CS51 Final Project Writeup

Neel Mehta, Saranya Vijayakumar, Samuel Lam, Ajay Nathan

Brief Overview

A machine learning classifier that assigns scores to bills and representatives so as to calculate the probability that representatives will vote yes on a given bill. Data for the project will come from Quorum Analytics, of which Ajay is an employee.

Feature List

Core Features

Given a bill, its populairity, and how liberal or conservative it is, our project will be able to calculate the probability that a politican will vote yes on the bill. A graphical user interface to allow users to select a politican and manually input a bill.

Cool Extensions

- Giving users the ability to select actual bills currently in Congress and use Quorum data to accurately determine the popularity of the bill and how liberal or conservative it is.
- Filtering by a bill's issue area. That is, we could classify members of Congress differently depending on what issue's the bill is talking about.
- Adding complexity and therefore accuracy to the classifier. Quorum Analytics has calculated over 100 different statistics about every member of Congress, and we can pick one or more of these to add to the classifer.

Technical Specs

Implementation

The backend algorithms will be written in Python, which is very well-suited for mathematical tasks and machine learning. Furthermore, its functional features will help us build and test our algorithm; we will take advantage of value-based programming, immutable (and hence easily testable) data structures, and bonuses like map() and fold().

We will use the lightweight Flask framework to present a simple RESTful API, including endpoints such as

- PUT lawmaker data
- PUT bill data

- GET list of lawmaker and information about each
- GET list of bills and information about each
- GET probability that a given politician will vote yes on a given bill (this will call our algorithm)

We will use standard HTML, CSS, and JavaScript to implement a simple frontend that allows users to choose politicians and bills and then run the algorithm.

We will deploy the app using Heroku.

Modeling & The Ideal Points Algorithm

Building model

Each lawmaker *i* has an ideal point X[i] that represents where they stand on the left/right spectrum.

Each bill *d* has a popularity (difficulty) B[d], which reflects how overall attractive the bill is, and a polarity (discrimination) A[d], which reflects where the bill is on the left/right axis. When popularity B[d] is high (very positive), most lawmakers will vote yes. When popularity B[d] is low (very negative), most lawmakers will vote no. When popularity B[d] is near zero, the polarity A[d] and the individual lawmaker's ideal point X[i] determine how the lawmaker will vote.

X[i], A[d], and B[d] are usually distributed according to the standard normal distribution N(0, 1).

The signs of X[i] and A[d] are arbitrary as long as they are consisent (positive can indicate liberal and negative conservative, or vice versa.) We will see that, mathematically, it only matters if the signs of X[i] and A[d] are the same or different.

We can use this information to predict if the lawmaker will vote yes:

```
P(yes)[i,d] = S(X[i]A[d] + B[d])
```

where S(x) = the logistic function `(e^x)/(1 + e^x)`. S(x) tends toward 0 as x approaches -Infinity and tends toward 1 as x approaches +Infinity. This distribution is useful because it allows for outliers.

We are also given C[i,d], where

```
C[i,d] =
    1 if lawmaker i actually voted yes on bill d
    0 if they voted no
```

We can now compare the predicted outcomes P(yes)[i,d] to the actual outcomes C[i,d] to

determine the accuracy of the model.

Testing model

For each pairing of lawmaker and bill, we can determine the error score E[i,d], which simply measures if the prediction was right or wrong.

```
E[i,d] =
    1 if P(yes)[i,d] < 0.5 and C[i,d] = 1 (yes)
    1 if P(yes)[i,d] > 0.5 and C[i,d] = 0 (no)
    0 otherwise
```

Essentially, we say the model erred if it predicted a better-than-average chance that the lawmaker would vote yes when they actually voted no, or the reverse.

Now, since E[i,d] is simply whether the prediction was right or wrong, a P[i,d] close to 0.5 is more likely to be wrong than a more extreme P[i,d].

But we can control for this by introducing the expected error rate R[i,d], which is defined as

```
R[i,d] = min(P(yes)[i,d], 1 - P(yes)[i,d])
```

Note that E[i,d] is either 0 or 1, while R[i,d] is any real number between 0 and 1, inclusive.

As mentioned above, we want to determine a consistent measure of error, which we do by defining an excess error EE[i,d]:

```
EE[i,d] = E[i,d] - R[i,d]
```

The expected/average value of EE[i,d] is 0, so finding the average EE[i,d] for all lawmakers *i* and bills *d* is an informative measure of the error in the model.

Machine learning

We're investigating possible machine learning algorithms to use to determine the optimal X[i] for every lawmaker *i* and the optimal A[d] and B[d] for every bill *d*. These include:

- Naive Bayes / Kernel approximation method i.e. RBF: seems useful but we aren't sure how we would apply it to our specific problem.
- Clustering with k-means: make a 4-dimensional space (X[i], A[d], B[d], C[i,d]) and find clusters in this space.
- Regression: simply tweak the parameter values based on whether the prediction was

right or wrong.

Next Steps

By next checkpoint:

- Learn Python
- Set up Flask and Heroku development environments
- Pick a machine learning algorithm
- Get the data from Quorum
- Make mockups of frontend pages

References

- How They Vote: Issue-Adjusted Models of Legislative Behavior
- Practical Issues in Implementing and Understanding Bayesian Ideal Point Estimation
- Topic-Factorized Ideal Point Estimation Model for Legislative Voting Network