CMSC B113 - Computer Science 1
Fall 2020
Homework Assignment #3

Overview

In this assignment, you will continue in your role of a political campaign consultant who has been hired by a candidate to help them get a sense of how likely they are to win an upcoming election. To do this, you will write two Java programs that perform calculations and simulations related to the election.

For each program, first read the problem, then devise a solution. Clearly identify all the inputs and outputs. Write down the steps and calculations that will need to be performed. Code the steps into a Java program. Create, compile and run the program and test it on several inputs. Double-check the program's answers with the ones we have provided to confirm that the results are correct.

Note that in this assignment, you will also be asked to write up a brief report regarding your findings and observations. Be sure to allocate time for doing that write-up, in addition to programming and testing your solution.

Learning Outcomes

In completing this assignment, you will learn to:

- Use if-else statements in a Java program
- Use for-loops in a Java program
- Use arrays to store and process data
- Use Math.random() to simulate events
- Write code that performs data validation on runtime arguments

Part 1: Poll-based Prediction - Predictor

In this task, we will write a program, called **Predictor**, that predicts a candidate's likelihood of victory by taking into account the current poll and its margin of error. For example, if a poll predicts that a candidate has a 47% chance of winning a state with a 6% margin of error, can we make an overall prediction that the candidate will win a state? What if the polls could be affected by running an ad campaign? Etc.

One way to do this is to simulate an actual vote incorporating the margin of error.

Let's say, a poll (POLL) in a state shows that a candidate is likely to receive 47% of the vote in their favor with a margin of error +/- 6% (MOE). With this information, we can do a simulation of the state's voting, and arrive at the % of votes received by the candidate in a poll:

SamplePoll = POLL + X * MOE

Where X is a number in the range [-1.0..1.0]. For example, if X is -0.5, then the candidate will receive 0.47 + -0.5 * 0.06 = 0.44. This means, in this instance, the candidate received only 44% of the votes (i.e. losing the state).

Alternately, if X turns out to be 0.8 then the candidate will receive 0.47 + 0.8 * 0.06 = 0.518 or 51.8% of the votes cast thereby winning the state.

We can use Java's **Math.random()** function to generate values of X in the range [-1.0..1.0] and perform a simulation of several sample elections based on POLL and MOE to see if a candidate is likely to win or lose a state. For example,

```
$ Predictor Keystone 0.47 0.06 1000000
Candidate win likelihood for Keystone state is 25.02%
```

The above means that after performing 1 million randomized simulated polls based on POLL and MOE the candidate has a 25% likelihood of winning Keystone state. Here are some more examples.

```
$ Predictor Keystone 0.53 0.06 1000000
Candidate win likelihood for Keystone state is 74.99%

$ Predictor Golden 0.49 0.07 1000000
Candidate win likelihood for Golden state is 42.82%

$ Predictor Nutmeg 0.51 0.02 1000000
Candidate win likelihood for Nutmeg state is 75.01%
```

Write a complete Java program, called **Predictor**, that performs the calculations above. Here is the basic structure of such a program (*N* is the number of simulated polls. 1 million is examples above):

Input: state, POLL, MOE, N

wins ← 0
do N times
    Compute SamplePoll ← …
    If SamplePoll > 0.5 then
        wins ←wins + 1
Compute and output overall win likelihood (wins/N)

For your sample outputs, run the program for N = 10 million and the following polls:

STATE	Poll	Margin Of Error
Garden	49%	3%
Empire	52%	2%
Nutmeg	43%	8%
Golden	51%	5%
LoneStar	48%	4%

Write a short 1-2 paragraph report for the candidate based on your output from Predictor.

What to hand-in for this part:

1. A completed, properly commented and formatted program source code.
2. Three sample runs from Keystone, Golden, and Nutmeg states as shown above.
3. Outputs from the five states in the table above.
4. Your 1-2 paragraph report as described above.

Part 2: Predicting All States - USAPredictor

With a predictor ready for one state, we can now use the logic from Part 1 to do predictions for all states. In this part, you will write a program, called **USAPredictor**, that incorporates the polls from each state, and simulates and outputs the likelihood of a candidate winning (as above). For the sake of simplicity, we will assume that all the polls in a run will have the same margin of error. Your program will be able to input the margin of error for each simulation. Here is an example:

```
$ java-introcs USAPredictor 0.05

               State Polls Likelihood
             Alabama 25.58   0.00
              Alaska 51.74  67.48
             Arizona 54.07  90.75
            Arkansas 39.49   0.00
          California 62.19 100.00
            Colorado 56.96 100.00
                 ...   ...    ...   [output deleted for brevity]
            Virginia 58.74 100.00
          Washington 62.41 100.00
       West Virginia 33.46   0.00
           Wisconsin 60.82 100.00
             Wyoming 34.90   0.00

Margin of Error = 0.05
```

And, another example:

```
$ java-introcs USAPredictor 0.08

               State Polls Likelihood
             Alabama 25.58   0.00
              Alaska 51.74  60.89
             Arizona 54.07  75.46
            Arkansas 39.49   0.00
          California 62.19 100.00
                 ...   ...    ...   [output deleted for brevity]
            Virginia 58.74 100.00
          Washington 62.41 100.00
       West Virginia 33.46   0.00
           Wisconsin 60.82 100.00
             Wyoming 34.90   0.00

Margin of Error = 0.08
```

The data being used is derived from a poll for a previous presidential election in the United States. As you can clearly see from the above, the candidate's victory in some states is guaranteed (100.0% likelihood) and a loss in some (0.00% likelihood). Also, notice how if the margin of error varies, the likelihood changes. So, the quality of the polls matter!

The data containing the states and polls for each state (for one candidate) from the election has been made available to you, preloaded in two Java arrays, `states[]` and `polls[]`, in the file `USAPredictor.java` (download this file from the class webpage). Examine the way the data is arranged. There are 51 entries in each array (50 states, plus District of Columbia). You can do the simulation as follows:

Input: MOE
N ← 1000000

for each state do
    wins ← 0
    do N times
        Compute SamplePoll ← …
        If SamplePoll > 0.5 then
            wins ←wins + 1
    Compute and output state, poll, and overall win likelihood (wins/N)

Note: Take a close look at the poll data in `USAPredictor.java` file (`polls[]` array). It is in percentages. Make sure you divide each by 100.0 when you use it in computing *samplePoll*.

Make sure your program outputs the results as shown above. Show your results for MOE = 0.5 and 0.8.

The output produced by USAPredictor can be used by the campaign to make decisions the number and kinds of ad campaigns to run in each state. We will address that next.

What to hand-in for this part

1. A completed, properly commented and formatted program source code.
2. A complete output from each of the two runs requested above (with MOE = 0.05, and 0.08).
3. A short (1-3 sentence) analysis of your interpretation of results.

Part 3: Optimizing Spending - AdSense

Your next task is to help your candidate determine how to allocate their campaign budget to gain the highest number of votes among the SWING voters described in Assignment#2.

Your candidate has $1,000,000 to spend among three different types of advertisements:

- **POSITIVE** ads are those that convey a positive message about your candidate
- **ATTACK** ads are those that attack your opponent
- **ISSUE** ads are those that describe your candidate's stance on an important issue

POSITIVE, ATTACK, and ISSUE represent the amount of dollars (or a percent of ad spending budget). I.e. If the ad budget is $1,000,000 then

POSITIVE + ATTACK + ISSUE = $1,000,000

Your campaign strategists have determined that the number of votes that your candidate will gain as a result of spending is

(POSITIVE * 0.0005) + (POSITIVE * ATTACK * ISSUE * 0.0001)

But, due to the negative nature of attack ads, the number of votes that your candidate will lose is

(ATTACK * 0.0009)

Your job is to determine the amount of money to spend on each type of ad (POSITIVE, ATTACK, ISSUE) in order to maximize the total number of votes gained, i.e.

(POSITIVE * 0.0005) + (POSITIVE * ATTACK * ISSUE * 0.0001) – (ATTACK * 0.0009)

Determining the actual maximum value is quite difficult, and trying all possible combinations of spending amounts would be quite time consuming, so you decide to solve this by choosing sets of random numbers for POSITIVE, ATTACK, and ISSUE and seeing which set leads to the highest value. Although this may not be the true maximum value, if you try enough random sets, it should get you close.

Your approach is as follows:

- Set POSITIVE to a random int between 0 and 1,000,000, inclusive
- Set ATTACK to a random int between 0 and (1,000,000 – POSITIVE) inclusive, i.e. the amount of money left over
- Set ISSUE to 1,000,000 – POSITIVE – ATTACK
- Calculate the number of votes gained using the formula above
- If this number is higher than the highest number of votes gained seen so far, keep track of these values of POSITIVE, ATTACK, and ISSUE (this is the amount of $1000000 you will spend on ads in each category).

After calculating the number of votes gained for 10,000 random combinations of POSITIVE, ATTACK, and ISSUE, print the values of the combination that led to the highest number of votes gained. This tells the candidate how they should allocate money toward the three types of ads.

Below, we provide an outline of the method to do the computation:

Repeat 10,000 times:
   POSITIVE = Random(0, 1000000)
   ATTACK = Random(0, 1000000 – POSITIVE)
   ISSUE = 1000000 – POSITIVE – ATTACK
   votesGained = …
   if votesGained > maxVotesGained
      maxPositive = POSITIVE
      maxAttack = ATTACK
      maxIssue = ISSUE
      maxVotesGained = votesGained
print maxPositive, maxAttack, maxIssue

Write a complete Java program called **AdSense** that performs the above computations. Once done, run it several times and observe the amounts to spend in each ad category. You will notice that the amount to spend in each category fluctuates quite a bit. Why?

When you are doing randomized trials, as you have seen in similar programs in class, performing more trials leads to convergence. That is, instead of 10,000 trials, what if you did 100,000? Or 1,000,000? Or even 10,000,000?

Modify your AdSense program to input the number of trials to be performed so you can run a different number of trials. For example,

```
$ java-introcs AdSense 10000

Max Positive = $747803
Max Attack = $235978
Max Issue = $16219
```

Or, to do more trials

```
$ java-introcs AdSense 1000000

Max Positive = $980047
Max Attack = $5494
Max Issue = $14459
```

As you increase the number of trials you will notice consistent convergence. If in addition of printing out exact dollar amounts, you print out the percentage of your spending budget for each category, you will quickly observe this convergence (see example below).

```
$ java-introcs AdSense 1000000

Max Positive = $980047 0.980047%
Max Attack = $5494     0.005494%
Max Issue = $14459     0.014459%
```

What is it? What if the impact of attack ads is losing 0.05% of voted (instead of 0.01% as used above)?

By the way, this approach to optimization is known as a "Monte Carlo Simulation" and it's a real thing! You have seen other examples of this technique in class and in your text. It's a very handy way of trying to maximize the value of an expression by using randomization.

What to hand-in for this part

1. A completed, properly commented and formatted program source code.
2. Your output for running the simulation for 1 million and 10 million trials, for both losing 0.01% votes for attack ads, as well as losing 0.05% (as suggested above). There will be a total of four sample runs.
3. A short (1-3 sentence) analysis of the interpretation of your results.

Getting Help

Learning how to program can be quite challenging at first and it is completely normal to struggle and run into issues on your first assignment. Don't get discouraged, though! You learn by overcoming that struggle and finding ways to address the issues that you face.

If you find yourself stuck on this assignment and not making progress for, say, 30 minutes or so, then the best thing to do may be to walk away from it for a while and try to think about something else. It often is the case that your brain just needs a rest and that the solution will seem a bit clearer if you look at it with a refreshed mind.

If you're still stuck, though, then don't worry! The CMSC B113 instruction staff is here to support you and to help you do well on this assignment.

The course Teaching Assistants (TAs) are available for drop-in office hours, and the schedule is posted on the course website. You do not need an appointment for these office hours: you can just show up and the TA will help you as soon as they can.

You don't need to wait for office hours, though. You can post a question in Piazza so that the community of students, TAs, and instructors can try to help you, as well. Please be sure not to accidentally post your solution publicly, though! You should at most only be posting 3-4 lines of code or so.

Last, it is also okay to email your instructor, especially with private concerns, though posting in Piazza is likely to be the fastest way to get help, especially if no office hours are being held.

Submitting Your Solutions

To submit your work, copy/paste the source code, along with samples of input/output, of your Java programs into a *single* PDF, and submit the PDF in the "Assignment 3" subfolder of the Dropbox folder that your instructor created for you.

Please be sure to put all of your code into a *single* document and please only submit a PDF, not a Microsoft Word .docx file or the .java source files.