GSMST

Applications of Linear Algebra in Programming

Senator Voting Records Lab

Submitted By:
Anish Goyal
4th Period

Submitted To:
Mrs. Denise Stiffler
Educator

February 15, 2023

Table of contents

2.12 Lab: Comparing voting records using dot-product

2.12.2 Reading in the file

Task 2.12.1

Write a procedure `create_voting_dict(strlist)` that, given a list of strings (voting records from the source file), return a dictionary that maps the last name of a senator to a list of numbers representing that's 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).

```
list1 = []
with open("voting_record_dump109.txt", 'r') as data:
  for line in data:
    data = line.split()
    list1.append(data)

def create_voting_dict(strlist): return {strlist[x][0]: [int(val) for
↪   val in strlist[x][3:]] for x in range(len(strlist))}

voting_dict = create_voting_dict(list1)
```

2.12.4 Policy comparison

Task 2.12.2

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.

```
def policy_compare(sen_a, sen_b, voting_dict): return
↪   sum([voting_dict[sen_a][i] * voting_dict[sen_b][i] for i in
↪   range(len(voting_dict[sen_a]))])
```

Task 2.12.3

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).

```python
def most_similar(sen, voting_dict): return max([(other_sen,
↪ policy_compare(sen, other_sen, voting_dict)) for other_sen in
↪ voting_dict.keys() if other_sen != sen], key=lambda x: x[1])[0]
```

Task 2.12.4

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`.

```python
def least_similar(sen, voting_dict): return min([(other_sen,
↪ policy_compare(sen, other_sen, voting_dict)) for other_sen in
↪ voting_dict.keys() if other_sen != sen], key=lambda x: x[1])[0]
```

Task 2.12.5

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

```python
print(most_similar('Chafee', voting_dict))
print(least_similar('Santorum', voting_dict))
```

```
Jeffords
Feingold
```

Task 2.12.6

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

```python
print("As a dot product: " + str(policy_compare('Sessions', 'Shelby',
↪ voting_dict)))
print("As a percent: " + str(policy_compare('Sessions', 'Shelby',
↪ voting_dict)/len(voting_dict['Sessions'])*100) + "%")
```

```
As a dot product: 38
As a percent: 82.6086956521739%
```

2.12.5 Not your average Democrat

Task 2.12.7

Write a procedure `find_average_similarity(sen, sen_set, voting_dict) that, given the namesenof 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).

```python
def find_average_similarity(sen, sen_set, voting_dict): return
↪   sum([policy_compare(sen, other_sen, voting_dict) for other_sen in
↪   sen_set]) / len(sen_set)

democrats = {list1[x][0] for x in range(len(list1)) if list1[x][1] ==
↪   'D'}
most_similar_democrat = max([(sen, find_average_similarity(sen,
↪   democrats, voting_dict)) for sen in voting_dict.keys() if sen in
↪   democrats], key=lambda x: x[1])

print(most_similar_democrat)
```

```
('Biden', 34.86046511627907)
```

Task 2.12.8

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 Task 2.12.7? Can you explain?

```python
def find_average_record(sen_set, voting_dict):
    count = len(sen_set)
    return [sum(col) / count for col in zip(*(voting_dict[sen] for sen
     ↪  in sen_set))]

average_democrat_record = find_average_record(democrats, voting_dict)
voting_dict['average'] = average_democrat_record
most_similar('average', voting_dict)
```

```
'Biden'
```

These results are the same for the following reason:
The max of all senator dot product averages is max $\left(\frac{\sum_i^N s \cdot s_i}{N}\right)$.
while the max of the dot product of all senators with the average senator is max $\left(\left(\frac{\sum_i^N s_i}{N}\right) \cdot s\right)$,
both of which are equal due to the Distributive Property of Vectors.

2.12.6 Bitter Rivals

Task 2.12.9

Write a procedure `bitter_rivals(voting_dict)` to see which two senators disagree the most.

```python
def bitter_rivals(voting_dict):
    min_sim = float('inf')
    names = list(voting_dict.keys())
    num_names = len(names)
    rival_pair = None
    for i in range(num_names-1):
        for j in range(i + 1, num_names):
            sim = policy_compare(names[i], names[j], voting_dict)
            if sim < min_sim:
                min_sim = sim
                rival_pair = [names[i], names[j]]
    return [rival_pair, min_sim]

print(bitter_rivals(voting_dict))
```

```
[['Feingold', 'Inhofe'], -3]
```