be making the code reusable for other data sets, we started to read about JavaScript Design Patterns to make each class not depend on the input data.

Instead of rebuilding the tree each time, currently we were able to only show part of the tree using the filter function. However, we need to rescale and reposition the tree after filtering. We tried with scaling function on the whole tree, which did not work well since it would also change the size of the link and made the visualization look terrible. Next step would be finding a better way to zoom the tree.

We wrote scaling function to repostion and transform the visible tree node that met with user's interest based on persistence level. We also added two buttons that the user could click to update the tree structure.

We added interaction between tree and plot such that when a user clicked on one tree node, the points that belonged to this cluster would show up in the plot. However, we did not know whether this part took a long running time since each node had to go to the base partitions (the bottom leaves of the tree) to fetch out the points in this node.

After some study, we found out that the filter function might not work as we supposed. It created giant array whenever we used that function, which was not preferred, especially when the samle size was very large. Right now we wrote a scaling function to scale and reposition each tree node that is visible at persistence level of interest. However, we were not sure whether this would be a better option than using function in d3 to re-layout the tree. We would try this method and see whether this would give better performance.

After we read about JS Design Pattern, we decided to change the main script to something that behaved like a manager, where only the manager knew about all other classes and different classes did not know about each other. We would look at dispatch in d3 and might use that in our design.

Thinking of whether it was necessary to adding collapsing and opening to the tree node.

The updateTree function takes a lot of running time when more nodes were displayed as persistence goes down. We need to check each part of the function using console.time and console.timeEnd. We will think of what can be done accordingly after we figure out which part causes the lagging behavior in the front end.

It turned out that redrawing (transforming) nodes and links took most of the time, though we only updated the visible nodes and links, which should not take that much time. Anyway, we need to think of an alternative way to update the tree.

As we analyze further, we find out the scaling function we write ourselves caused this problem. We were transforming the nodes and links based on the relationship between visible tree and the overall tree structure. We will think of an alternative way to do this.

A scaling factor is calculated from visible tree and overall tree to transform the treenode. We can get rid of the lagging behavior of the visualization in this case.

We separate different classes of the visualizations so that these visualizations do not know and depend on each other. There is a manager script that knows about the behavior of all the classes and all the interactions are done in this class.

We add some animation when updating the tree structure.

There is some lagging behavior when first constructing the tree. We are not sure right whether that can be got rid of.

Several things that may be done: 1. Changing the size of the tooltip. 2. Labeling the axis of the plot. 3. Interaction for user's input (Slider would be better than buttons) 4. Hover or Brush for scatter plot. 5. Selecting attributes of dataset to visualize; 6. Create Slider

Optional: 1. Regression Curve, 2. PCA Projection, 3. Rebuild the tree each time.

When any of above is done, will be highlighted yellow.

Tooltip size is changed. Dynamically changing the size of tooltip may be added later if we have time.

Slider is added. This part takes a while when we try to keep separation of concerns. We create an Event class that represents the slider, which should not know anything about other class. Thus the interaction part for the slider is done in Manager.js, which knows about all the classes.

We try to reconstruct the tree each time by setting a filter on each node and flip the attribute between children and \_children. In this way the tree layout would also update based on the current tree. However, this increases runtime compared with the previous method. However, there is a problem. If a new tree is constructed each time, it will lose the information of the previous node. Then we cannot listen to the change on the treenodes since the nodes are removed each time. Right now we are thinking of a way to solve this issue.

We rewrite the update function for the tree, such that tree is rebuilt each time, which leads to better layout for the tree. We also uses classes instead od removing and appending each time, which would give better runtime. We also add more interactions on the tree and separate clicking with double clicking. Right now when users click on the tree, plots are updated. When users double click on the tree, the corresponding node would open or collapse.

We will now focus on the interaction on the plot.

There is a bug with the treeupdate function, which is shown when the slider is used. We will worry about this in the end.

Attribute labels are added to the plots.

Basic interactions are added to the scatter plot. The value of the point (all the attributes) is shown when hovering over the plot.

* Exploratory Data Analysis: What visualizations did you use to initially look at your data? What insights did you gain? How did these insights inform your design?
* Design Evolution: What are the different visualizations you considered? Justify the design decisions you made using the perceptual and design principles you learned in the course. Did you deviate from your proposal?
* Implementation: Describe the intent and functionality of the interactive visualizations you implemented. Provide clear and well-referenced images showing the key design and interaction elements.
* Evaluation: What did you learn about the data by using your visualizations? How did you answer your questions? How well does your visualization work, and how could you further improve it?