CPS 590.2 - Computation, Information, and Learning in Market Design

Homework 2 (due Feb. 22 before or at the beginning of class)

Please see the rules for homework on the course website, and contact Vince (conitzer@cs.duke.edu) or Michael (malbert@cs.duke.edu) with any questions. To turn in this assignment, please send your `.mod` and `.out` files by e-mail to both of us. Please use clear variable names and write comments in your code where appropriate (you can put comments between `/*` and `*/`, or start a line with `#`).

0. Installing the GNU linear programming kit.

You can find the GNU linear programming kit on the Web (`https://www.gnu.org/software/glpk/`). You are free to install it on your own computer. If you have trouble, below are some instructions for installing glpk on your login.oit.duke.edu user space that might be helpful (but could be a little tedious, so you are encouraged to try installing on your own machine first). If you still have trouble, please let us know. After you have successfully installed everything you are highly encouraged to check out the "examples" directory for examples of how to use the modeling language, as well as the examples from class which are on the course website.

Instructions for getting onto OIT computers

Of course, your first option is to work at a computer on campus. Otherwise, using ssh, login to your account on a Duke OIT machine and do your work there. Here is a good manual on how to do this on Duke's network.

`http://www.cs.duke.edu/~alvy/courses/Remote_Access.pdf`

You will have to work from the command line, but if you don't already know how this is a great time to learn! Here is a good tutorial for some of the basics.

`http://www.cs.duke.edu/~alvy/courses/unixtut/`

It may also be good to work from a command line text editor, like vim or emacs.

Installation Instructions for GLPK

Individual students need to install GLPK in their login.oit.duke.edu user spaces with the following commands:

```
mkdir ~/glpk
cd ~/glpk
wget ftp://ftp.gnu.org/gnu/glpk/glpk-4.57.tar.gz
tar -xzvf glpk-4.57.tar.gz
cd glpk-4.57
./configure
make
```

The program can then be run from the following directory.

`~/glpk/glpk-4.57/examples`

If you want to solve an LP/MIP expressed using the modeling language, navigate to the above directory and type

`./glpsol --math`

You will also need to specify the file that you want to solve, e.g.

`./glpsol --math problem.mod`

and you will also need to specify a name for a file in which the output will be stored, preceded by –output. So, typing

`./glpsol --math problem.mod --output problem.out`

will instruct the solver to solve the LP/MIP problem.mod, and put the solution in a new file called problem.out.

You will need an editor to read and edit files. One such editor is emacs (but any text editor will do). For example, typing

`emacs problem.out`

will allow you to read the output file.

1. Practice with GLPK: Automatically designing an optimal proper scoring rule.

So far, we have not yet defined any quantitative sense in which one proper scoring rule is "better" than another. Here we will define such a sense. At a high level, we want to do two things:

1. We want *strict* incentives for the forecaster to report truthfully.
2. We do not want to spend too much money.

Specifically, assume that we have a budget B that we can spend (and we cannot take money away from the forecaster), so that any score we can give must be in the interval $[0, B]$. Under this constraint, we want to maximize the *worst-case* incentive for the agent to report the truth rather than something false.

As a concrete example, consider a seting with three outcomes $\Omega = \{1, 2, 3\}$ where we consider only three possible distributions: $(3/4, 1/4, 0), (1/2, 1/4, 1/4), (0, 1/2, 1/2)$. (For example, suppose we know the forecaster can only receive one of three possible signals, based on which the forecaster makes the prediction, that result in these posterior distributions.)

Consider the following proper scoring rule: the *spherical* scoring rule, for which $S(\mathbf{p}, \omega) = \frac{p_\omega}{\sqrt{p_1^2+\ldots+p_n^2}}$, where n is the number of outcomes. This rule has the property that scores are always nonnegative, which we want. On the distributions above, it spends at most 0.948683 (which is what it spends if outcome 1 happens and $(3/4, 1/4, 0)$ is reported). For any two distributions $\mathbf{p} \neq \mathbf{p}'$ (among the three listed above) this rule gives at least 0.0589738 incentive to report $\mathbf{p}$ rather than $\mathbf{p}'$ (assuming $\mathbf{p}$ is the true distribution); the incentive is exactly this when $\mathbf{p} = (1/2, 1/4, 1/4)$ and $\mathbf{p}' = (3/4, 1/4, 0)$. That is, $\frac{(1/2)^2+(1/4)^2+(1/4)^2}{\sqrt{(1/2)^2+(1/4)^2+(1/4)^2}} - \frac{(1/2)(3/4)+(1/4)(1/4)}{\sqrt{(3/4)^2+(1/4)^2}} = 0.0589738$.

Now consider the following scoring rule instead (which is only defined for those three distributions):

- $S((3/4, 1/4, 0), 1) = 1, S((3/4, 1/4, 0), 2) = 0, S((3/4, 1/4, 0), 3) = 0$;
- $S((1/2, 1/4, 1/4), 1) = 0.545455, S((1/2, 1/4, 1/4), 2) = 0.636364, S((1/2, 1/4, 1/4), 3) = 1$;
- $S((0, 1/2, 1/2), 1) = 0, S((0, 1/2, 1/2), 2) = 1, S((0, 1/2, 1/2), 3) = 1$.

Note that it uses budget 1. For any two distributions $\mathbf{p} \neq \mathbf{p}'$ (among the three listed above) this rule gives at least 0.181818 incentive to report $\mathbf{p}$ rather than $\mathbf{p}'$ (assuming $\mathbf{p}$ is the true distribution). For example, if the true distribution is $(3/4, 1/4, 0)$, then the incentive for reporting truthfully rather than reporting $(1/2, 1/4, 1/4)$ is $(3/4)(1 - 0.545455) + (1/4)(0 - 0.636364) + (0)(0 - 1) = 0.181818$. In fact, this scoring rule is optimal in this sense (it maximizes the worst-case incentive under the constraint that the budget is 1).

In the MathProg (`.mod`) language, write up a linear program for optimizing proper scoring rules in the sense above. It should start as follows:

```
param m; #the number of distributions
param n; #the number of outcomes
param prob{i in 1..m, j in 1..n}; #the probabilities assigned by
distributions to outcomes
param B; #the budget (maximum payment)
var score{i in 1..m, j in 1..n}, >=0, <=B; #the score function for which we
are trying to solve
```

You may introduce additional variables if this is helpful. Hint: the following

```
{i1 in 1..m, i2 in 1..m: i1!=i2}
```

will generate all pairs of $i1$ and $i2$ where the two are not equal to each other.

It should take data in the following format (the below is the example from before):

```
data;
param m:=3;
param n:=3;
param B:=1;
param prob:  1    2    3 :=
1            .75  .25  0
2            .5   .25  .25
3            0    .5   .5;
end;
```

You should test your linear program formulation on the instance above, and on any other instances that you want to try. We will test your linear program formulation on secret test cases. For this purpose, please turn in an incomplete `.mod` file that ends with `data;` and call it `proper_incomplete.mod`. Also turn in `proper.mod` and `proper.out` where the above instance is included.