DS Roundtable: An Introduction to Linear Programming

7/21/2020

Andrew Wheeler, PhD

DSML Team

andrew.wheeler@hms.com

Overview of the Problem

- Models give a predicted probability or a predicted value
- Predictions do not intrinsically translate to *what* you should do with the information
- Linear programming is how to make decisions based on those model outputs given costs/benefits and constraints

Example 1

- Predicted probability *of rain* is 5%
  - would you bring your umbrella?

- Predicted probability *of used car not working* is 5%
  - Would you buy that car?

- What is the difference? **Costs/Benefits** of false negative

Example 2

- Utility of getting fast food for dinner
  - Benefits – fast, cheap
  - Costs – unhealthy, not as good as home cooked meals

- Costs/Benefits don't change, so why do you get fast food sometimes and not others?
  - Kids have event after school, more likely to get fast food

- What is the difference? **Constraints**

Cost/Benefits & Constraints

- Better Example: When to audit a claim to determine overpayment, based on predictive model probability

- Costs: time it takes to review claim
- Potential Benefits: Probability of overpayment * Magnitude of Overpayment
- Constraints: We want the finding rate to be above a particular level and to not overburden particular providers

Simple Example (no constraints)

Select 2 cases to audit:

	Probability of Overpayment	Estimate of Overpayment Amount	Expected Return
Claim 1	0.05	$10,000	?
Claim 2	0.10	$3,000	?
Claim 3	0.50	$500	?

Simple Example (no constraints)

Select 2 cases to audit:

	Probability of Overpayment	Estimate of Overpayment Amount	Expected Return
Claim 1	0.05	$10,000	$500
Claim 2	0.10	$3,000	$300
Claim 3	0.50	$500	$250

You would select Claims 1 and 2

- total expected return = (500 + 300)
- Estimated overall finding rate is (0.05 + 0.10)/2 = 0.075

Simple Example (constrain finding rate above 20%)

Select 2 cases to audit, but need to keep finding rate above 20%

	Probability of Overpayment	Estimate of Overpayment Amount	Expected Return
Claim 1	0.05	$10,000	$500
Claim 2	0.10	$3,000	$300
Claim 3	0.50	$500	$250

You would select Claims 1 and 3

- total expected return = (500 + 250)
- Estimated overall finding rate is (0.05 + 0.50)/2 = 0.275

Using python *pulp* library to specify models

Making Example Data

```
'''
Linear programming examples
Andy Wheeler, andrew.wheeler@hms.com
'''

#This is the library I like to use
#Many exist though
import pulp

#These are only necessary for simulating data
#You can pass in lists to pulp
import numpy as np
import pandas as pd

###############################################
#Simple Example

#Creating data
prob = [0.05, 0.10, 0.50]
over = [10000, 3000, 500]
exp_ret = [p*o for p,o in zip(prob,over)]
case_index = list(range(len(prob)))
tot_audit = 2 #total number of claims to select
hit_rate = 0.2 #finding rate constraint
```

Setting Up Model & Solving

```

#This is the model
P = pulp.LpProblem("Choosing Cases to Select", pulp.LpMaximize)

#These are the binary decision variables
D = pulp.LpVariable.dicts("Decision Variable", [i for i in case_index],
                          lowBound=0, upBound=1, cat=pulp.LpInteger)

#Objective Function
P += pulp.lpSum( D[i]*exp_ret[i] for i in case_index)

#Constraint on total number of claims selected
P += pulp.lpSum( D[i] for i in case_index ) == tot_audit

#Constraint on the overall hit rate
P += pulp.lpSum( D[i]*prob[i] for i in case_index ) >= hit_rate*tot_audit

#Solve the problem
P.solve()

```

Simulating Data to Look Like Claims

Making Example Data

```
##############################################
#More complicated example data
#Has example providers as well

np.random.seed(10)
n = 20000 #total number of cases
underpay_est = np.random.lognormal(mean=7.6,sigma=0.5,size=n)
#about 25% overall finding rate
prob_over = np.random.beta(2.5,10,size=n)
exp_return = prob_over*underpay_est

#providers have a differential probability of being selected
prov = list('ABCDE')
prov_n = len(prov)
prov_index = list(range(1,prov_n+1))
prov_tot = sum(prov_index)
prov_prob = [i/prov_tot for i in prov_index]
prov_claims = np.random.choice(a=prov, size=n, replace=True, p=prov_prob)

sim_dat = pd.DataFrame(zip(underpay_est, prob_over,
                           exp_return, prov_claims),
                       columns=['underpay_est', 'prob_over',
                                'exp_return', 'prov_claims'])

##############################################
```

Function with Provider Constraints

```
def selection_model(er,prob,provider,data,cases_const,finding_const,provider_const):
    #Preparing simpler lists
    er_list = list(data[er])
    min_const_list = list( data[prob] - finding_const )
    prov_list = list(data[provider])
    index_list = list(range(len(er_list)))
    #getting the locations for each provider in the data
    all_prov = set(prov_list)
    prov_loc = {}
    for p in all_prov:
        prov_loc[p] = list( np.where(data[provider] == p)[0])
    #Now setting up the model
    Sel_Mod = pulp.LpProblem("Selection Model", pulp.LpMaximize)
    Dec_Vars = pulp.LpVariable.dicts("Selected Cases",
                                     [i for i in sim_dat.index],
                                     lowBound=0, upBound=1,
                                     cat=pulp.LpInteger)
    #Objective Function
    Sel_Mod += pulp.lpSum( Dec_Vars[i]*er_list[i] for i in index_list)
    #Constraint on total number of claims selected, has to be equal or fewer
    Sel_Mod += pulp.lpSum( Dec_Vars[i] for i in index_list ) <= cases_const
    #Constraint on finding rate, taking into account total cases selected
    Sel_Mod += pulp.lpSum( Dec_Vars[i]*min_const_list[i] for i in index_list ) >= 0
    #Provider constraints
    for p in all_prov:
        Sel_Mod += pulp.lpSum( Dec_Vars[i] for i in prov_loc[p] ) <= provider_const
    #Solve the Problem
    Sel_Mod.solve()
    #Get the decision variables
    dec_list = [Dec_Vars[i].varValue for i in index_list]
    return dec_list
```

Provider Constraints

No Provider Constraints

```
In [11]: sim_dat['Selected1'] = selection_model(er='exp_return',prob='prob_over',provider='prov_claims',
    ...:                                        data=sim_dat,cases_const=1000,finding_const=0.3,
    ...:                                        provider_const=1000)
    ...:
    ...: print( sim_dat.loc[ sim_dat['Selected1'] == 1, 'prov_claims'].value_counts() )
    ...:
E    350
D    269
C    199
B    114
A     68
Name: prov_claims, dtype: int64

In [12]:
```

Provider Constraints <= 250

```
In [12]: sim_dat['Selected2'] = selection_model(er='exp_return',prob='prob_over',provider='prov_claims',
    ...:                                        data=sim_dat,cases_const=1000,finding_const=0.3,
    ...:                                        provider_const=250)
    ...:
    ...: print( sim_dat.loc[ sim_dat['Selected2'] == 1, 'prov_claims'].value_counts() )
    ...: ################################################
E    250
C    250
D    250
B    160
A     90
Name: prov_claims, dtype: int64
```

The Finding Rate Frontier

```
##############################################
#We can see how changing the finding rate constraint
#Effects our estimates of the expected return

hit_rate = np.linspace(0.10, 0.50, 40)
total_return = []

for h in hit_rate:
    sel_list = selection_model(er='exp_return',prob='prob_over',provider='prov_claims',
                               data=sim_dat,cases_const=1000,finding_const=h,
                               provider_const=250)
    sel_np = np.asarray(sel_list)
    est_ret = (sel_np * sim_dat['exp_return']).sum()
    total_return.append(est_ret)
```

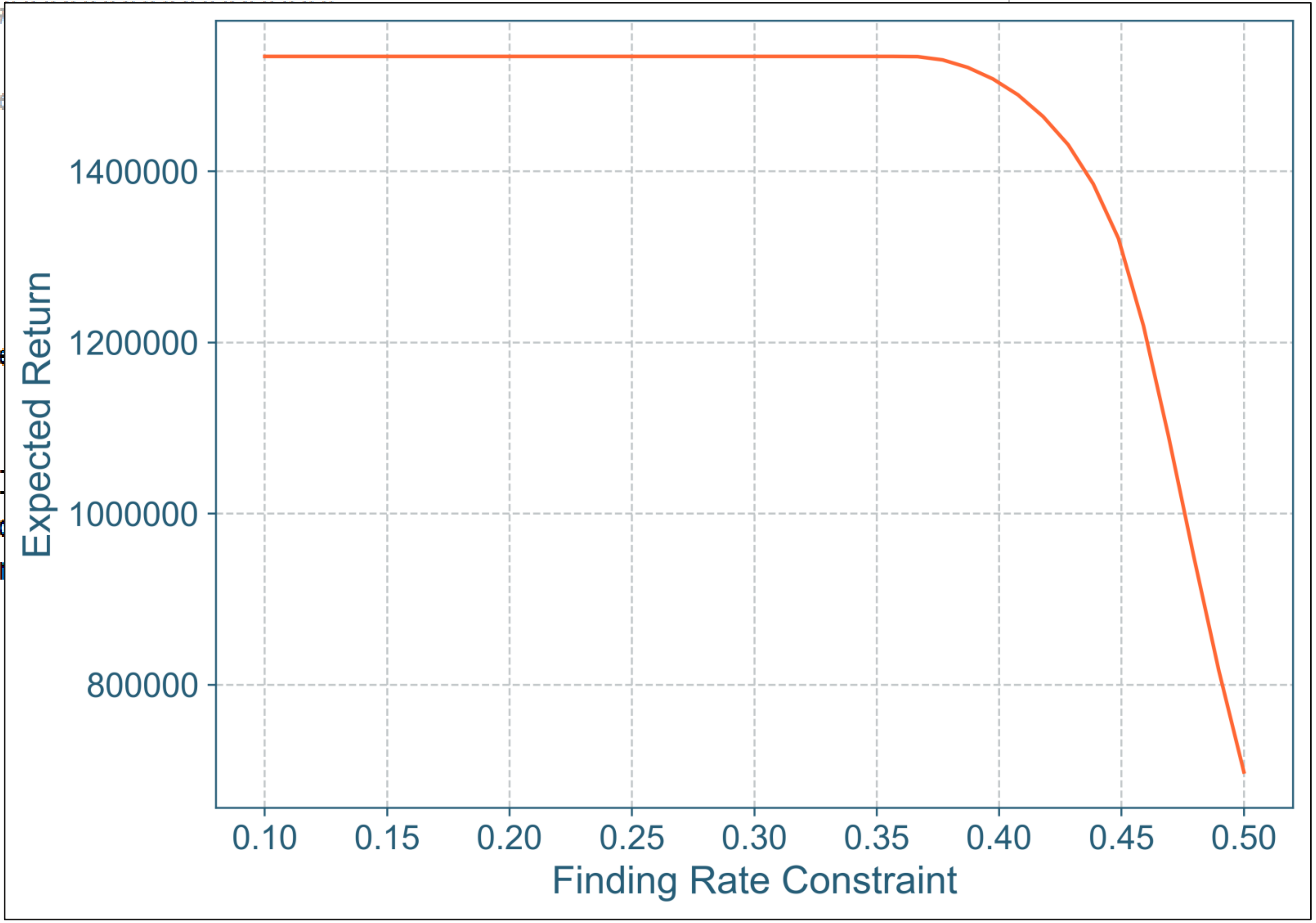
The Finding Rate Frontier

```python
##############################
#We can see how changing the
#Effects our estimates of the

hit_rate = np.linspace(0.10,
total_return = []

for h in hit_rate:
    sel_list = selection_mode


    sel_np = np.asarray(sel_l
    est_ret = (sel_np * sim_c
    total_return.append(est_r
```



Links to Resources

- Python code giving example use, https://github.com/hmsholdings/data-science-utils/tree/master/education/Advanced_DataScience/IntroLinearProgramming
- I have multiple blog posts/papers illustrating different linear programming problems:
    - An intro to linear programming for criminologists
    - Optimal treatment selection with network spillovers
    - Optimizing police patrol areas with workload constraints
    - Racial Equity constraints in predictive policing applications
- The blog Yet Another Math Programming Consultant gives many different examples of linear programming applications

Questions?

DS Roundtable: An Introduction to Linear Programming

7/21/2020

Andrew Wheeler, PhD

DSML Team

andrew.wheeler@hms.com