CS 6630 Project Journal

Beginning 11-4-2015

Curtis Miller and Jignesh Rawal

Masters of Statistics and Masters of Computer Science

Index

Contents

PoliVis Project Proposal

Entry by Curtis Miller

0.1 Background and Motivation

The U.S. Congress is a complex entity, with 100 senators and 435 members of the House of Representatives, or 535 members total. While political party is certainly a very important factor in terms of how an individual congressman will vote, it is not a purely determining factor as in other countries where parties often vote as blocks and legislators merely occupy a seat for their political party. Congressmen, especially in the Senate, often act individually, and taking a position contrary to the party's position is not unheard of (although Congress is becoming more polarized and crossing party lines is less common). Thus, the political structure of Congress is much more complex than simply which party holds the majority.

This project aims to visualize congressional voting patterns in terms of how similarly congressmen vote. If two congressmen tend to vote the same on bills or tend to cosponsor the same legislation, they may be considered "similar". When thinking about relationships between members of Congress in this way, we would like to visualize these relationships. This may be useful to politically active individuals and organizations such as lobbyists or lobbying groups and firms (where I worked as an intern for over a year), where determining a strategy often requires understanding these kinds of relationships.

0.2 Project Objectives

The project's overarching goal is to allow exploratory analysis of congressional relationships. A user of the app we create would be able to select a congressman or group of congressmen (say, a state's congressional delegation, members of a particular committee, or individuals who vote "Yay" in favor of a certain bill). The app would then use some visual idiom to show how that congressman (or the group) relates to the other members of Congress and show how other members are "similar" to the

selected individual or group. This could be used to answer a number of questions, such as:

- If an individual congressman is about to sponsor a bill, who are other congressman who may cosponsor it?
- Does a particular delegation (say, the Utah delegation) tend to vote similarly to the delegations from Utah neighbors? In other words, do we see strong regional voting blocks?
- Is Congress becoming more polarized? Do we see less crossover than we once saw before?
- Can we identify any "maveriks"? That is, can we find individuals who are largely marginalized? On the flip side, can we find individuals who vote very similarly to their party or the Congress as a whole?

These are interesting relationships and questions that the app created in this project could discover and answer.

0.3 Data

The Library of Congress website, along with the websites for the U.S. Senate and the House of Representatives, contain voting and sponsorship records. These are usually HTML or XML documents. The roll call votes are XML documents, so they are easier to read and process. Prof. Lex has directed us to a similar project done in a class at Harvard University that may already provide scrapers and data for use.

0.4 Data Processing

The data in the XML files would need to be read and processed. Also, it may be more efficient to process the data and creating the data structures used to calculate "similarity" (which, between two members of Congress, is the probability another member of Congress votes similarly to the selected member, or in another view, sponsor's the same legislation) prior to displaying that data visually. In other words, the relationships likely will not be determined dynamically, but from a file providing the relationships in a way that can be easily processed when the app is online.

0.5 Visualization Design

The visualization design needs to show the chance any legislator votes similarly to the selected legislator. It needs to show this at the regional level and in some other idiom. My visualization includes an (interactive) map and an (interactive) scatterplot, where the x-axis represents congressional members' ideological position (as determined by the DW-NOMINATE score, which measures ideological extremity) and percentage of times of agreement is the y-axis. The x-axis scale is actually an adjustable power scale that allows for the user to make differences appear more or less extreme on the ends, which allows for easier identification of outliers.

The design of the visualization is attached to this document.

0.6 Must-Have Features

The features this app must have include:

- Roll call voting data for the Senate for the 114^{th} Congress.
- A scatterplot representing the percentage of time congressmen vote with the selected individual or group versus their ideological score.
- A map that shows via luminosity how frequently a congressman (or group) agrees with state delegations across the country, with hue indicating with which party this agreement frequently occurs (it is a continuous scale between blue and red).
- The ability to select a congressman from a side list, the scatterplot, or a congressional delegation from the map.

0.7 Optional Features

Optional features include:

- Roll call voting data for the House of Representatives (this is more complex because the House has many more members).
- Ability to look at patterns for different Congresses (in other words, the ability to look back in time).
- A power scale slider that allows for exageration of differences, making outliers easier to spot.

- Name, state, party, committee, tenure, and bio information for a selected legislator.

- When no one is selected, the visualization shows who is often voting on the winning side of issues.

- An additional visualization that shows who frequently cosponsors legislation sponsored by the selected congressman (this is difficult to do because the data is difficult to obtain, requiring scraping, and it is difficult to apply this when more than one congressman is selected).

0.8 Project Schedule

The data should be collected and processed by November 6th. The visualizations should be in a working state by November 20th. Sliders and selectivity should be completed by November 27th, and the overall app structure should be completed by December 4th. At this point, the app will be completed and ready to be deployed. Optional features may be added after the required features are present.

Wednesday, 4 November 2015

Entry by Curtis Miller

We have made some progress on downloading the data and are a step closer to making the data useable. I would like to move forward on soon being able to making a useable JSON file. Another thing we could do today is create some of the files (HTML, JSON, JSS, etc.) that will compose the final visualization, along with files that, while not a part of the final project, will allow us to test whether certain components are working. For example, one test HTML file will load the data. Another may perform some of the calculations and processing necessary in the final visualization. We may add more of these as necessary.

1 Data Acquisiton, Processing, and Preparation

Today we made progress in data acquisition and making the data useable so it may eventually be loaded into the visualization. We have downloaded data and

1.1 Data Acquisition

Jignesh Rawal downloaded the data from the U.S. Senate website and created spreadsheets containing the necessary information. The data exists on the Senate website as XML files, so presumably he downloaded it and somehow parsed it into a spreadsheet. The data on the sheets "vote" and "Vote 296&295" appear to be the most useful at this point in time. The data on "vote" contains the results of votes that will be used for a default display when no Senators have been selected in the visualization. "Vote 296&295" has individual votes on a particular roll-call vote that will be needed. Right now, this is not in a format that I imagine the end data being in, which is effectively a two-dimensional array or matrix, but I could read the data in these sheets into R and would then be able to turn it into a matrix and then the JSON file that I want. However, I am not sure whether JSON would allow the data to be processed like I would in R, so it's possible that a matrix is not the best way to handle it.

1.2 Data Pre-Processing and Preparation

Perhaps a better structure for the end JSON file used in the visualization would be as follows:

```
{
  "senators":[
    {
      "name":"Ayotte",
      "state":"NH",
      "party":"R",
      "votes":[
        "Yea",
        "Nay",
        "Yea",
        "Not Voting",
        ...
        "Yea"
      ]
```

```
    },{
      "name":"Alexander",
      ...
    }, ...
    },{
      "name":"Wyden",
      ...
    }
  ],
  "winning_vote":[
    "Yea",  // Not how the result is actually
        recorded (could be "Passed" or "Confirmed")
    "Nay",
    "Nay",
    ...
    "Nay"
  ]
}
```

This data format appears to be a format that would allow for easier computation in JavaScript than if it were a two-dimensional array. When a senator is selected, their object in this data object is added to the object holding all selected senators. Then, the function that computes similarity would iterate through all senators *not* in the selection and compute similarity based on the array `"votes"`.

Getting the data from the spreadsheet format into a format like this would require loading the data into R. Hopefully I can write an R function soon that does this, while Jignesh Rawal gets more data and processes it.

Tuesday, 10 November 2015

Entry by Curtis Miller

1 Data Acquisiton, Processing, and Preparation

1.1 Data Pre-Processing and Preparation

Jignesh Rawal has loaded most of the data into the online spreadsheets. Now the data must be processed and turned

into a useable format. Since I know R, I will try to use it to create a JSON file with the data in the above format.

I use the following R code.

```r
# install.packages("rjson")
library(rjson)

# Read in Senate voting record; based on the sheet Vote 296&295
read.csv("SenateVote114.csv") -> senate.vote
attach(senate.vote)
# Constructing unique name, state, and party pairing
senators <- last_name[1:100]
home_state <- state[1:100]
party_member <- party[1:100]

# Getting votes on issues
record <- t(sapply(senators, function (sen) {
  return(sapply(unique(vote_number), function (vote) {
    return(vote_cast[which(last_name == sen & vote_number == vote)])
  }))
}))

# Create list data object that will become the JSON string
data.obj <- lapply(1:100, function (i) {
  return(list(
    name = senators[i],
    state = home_state[i],
    party = party_member[i],
    votes = record[i,]
    ))
})
names(data.obj) <- senators

# Create metadata
meta.obj <- list("senators" = senators, "votes" = unique(vote_number))
detach(senate.vote)

# Read in results of votes; based on sheet vote
read.csv("SenateResults114.csv") -> senate.res
attach(senate.res)
levels(result) <- c("Yea", "Yea", "Yea", "Nay", "Yea")
data.obj$winning_vote = result

# Convert to JSON string
```

```
toJSON(data.obj) -> json.string

# Save string as json file
json.file <- file("SenateRecord114.json")
write(json.string, json.file)
close(json.file)
```

This code produces the JSON file that is actually used in the applet, along with associated metadata (senators, states, parties, votes). In the process, the structure of the data changed somewhat from what I originally envisioned. Below I describe the new JSON structure.

```
{
  "senators":[
    "Ayotte":{
      "name":"Ayotte",
      "state":"NH",
      "party":"R",
      "votes":[
        "Yea",
        "Nay",
        "Yea",
        "Not Voting",
        ...
        "Yea"
      ]
    },
    "Alexander":{
      "name":"Alexander",
      ...
    }, ...
    },
    "Wyden":{
      "name":"Wyden",
      ...
    }
  ],
  "winning_vote":[
    "Yea",  // Not how the result is actually
       recorded (could be "Passed" or "Confirmed")
    "Nay",
    "Nay",
    ...
    "Nay"
  ]
}
```

2 Application Interface

I have created the files that will basically serve as a template going forward. I downloaded the HTML5 Boilerplate to use as a start for the site, and going forward we can build off these templates to create the applet.

Jignesh Rawal has started to work on creating the basic JavaScript functions for loading in the data. He unfortunately cannot use git, so he will send me the functions (or at least their basic logic) via e-mail, and I will add them to the git repository where the files are being kept.

Once the data is loaded in, I can start working on functions that perform the necessary computations on the data that will be displayed. If I start by making these functions create dummy data when called, Jignesh Rawal and I can fork responsibilities; he can start working on creating the visualizations (or a basic framework) while I work on the code that gets the actual calculations from the data.

Formulae

Functions and Algorithms