UofT CARTE Labatt ML Bootcamp

Lab 3a

Lab author: Alexander Olson, alex.olson@utoronto.ca

In this lab we will look at practical examples of using optimization, with the open source Google OR-Tools toolkit. Let's install that now:

In [1]:
```
!pip install -U ortools
```

```
Requirement already satisfied: ortools in /Users/alex/miniconda3/envs/bootcamp/lib/pytho
n3.9/site-packages (9.3.10497)
Requirement already satisfied: numpy>=1.13.3 in /Users/alex/miniconda3/envs/bootcamp/li
b/python3.9/site-packages (from ortools) (1.22.4)
Requirement already satisfied: absl-py>=0.13 in /Users/alex/miniconda3/envs/bootcamp/li
b/python3.9/site-packages (from ortools) (1.1.0)
Requirement already satisfied: protobuf>=3.19.4 in /Users/alex/miniconda3/envs/bootcamp/
lib/python3.9/site-packages (from ortools) (4.21.1)
```

Optimizing beer production

Let's say that you are trying to decide how much to produce of four different beers - an IPA, a lager, a wheat beer and a pilsner. All these beers use the same basic ingredients, but in different quantities. On top of that, they sell for different prices, and there is a certain amount of demand for each beer that we are don't expect to go over. With all this in mind, we can use integer programming to figure out the optimal amount we should produce of each beer so that our revenue is maximized.

Let's start with some boilerplate creating the model we are going to use:

In [2]:
```
from ortools.linear_solver import pywraplp
# Create the mip solver with the SCIP backend.
solver = pywraplp.Solver.CreateSolver('SCIP')
```

Now we need to take the information about our four beers, and translate it into the format the model understands.

IPA

The IPA uses 3 units of grain, 1 unit of hops, 2 units of yeast and 10 units of water.

It sells for $4.75, and we don't expect to sell more than 300 units.

Lager

The Lager uses 4 units of grain, 2 of hops, 1 yeast and 9 water.

It sells for $5.25 and we expect demand to max out at 400 units.

Wheat Beer

The Wheat Beer uses 1 grain, 1 hops, 6 yeast and 11 water.

It sells for $4.00 and is expected to be popular, with up to 700 units sold.

Pilsner

The Pilsner uses 2 grain, 3 hops, 2 yeast and only 4 water.

It sells for $2.75 and we only expect to sell up to 200 units.

Now that we have laid out this information, we need to go through and input it to the model.

Your Turn

First let's define the variables that track how many units of each beer we are going to produce. Since we only make whole numbers of units, these need to be integer variables. We have an upper and lower limit for each variable - below we have created the variable for the IPA for you. Go ahead and add the other three variables, using the same format as the example.

In [3]:
```
I = solver.IntVar(0.0, 300.0, 'IPA') #0 is the minimum number of units, and 300 is the m
L = solver.IntVar(0.0, 400.0, 'Lager')
W = solver.IntVar(0.0, 700.0, 'Wheat Beer')
P = solver.IntVar(0.0, 200.0, 'Pilsner')
```

Now we need to add our constraints. In this case, we only have a finite amount of each of the ingredients, and we need to decide where best to allocate them. Let's say we have 1500 units of grain, 3000 units of hops, 6000 units of yeast and 10000 units of water.

As above, we have provided the first example - the constraint for grain. This adds up the number of units of grain used across all the beers and says that it cannot be higher than 1500. Go ahead and add similar equations for hops, yeast and water:

In [4]:
```
# Grain: 3 per IPA + 4 per Lager + 1 per Wheat + 2 per Pilsner <= 1500 total units
solver.Add(3*I + 4*L + 1*W + 2*P <= 1500)

solver.Add(1*I + 2*L + 1*W + 3*P <= 3000)

solver.Add(2*I + 1*L + 6*W + 2*P <= 6000)

solver.Add(10*I+ 9*L+ 11*W + 4*P <= 10000)
```

Out[4]:
```
<ortools.linear_solver.pywraplp.Constraint; proxy of <Swig Object of type 'operations_re
search::MPConstraint *' at 0x10d857570> >
```

Now we have set up our system with the amount of each resource used in producing a given beer, how much of the resources we have available, and what the maximum amount of each beer we want to produce is.

The last step is to define for the model what exactly we want to maximize. Here, it's the revenue from selling these beers, and so we will define below how to calculate this:

In [5]:
```
solver.Maximize(4.75*I + 5.25*L + 4.00*W + 2.75*P)
```

We are ready to run the model!

In [6]:
```
status = solver.Solve()
```

In [7]:
```
if status == pywraplp.Solver.OPTIMAL:
    print('Solution:')
    print('Objective value =', solver.Objective().Value())
    print(f'{I.solution_value()}x IPA')
    print(f'{L.solution_value()}x Lager')
    print(f'{W.solution_value()}x Wheat Beer')
    print(f'{P.solution_value()}x Pilsner')
```

```
Solution:
Objective value = 4008.75
174.0x IPA
0.0x Lager
700.0x Wheat Beer
139.0x Pilsner
```

Fantastic! The optimizer quickly solved for the optimal amount of each beer we should produce given these criteria. As you can see, though, the model has identified that it is much better for us to produce the IPA and the Wheat beer than the other two.

Bonus

1. Let's say that even in a year where selling the pilsner or the lager is less profitable, we still want to produce a certain amount so that we don't lose brand awareness. Where in the code that's been written can we define a minimum quantity? Go ahead and add in whatever minimum you would like to see what happens.

2. We can use a model like this to figure out if the price we have set is too low for a given beer. If we raise or lower the price of one of the beers, how does it affect the final result?

3. We can also use the model to figure out which of our ingredients is the most limited, i.e. keeping us from immediately producing more. Can you figure out in this example which of the four ingredients is limiting our production?