Max Curran

Semantic Data Analysis Final Write-up

Introduction

Last semester I worked with Jim McCusker on the HHS Metadata Challenge. The project was an attempt to create a source of data stored in annotated RDFs which would allow for the visualization in a meaningful way. The data was taken from a CKAN database and converted to RDF. The data is originally stored as a csv file, and is converted using the csv2rdf4lod tool. Then the real effort of the project was to annotate and standardize the RDF vocabulary, so that it would simple to compare any of the datasets. It would also allow for the data to be visualized in an intuitive way, by taking advantage of the linked datasets.

For the project I contributed a set of functions which would allow for quick and highly variable visualizations. I then added a way to use qb.js to use these functions to visualize the data. This framework retrieves the RDF through a SPARQL query and then visualizes a scatter plot matrix. However, I extended it to allow for one of my functions to be supplied along with a set of options which would cause the data to be visualized in a different form.

Charts

The charts were designed to compartmentalize the visualization, analysis and data retrieval. In the past I had found that performing all of these tasks in conjunction to cause various errors. It is also difficult to recreate the visualization with different analysis or data retrieval. In order to create the lowest dependency between the visualization, analysis and data retrieval, I wrote functions for analysis and visualization for a variety of types of charts. All of these functions took the data, and a set of options stored as a JavaScript object. This would allow a user to retrieve the data however they chose then convert it to the correct format, then supply it to the function with all desired options and then the visualization would work. This of course meant that the functions must make no assumptions about any specifics of the visualization. All the specifics would be set by the user in the options object they pass to the function. The separation of analysis and visualization was done so that the user could perform their own analysis on the data, and simply call the visualization function instead of the analysis function.

Each type of chart has different options based on the nature of the chart. These difference in options could have been reflected by different input parameters; however, all of the functions simply accept an options JavaScript object. This is because the functions differ only can therefore be given the same data and options parameter and still work. The options which don't affect the visualization would just be ignored. This fact is important because in JavaScript functions are first ordered values, meaning variables can be set the function. This allows the functions to be supplied to the qb.js to take one of the functions and treat it in a general sense. It is simpler than directly integrating the two, because any new functions do not have

to be integrated. As long as they have two input parameters they will work with the framework.

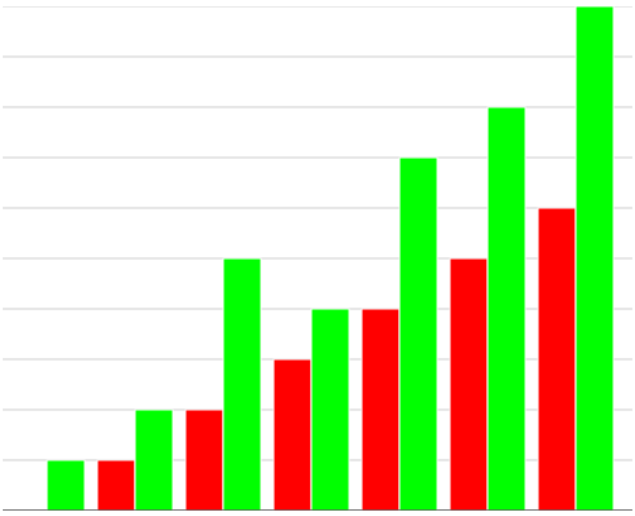
The specific charts I implemented were the Bar Chart, Scatter Plot, Pie Chart, and Parallel Coordinates. All of these charts have two related function, one to "make" the chart and one to "draw" the chart. The "make" function is where all of the analysis occurs. After the analysis the corresponding draw function is called. The "draw" function creates the actual visualization. All of the functions have a simple naming convention as well. The "make" function is always called "make*Chart*" and the "draw" function is always called "draw*Chart*." Although some charts, such as the Bar Chart, do not have any analysis as of now, they still have a "make" function in order to keep everything standard and provide a place for any future analysis.

Another important feature of the charts is that each chart is in one JavaScript file. There is also one file for all analysis function to allow these to be reused for other charts. Another file in the set contains functions for taking the data from a given form to the assumed form. This was done to separate things based on functionality. It also allows for the reuse of functions, namely in analysis.

Bar Chart

The Bar chart is a simple visualization. It shows data by rectangles, whose size is determined by the data. Although simple, it can be more than adequate to visualize low dimensional data. In general one axis is discrete or consecutive values. The other dimension is then of a real value. The rectangles have fixed width in the discrete dimension, and have varying length based on the real value. Also one edge is fixed to the axis of the discrete dimension. Since the visualization

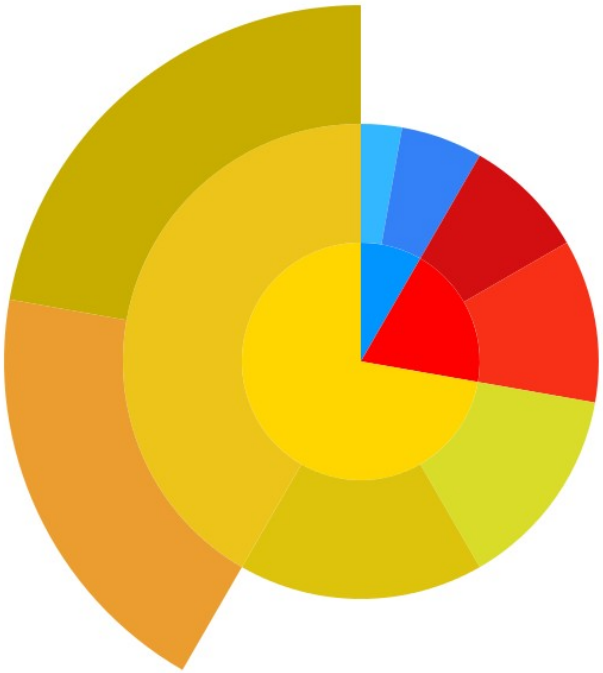
needed to show any dataset, the visualization can handle datasets, with values with opposite sign. In this case the axis for the discrete value is translated to allow rectangles to be shown in their entirety. The visualization also allows for the chart to be orientated in one of four ways. Although the charts show the same data, the differing orientations appear drastically different and a user may want to use this. The bar chart features no analysis due to its simple nature. However, it may be useful to provide a tool which colors each bar based on a supplied dimension. I also created an extension of the bar chart to allow multiple real values to be visualized for each discrete value. Below is an example of the Bar chart with multiple real values:



Pie charts

The next simplest chart I created is the Pie Chart. A Pie Chart represents values by dividing a circle into sections whose angles are relative to the values in the data. This type of visualization is not effective, because it is impossible to compare two values from different Pie Charts, because the section it is given is determined by its relative value to the sum of the data. However, it is still useful

when the only concern is to the value relative to other values within the same dataset. I extended the normal pie chart, which can only display a list of values, to allow for the creation of a radial tree like visualization. The visualization accepts an array, where each element is an array or a number. It then goes through each element of the array; if the element is a number it draws a section with a radius relative to the number to the sum of all elements in the array. To compute the sum, it sums all numbers and the sum of all arrays. If the element is an array, it draws a section with a radius relative to the sum of that array to the sum of all elements in the array, and then recurses on the particular array. Then the inner and outer radius is determined based on the elements depth in the original array. For example for the array, [[1,2] , [3,4] , [5,6, [7,8]]], the following Pie Chart would be made:



Scatter Plot

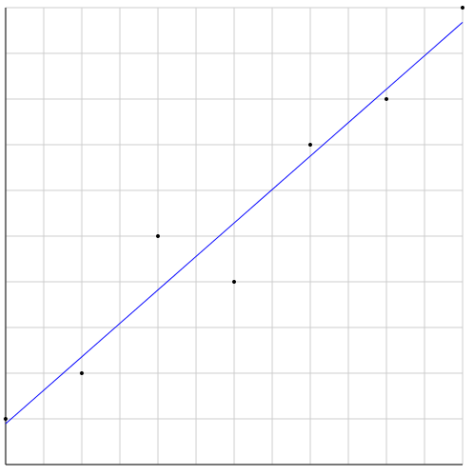
A Scatterplot displays two dimensional data. Each point is represented by a marker whose projection onto orthogonal axes is based on the value of the point in the corresponding dimension. Scatter plots can show clusters of data. They also allow for the identification of correlation between two dimensions. The main drawback of a scatter plot is that at most three dimensions can be shown in a single chart; however, my function can only display two dimensions. For data with more than three dimensions it is possible to display multiple scatterplots, each with only two dimensions of the data. Although I do not have code to create one of these, my scatterplot function can be used as a building block of it.

Along with points of the data, a scatter plot allows for continuous curves to be displayed as well. In order to aid in identification of relationships between the dimensions, I added the ability to compute a least squares regression of a polynomial of a specified degree. The algorithm tries to find a polynomial which minimizes the sum of the distances between the points and the function. It does this by solving the following matrix equation:

$$\begin{bmatrix} n & \sum x_i & \cdots & \sum x_i^m \\ \sum x_i & \sum x_i^2 & & \sum x_i^{m+1} \\ \vdots & & \ddots & \vdots \\ \sum x_i^m & \sum x_i^{m+1} & \cdots & \sum x_i^{2m} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{m+1} \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum y_i x_i \\ \vdots \\ \sum y_i x_i^m \end{bmatrix}$$

Where x are values of the point along one dimension, y are the values of the points along the other, and m is the degree of the polynomial. The feature allows for more accurate identification of relationships between dimensions. The user no longer has

to rely on what they believe, but can rely on an actual curve that is drawn. An example of a scatter plot with a linear least squares regression is shown below:



Parallel Coordinates

Parallel Coordinates is a variation of the Scatter Plot. In a Scatter Plot, the axes are orthogonal; however, in Parallel Coordinates the axes are all parallel to each other. This allows for any number of dimensions to be shown at once. Of course the notion of a point and how it displayed must be altered. In a scatter plot a maker is drawn and its projection onto the axes is based on the point. In Parallel Coordinates, line segments are drawn between each set of adjacent axes, and the endpoints correspond to the value of the point at those axes. Parallel coordinates are useful because they are a good way to visualize many dimensions at once. However, most people are not familiar with the visualization and it is hard to recognize most common relationships. People are used to seeing these relationships on orthogonal axes and not parallel axes. Although Parallel Coordinates do allow for the display of all axes at once, relationships can only be determined effectively between adjacent axes. This means that the ordering for the axes is important. It is sometimes difficult to track points across the chart as a

whole, especially when there are many points. Coloring the points allows for easier tracking since they would be different from other points. The following sections discuss algorithms to alleviate the tracking and ordering problems.

Ordering Algorithm

Last semester, I devised an algorithm to order the axes of a Parallel Coordinates chart. This allows the dimensions to be specified at runtime, and still produce a visually relevant ordering. Usually the creator of the chart would specify a set ordering. However, allowing the user to select dimensions to be displayed gives the power to interact with the data and obtain more information from the visualization. For the axes to be ordered in a relevant manner, an order must be specified for every combination of dimensions, or an algorithm which allows the order to be determined automatically must be created. I opted for the second option.

The order of the axes is important because it allows the user to see relevant patterns in the data. Since not every pair of axes has a significant relationship, it would be beneficial for these pairs to appear consecutively. This lead to the general algorithm, a value would be assigned to each pair of axes. Then the order which minimizes the error is selected, where the error for an order is the sum of the values of any axes that are adjacent to one another. This problem can be abstracted to the Traveling Salesman problem. The graph is a complete graph, where each vertex is an axis. The weight of an edge is the value for the pair of axes it is connecting. Then you must find the minimum walk which visits every vertex once, but starts and stops at different vertices.

The key to the algorithm is how to compute the value given a pair of axes. I have created two functions, one of which is highly mathematical, and the other is highly visual. Both of these produce different results, but neither is better than the other, since there is absolutely no way to quantify the effectiveness of a chart. The highly mathematical function tries to display axes which show either a linear or quadratic relationship. It does this by computing the linear and quadratic least squares regression error. It then normalizes them and returns the smaller one. The highly visual model counts the number of times the line segments between two axes cross.

K-means Clustering

K-means clustering is a way to help solve the tracking problem, by grouping similar points into groups and assigning each group a color. This does not allow for tracking of individual points, but allows for groups which are similar to be visually clumped together and since these colors are assigned based on the properties of the points, tracking their approximate values is possible. The K-means clustering algorithm iteratively assigns a point to a cluster by finding the cluster whose average is closest to the point. It then computes the average of all clusters. This process is repeated until no points change cluster. The key to the algorithm is how to initially assign points to clusters. After trying multiple methods, I found the best way was to initialize the clusters was to randomly select points and assigning the cluster averages to those points. The probability of each point is relative to the similarity, the reciprocal of distance, of the closest point already selected. This starts the clusters out being far apart. Another way to improve the results of the algorithm are to run it multiple times and use the clustering that minimizes the K-

means SSE objective, which tries to minimize the sum of the distance between all points and its corresponding cluster center.

K-means clustering separates the data into Voronoi cells. This means that a point is assigned to the closest cluster, and only to that cluster. The density of points does not affect cluster assignment, but it does affect the location of clusters. This means that two clusters of differing sizes may be partitioned incorrectly by the algorithm. K-means can also only deal with linear space. This means some oddly shaped clusters will be clustered incorrectly. The algorithm also requires that the user specify the number of clusters. There are heuristics for selecting this value, but they required the algorithm to be run multiple times.

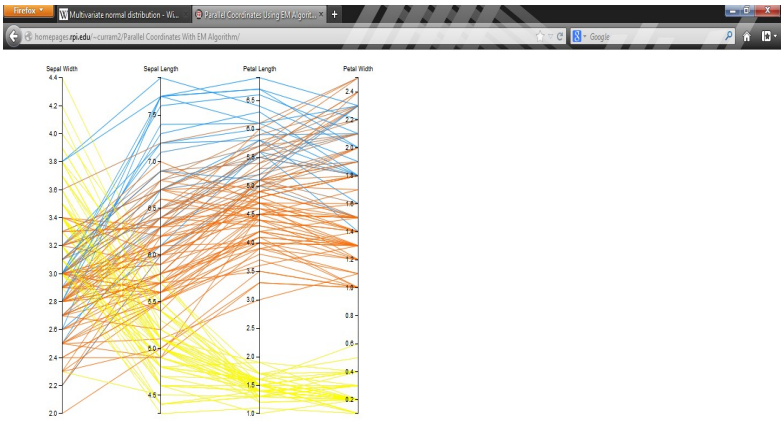
Kernel K-means

One way to improve the K-means algorithm is to make use of a Kernel matrix. A Kernel matrix holds the distances between all pairs of points in a new feature space. This feature space does not have to be linearly related to the input space, where the input space is where the points are located. The points may be translated into a quadratic space, where any quadratic relationships would appear linear. Then using the values in the Kernel, it is possible to find quadratic cluster using the modified K-means algorithm. There are some slight modifications that need to be made to support the Kernel. The values of the points in the feature space are unknown, because some feature spaces have infinite dimensions, and it is always very costly to compute. Due to this, it is impossible to know the exact cluster averages. However, it is possible to compute the distance to the cluster average, by using the values in the Kernel matrix. This algorithm, which I have implemented, allows for the non-linear clusters to be found.

Expected-Maximization Algorithm

The Expected-Maximization Algorithm, EM, is another clustering algorithm. Both K-means and EM use cluster centers to model the cluster; however, K-means is distance based, and EM is probability based. This means that EM computes the probability that a point belongs to a cluster, whereas K-means assigns each point to one cluster. Therefore, EM allows for the identification of outliers. The EM algorithm works by keeping track of an average, a covariance matrix, and a probability for each cluster. It also keeps track of the probability of a point to be in a cluster. It then iteratively computes the probabilities of the points, and then uses this to compute the attributes of a cluster. This continues until some stopping requirements are met, in my algorithm that the cluster averages vary by less some constant between iterations. The probabilities are computed by computing the product of the probability of a cluster with the multivariate Gaussian distribution for a cluster and a point. Then these values are normalized for each point over all clusters. The attributes of the clusters are weighted by the point's probabilities. As with K-means, the initialization of the data is important. My technique is to select one point for each cluster using the same method as K-means, by using the similarity of a point to the closest point already selected as the probability for selecting. Then the probability of each point is set to one for the cluster it is closest to, and zero for the rest. Then the actual averages, covariance, and probabilities are computed. In order to improve the results of the algorithm, just as with K-means, it is run multiple times and the "best" one is returned. The "best" result is computing by using the K-means SSE objective. For EM to work with the K-means SSE objective, a point is considered assigned to the cluster with the highest probability.

The advantage of the EM algorithm in a visual sense is that the probabilities can be used to assign each point with a unique color. In most cases the color will be too similar to be differentiated, border cases will still stand out. As shown below, outliers are easy to identify by making use of the probabilities in the coloring of each point:



As you can see some lines are colored a purple, because they are between the blue and red cluster. It is also useful, because there are no points which are yellowish; they are either yellow or not yellow. This shows the yellow cluster is clearly defined.

Qb.js

Qb.js is a data exploration tool written in JavaScript that uses D3 and jQuery that allows for simple exploration of data stored in RDF datacubes. The data is stored as an RDF with a specific vocabulary and retrieved using a SPARQL query. The data is then visualized using D3. I extended this to allow for more than one

type of visualization. Making using that in JavaScript functions are first-class values, I added a way to supply the framework with a visualization function, as well some options for the visualization function. Since JavaScript is a dynamic language, there is no way to guarantee the input of the function, but it is assumed to be of the same form as those which I have written. The options are also modified to indicate that the source of the data is from qb.js, so that it knows how to convert the data from the form it is in qb.js to the form it desires.

This was done, because it allows for easy extension and is very flexible. If another visualization function was to be written and it was written carefully, then it should require no work to allow qb.js to use it to visualize its data. That being said, the author must make sure their input is what qb.js expects, and that they take into account the form of the data given to the function by qb.js. This may constrain the author, but is much easier than requiring integration with qb.js.

I have not fully implemented this due to problems with trying to run qb.js on my machine. I am able to get qb.js to pass my functions the data, but I cannot convert the form of the data as of now, because qb.js the data which qb.js returns is full of bad data, undefined and empty strings.

Conclusion

This last semester I wrote a variety of functions which would allow for rapid creation of customized visualization. They accomplished this by accepting a JavaScript object full with options for the visualization. These options are not completely comprehensive, but allow for a wide variety of possible visualizations. These could very easily be used as a prototype visualization or as a quick and easy

debugging tool. The visualizations also incorporated a variety of analysis tools such as clustering and least-squares regression fitting. These tools give the visualization more usefulness, and can help identify patterns which are obscured by the size of the data. They also quantify relationships of the data, instead of leaving the user to determine them for themselves. The visualizations also relied on a minimal relationship with the user which meant that they would work in most cases. I also created a minimal integration with these functions and qb.js which gave qb.js the power to visualize its data in a multitude of ways. The framework is given the visualization function and some options, and then the visualization function takes the data from qb.js and creates the visualization.