politics lab

Coding the Matrix, Summer 2013

Please fill out the stencil file named "`politics_lab.py`". While we encourage you to complete the Ungraded Problems, they do not require any entry into your stencil file.

Comparing Voting Records using Dot-Product

In this lab, we will represent a US senator's voting record as a vector over $\mathbb{R}$, and will use dot-products to compare voting records. For this lab, we will just use a list to represent a vector.

1 *Motivation*

These are troubled times. You might not have noticed from atop the ivory tower, but take our word for it that the current sociopolitical landscape is in a state of abject turmoil. Now is the time for a hero. Now is the time for someone to take up the mantle of protector, of the people's shepherd. Now is the time for linear algebra.

In this lab, we will use vectors to evaluate objectively the political mindset of the senators who represent us. Each senator's voting record can be represented as a vector, where each element of that vector represents how that senator voted on a given piece of legislation. By looking at the difference between the "voting vectors" of two senators, we can dispel the fog of politics and see just where our representatives stand.

Or, rather, stood. Our data are a bit dated. On the bright side, you get to see how Obama did as a senator. In case you want to try out your code on data from more recent years, we will post more data files on `resource.codingthematrix.com`.

2 *Reading in the file*

As in the last lab, the information you need to work with is stored in a whitespace-delimited text file. The senatorial voting records for the 109th congress can be found in `voting_record_dump109.txt`. Each line of the file represents the voting record of a different senator. In case you've forgotten how to read in the file, you can do it like this:

```
>>> f = open('voting_record_dump109.txt')
>>> mylist = list(f)
```

You can use the `split(·)` procedure to split each line of the file into a list; the first element of the list will be the senator's name, the second will be his/her party affiliation (R or D), the third will be his/her home state, and the remaining elements of the list will be that senator's voting record on a collection of bills. A "1" represents a 'yea' vote, a "-1" a 'nay', and a "0" an abstention.

Task 1: Write a procedure `create_voting_dict(strlist)` that, given a list of strings (voting records from the source file), returns a dictionary that maps the last name of a senator to a list of numbers representing that senator's voting record. You will need to use the built-in procedure `int(·)` to convert a string representation of an integer (e.g. '1') to the actual integer (e.g. 1).

3 *Two ways to use dot-product to compare vectors*

Suppose $\boldsymbol{u}$ and $\boldsymbol{v}$ are two vectors. Let's take the simple case (relevant to the current lab) in which the entries are all 1, 0, or -1. Recall that the dot-product of $\boldsymbol{u}$ and $\boldsymbol{v}$ is defined as

$$\boldsymbol{u} \cdot \boldsymbol{v} = \sum_k \boldsymbol{u}[k]\boldsymbol{v}[k]$$

Consider the k^{th} entry. If both $\boldsymbol{u}[k]$ and $\boldsymbol{v}[k]$ are 1, the corresponding term in the sum is 1. If both $\boldsymbol{u}[k]$ and $\boldsymbol{v}[k]$ are -1, the corresponding term in the sum is also 1. Thus a term in the sum that is 1 indicates agreement. If, on the other hand, $\boldsymbol{u}[k]$ and $\boldsymbol{v}[k]$ have different signs, the corresponding term is -1. Thus a term in the sum that is -1 indicates disagreement. (If one or both of $\boldsymbol{u}[k]$ and $\boldsymbol{v}[k]$ are zero then the term is zero, reflecting the fact that those entries provide no evidence of either agreement or disagreement.) The dot-product of $\boldsymbol{u}$ and $\boldsymbol{v}$ therefore is a measure of how much $\boldsymbol{u}$ and $\boldsymbol{v}$ are in agreement.

4 *Policy comparison*

We would like to determine just how like-minded two given senators are. We will use the dot-product of vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ to judge how often two senators are in agreement.

Task 2: Write a procedure `policy_compare(sen_a, sen_b, voting_dict)` that, given two names of senators and a dictionary mapping senator names to lists representing voting records, returns the dot-product representing the degree of similarity between two senators' voting policies.

Task 3: Write a procedure `most_similar(sen, voting_dict)` that, given the name of a senator and a dictionary mapping senator names to lists representing voting records, returns the name of the senator whose political mindset is most like the input senator (excluding, of course, the input senator him/herself).

Task 4: Write a very similar procedure `least_similar(sen, voting_dict)` that returns the name of the senator whose voting record agrees the least with the senator whose name is `sen`.

Task 5: Use these procedures to figure out which senator is most like Rhode Island legend Lincoln Chafee. Then use these procedures to see who disagrees most with Pennsylvania's Rick Santorum. Give their names.

Ungraded Task: How similar are the voting records of the two senators from your favorite state?

5 *Not your average Democrat*

Task 6: Write a procedure `find_average_similarity(sen, sen_set, voting_dict)` that, given the name `sen` of a senator, compares that senator's voting record to the voting records of all senators whose names are in `sen_set`, computing a dot-product for each, and then returns the average dot-product.

Use your procedure to compute which senator has the greatest average similarity with the set of Democrats (you can extract this set from the input file).

In the last task, you had to compare each senator's record to the voting record of each Democrat senator. If you were doing the same computation with, say, the movie preferences of all Netflix subscribers, it would take far too long to be practical.

Next we see that there is a computational shortcut, based on an algebraic property of the dot-product: the *distributive* property:

$$(\boldsymbol{v}_1 + \boldsymbol{v}_2) \cdot \boldsymbol{x} = \boldsymbol{v}_1 \cdot \boldsymbol{x} + \boldsymbol{v}_2 \cdot \boldsymbol{x}$$

Task 7: Write a procedure `find_average_record(sen_set, voting_dict)` that, given a set of names of senators, finds the average voting record. That is, perform vector addition on the lists representing their voting records, and then divide the sum by the number of vectors. The result should be a vector.

Use this procedure to compute the average voting record for the set of Democrats, and assign the result to the variable `average_Democrat_record`. Next find which senator's voting record is most similar to the average Democrat voting record. Did you get the same result as in Task 6? Can you explain?

6 *Bitter Rivals*

Task 8: Write a procedure `bitter_rivals(voting_dict)` to find which two senators disagree the most.

This task again requires comparing each pair of voting records. Can this be done faster than the obvious way? There is a slightly more efficient algorithm, using *fast matrix multiplication*. We will study matrix multiplication later, although we won't cover the theoretically fast algorithms.

7 *Open-ended study*

You have just coded a set of simple yet powerful tools for sifting the truth from the sordid flour of contemporary politics. Use your new abilities to answer at least one of the following questions (or make up one of your own):

- Who/which is the most Republican/Democratic senator/state?
- Is John McCain really a maverick?
- Is Barack Obama really an extremist?
- Which senator has the most political opponents? (Assume two senators are opponents if their dot-product is very negative, i.e. is less than some negative threshold.)