Homework Week 12 - ISYE 6501 - Spring 2020

Note to grader: All of the student commentary will be in purple text, while the prompt for the questions will be in black text.

Question 15.2

In the videos, we saw the "diet problem". (The diet problem is one of the first large-scale optimization problems to be studied in practice. Back in the 1930's and 40's, the Army wanted to meet the nutritional requirements of its soldiers while minimizing the cost.) In this homework you get to solve a diet problem with real data. The data is given in the file diet.xls.

1. Formulate an optimization model (a linear program) to find the cheapest diet that satisfies the maximum and minimum daily nutrition constraints, and solve it using PuLP. Turn in your code and the solution. (The optimal solution should be a diet of air-popped popcorn, poached eggs, oranges, raw iceberg lettuce, raw celery, and frozen broccoli. UGH!)

2. Please add to your model the following constraints (which might require adding more variables) and solve the new model:

a. If a food is selected, then a minimum of 1/10 serving must be chosen. (Hint: now you will need two variables for each food i: whether it is chosen, and how much is part of the diet. You'll also need to write a constraint to link them.)
b. Many people dislike celery and frozen broccoli. So at most one, but not both, can be selected.
c. To get day-to-day variety in protein, at least 3 kinds of meat/poultry/fish/eggs must be selected. [If something is ambiguous (e.g., should bean-and-bacon soup be considered meat?), just call it whatever you think is appropriate – I want you to learn how to write this type of constraint, but I don't really care whether we agree on how to classify foods!]

If you want to see what a more full-sized problem would look like, try solving your models for the file diet_large.xls, which is a low-cholesterol diet model (rather than minimizing cost, the goal is to minimize cholesterol intake). I don't know anyone who'd want to eat this diet – the optimal solution includes dried chrysanthemum garland, raw beluga whale flipper, freeze-dried parsley, etc. – which shows why it's necessary to add additional constraints beyond the basic ones we saw in the video! [Note: there are many optimal solutions, all with zero cholesterol, so you might get a different one. It probably won't be much more appetizing than mine.]

Solution Explanation

The first step to writing this program was to develop a rough structure of how the program should run. In order to understand how my program works, the steps will be listed out, then each step will get an explanation below. The steps that needed to be included are as follows:

1. Load the XLS data file
2. Split the data into a data frame for the foods (`dfDiet`) and the nutritional limits (`dfLimits`)
3. Initiate the problem object
4. Create the variables
5. Create the objective function
6. Create the contraints
7. Solve the problem
8. Print the results

1. Load the XLS data file

Looking at the `main()` function, the very first line of active code is loading the XLS file into a pandas data frame called `dfDiet` using the `read_excel` command. Once the data was in the program, I wrote a small function `renameColumns` in order to clean up the column names for easier reference during while writing the program.

2. Split the data into `dfDiet` and `dfLimits`

The "foods" field was used to determine which rows were part of the nurtritional limits. The nutrional limits were the bottom 2 rows of the XLS files and their labels were actually in the "serv_size" column, so the fields "foods" and "serv_size" were used to filter down to a data frame with only those 2 rows for `dfLimits` and the field "foods" was used to filter the `dfDiet` data frame to only include the 64 rows that included food.

3. Initiate the problem object

The PuLP problem object was created using the `LpProblem` method. The sense argument was set to "LpMinimize", since we are trying to minimize the cost of all the foods. The problem was named `dietProb`.

4. Create the variables

Three different variables were needed for the program: one for the continuous variables for serving size and two for the binary variables of existence. All variables were created using the `LpVariable` method.

For the amount of each food, a function was written to fill a dictionary called `varAmount` with an integer type variable for each of the 64 food items. Each variable has a lower bound of 0 and an upper bound of infinity. The strategy deployed was to have each unit of this variable be 1/10th of a serving. So if the program's results called for 15 units of Oranges, the interpretation would be 1.5 servings of Oranges.

For the existence of each food, functions were written to fill into 2 complementary dictionaries called `varInclude` and `varExclude`. Each of these structures contains 64 binary variables that can either be 1 or 0. A constraint will be written later to make sure that when one of these variables is 1, the other is always 0.

5. Create the objective function

The function `createObjective` was written to create the objective function, were each food's "price_per" field is multiplied by its `varAmount` and by 0.1 (since each unit is 1/10th of a serving) to get the cost of the diet.

6. Create the contraints

Three different types of constraints were need for this program: a) link the binary variables to the continuous, b) create the min/max nutrional limits, c) the "supplemental" constraints needed for Question 2.

The function `createBinaryCons` was written to link the binary variables. Each of the 64 foods received 3 of these types of constraint. First, `varInclude` plus `varExclude` should always equal 1, since these are complementary variables. Second, `varInclude` multiplied by a very large number (I chose 9999) should always be greater than or equal to `varAmount`, in order to make sure that the continous variable is never greater than 0, while the include binary variable is 0. Thrid, `varAmount` plus `varExclude` should always be greater than or equal to 1, in order to make sure that if the continuous variable is 0, then the exclude binary is TRUE and vice-versa.

The function `createLimitCons` was written to pull in all of the nutrional limits in the frame `dfLimits` and make sure the sum of `varAmount` across all foods stay within the limits. Note that the calculation for this multiplies `varAmount` by 0.1 (since each unit is 1/10th of a serving) to get the nutronal value of each field in the diet.

The function `createSuppCons` was written to add the constraints outlined in Question 2. Note that the function `main()` takes one argument called `includeSuppCons`. When this argument is True, then this constraints are included in the program, and are excluded otherwise. This allowed me to use the same program (and only change this argument) to get the results for Question 1 and Question 2. The first supplementary constraint was to make sure that the existence of broccoli (index == 0) plus celery (index == 2) was less than or equal to one. The second supplementary constraint was to add the existence of all proteins (index list = [8, 9, 27, 28, 29, 30, 31, 32, 41, 42, 43, 44, 49, 50, 51, 56, 57, 59, 61, 63]) and make sure then were greater than or equal to 3.

7. Solve the problem

The code execute to solve the problem was the function `dietProb.solve()`.

8. Print the results

To print the results, all 64 foods were looped through to create a data frame called `resultFrame`. Each row of the frame contains that food's index number, name, servings, continuous variable value (remember that servings = continuous variable value * 0.1), inlude binary variable value, and exclude binary variable value.

Additionaly, the function `dietProb.objective` was called to print the objective value (the total diet cost) of the problem and `LpStatus[dietProb.status]` was called to figure out whether the problem was solved and if the solution was optimal.

For both Question 1 and Question 2, the objective value, along with the rows of the `resultFrame` that were INCLUDED in the diet will be printed.

```python
# import libraries
from pulp import *
import pandas as pd
from collections import defaultdict

# ---------------------------------------------------------------------------
# main function
# ---------------------------------------------------------------------------
def main(includeSuppCons = False):

    # load the diet frame, rename columns
    dfDiet = pd.read_excel('diet.xls')
    dfDiet = renameColumns(dfDiet)

    # get frame of limits
    dfLimits =  dfDiet[dfDiet['foods'].isna()]
    dfLimits =  dfLimits[~dfLimits['serv_size'].isna()]
    dfLimits = dfLimits.reset_index()

    # ignore NA values from foods data
    dfDiet = dfDiet[~dfDiet['foods'].isna()]

    # create problem object
    dietProb = LpProblem(name = "The_Diet_Problem", sense = LpMinimize)

    # create variables
    varAmount = getVarAmount(dfDiet)
    varInclude = getVarInclude(dfDiet)
    varExclude = getVarExclude(dfDiet)

    # create objective function
    createObjective(dfDiet, dietProb, varAmount)

    # create binary related constraints
    createBinaryCons(dfDiet, dietProb, varAmount, varInclude, varExclude)

    # create min and max limit constraints
    createLimitCons(dfDiet, dfLimits, dietProb, varAmount, True)
    createLimitCons(dfDiet, dfLimits, dietProb, varAmount, False)

    # create custom constraints
    if includeSuppCons:
        createSuppCons(dietProb, varInclude)

    # write the problem to LP file (development only)
    # dietProb.writeLP('The_Diet_Problem.lp')

    # solve the problem
    dietProb.solve()

    # print status
    print("Status:", LpStatus[dietProb.status])

    # collect results
    resultFrame = defaultdict(list)
    for i in range(len(varAmount)):
        resultFrame['idx_nbr'].append(i)
        resultFrame['foods'].append(varAmount[i].name)
        resultFrame['servings'].append(varAmount[i].varValue * 0.1)
        resultFrame['var_value'].append(varAmount[i].varValue)
        resultFrame['include'].append(varInclude[i].varValue)
        resultFrame['exclude'].append(varExclude[i].varValue)

    # convert to pandas frame
    resultFrame = pd.DataFrame(resultFrame)

    # output to CSV (development only)
    # resultFrame.to_csv('resultFrame.csv', index=False)

    # print objective
    print("Total_Cost_Value: ", round(value(dietProb.objective), 2))

    # print final frame
    printFrame = resultFrame[resultFrame['include'] == 1]
    print(printFrame.to_string(index=False))

# ----------------------------------------------------------------------------
# rename the columns
# ----------------------------------------------------------------------------
def renameColumns(dfDiet):

    return dfDiet.rename(
    columns = {
        'Foods' : 'foods'
        , 'Price/ Serving' : 'price_per'
        , 'Serving Size' : 'serv_size'
        , 'Calories' : 'calories'
        , 'Cholesterol mg' : 'cho_mg'
        , 'Total_Fat g' : 'fat_mg'
        , 'Sodium mg' : 'sod_mg'
        , 'Carbohydrates g' : 'carb_g'
        , 'Dietary_Fiber g' : 'fiber_g'
        , 'Protein g' : 'protein_g'
        , 'Vit_A IU' : 'vita_iu'
        , 'Vit_C IU' : 'vitc_iu'
        , 'Calcium mg' : 'calcium_mg'
        , 'Iron mg' : 'iron_mg'
        }
    )

# ----------------------------------------------------------------------------
# variable for amounts
# ----------------------------------------------------------------------------
def getVarAmount(dfDiet):

    # structure for variables
    varAmount = {}

    # create each variable
    for i in range(len(dfDiet)):

        # create variables
        varAmount[i] = LpVariable(
            name = dfDiet.at[i, 'foods']
            , lowBound = 0
            , upBound = None
            , cat = LpInteger)

    return varAmount

# ----------------------------------------------------------------------------
# variable for existence
# ----------------------------------------------------------------------------
def getVarInclude(dfDiet):

    # structure for variables
    varInclude = {}

    # create each variable
    for i in range(len(dfDiet)):

        # create variables
        varInclude[i] = LpVariable(
            name = 'Exist_' + dfDiet.at[i, 'foods']
            , lowBound = 0
            , upBound = 1
            , cat = LpInteger)

    return varInclude

# ----------------------------------------------------------------------------
# variable for missing
# ----------------------------------------------------------------------------
def getVarExclude(dfDiet):

    # structure for variables
    varExclude = {}

    # create each variable
    for i in range(len(dfDiet)):

        # create variables
        varExclude[i] = LpVariable(
            name = 'Missing_' + dfDiet.at[i, 'foods']
            , lowBound = 0
            , upBound = 1
            , cat = LpInteger)

    return varExclude

# ----------------------------------------------------------------------------
# create objective function
# ----------------------------------------------------------------------------
def createObjective(dfDiet, dietProb, varAmount):

    # init formula variable
    objFn = 0

    # loop thru each food
    for i in range(len(dfDiet)):

        # add the price to objective function
        objFn += dfDiet.at[i, 'price_per'] * varAmount[i] * 0.1

    # add the objective function to the problem
    dietProb += (objFn, 'Total_Cost_Objective')

    return None
```

```python
# ----------------------------------------------------------------------------
# binary related constraints
# ----------------------------------------------------------------------------
def createBinaryCons(dfDiet, dietProb, varAmount, varInclude, varExclude):

    # contraints for binary links
    for i in range(len(dfDiet)):

        conName = str(i) + '_binary'
        dietProb += ((varInclude[i] + varExclude[i] == 1), conName)

        conName = str(i) + '_include'
        dietProb += ((varAmount[i] <= varInclude[i] * 9999 ), conName)

        conName = str(i) + '_exclude'
        dietProb += ((varAmount[i] + varExclude[i] >= 1), conName)

    return None

# ----------------------------------------------------------------------------
# create contraints
# ----------------------------------------------------------------------------
def createLimitCons(dfDiet, dfLimits, dietProb, varAmount, minCon = True):

    conNames = [

        'calories'
        , 'sod_mg'
        , 'cho_mg'
        , 'fat_mg'
        , 'carb_g'
        , 'fiber_g'
        , 'protein_g'
        , 'vita_iu'
        , 'vitc_iu'
        , 'calcium_mg'
        , 'iron_mg'

    ]

    # create each min contraint
    for conName in conNames:

        # init formula variable
        conLeftSide = 0

        # loop thru each food
        for i in range(len(dfDiet)):
            conLeftSide += (dfDiet.at[i, conName] * varAmount[i] * 0.1)

        # final function, add to problem
        if (minCon == True):
            # for min contraint
            conFn = (conLeftSide >= dfLimits.at[0, conName])
            dietProb += (conFn, 'Min_' + conName)
        else:

            # for max constraint
            conFn = (conLeftSide <= dfLimits.at[1, conName])
            dietProb += (conFn, 'Max_' + conName)

    return None

# -----------------------------------------------------------------------------
# create supplemental constraints
# -----------------------------------------------------------------------------
def createSuppCons(dietProb, varInclude):

    # do not include both broccoli and celery
    dietProb += (varInclude[0] + varInclude[2]) <= 1

    # must have at least 3 proteins
    proteinIdx = [8, 9, 27, 28, 29, 30, 31, 32, 41
        , 42, 43, 44, 49, 50, 51, 56, 57, 59, 61, 63]
    conLeftSide = 0

    for j in proteinIdx: conLeftSide += varInclude[j]
    dietProb += (conLeftSide >= 3)

    return None

# +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Question 1
# run WITHOUT supplement constraints
print('\n\n---- Question 1 Results ----')
main(includeSuppCons = False)

# +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Question 2
# run WITH supplement constraints
print('\n\n---- Question 2 Results ----')
main(includeSuppCons = True)
```

```
##
##
## ---- Question 1 Results ----
## Status: Optimal
## Total_Cost_Value:  4.38
## idx_nbr                foods  servings  var_value  include  exclude
##       0      Frozen_Broccoli       0.2        2.0      1.0      0.0
##       2          Celery,_Raw      49.9      499.0      1.0      0.0
##       4  Lettuce,Iceberg,Raw      68.9      689.0      1.0      0.0
##      15              Oranges       2.5       25.0      1.0      0.0
##      27         Poached_Eggs       0.2        2.0      1.0      0.0
##      48        Peanut_Butter       0.5        5.0      1.0      0.0
##      52   Popcorn,Air_Popped      13.7      137.0      1.0      0.0
##
##
## ---- Question 2 Results ----
## Status: Optimal
## Total_Cost_Value:  4.52
## idx_nbr                foods  servings  var_value  include  exclude
##       2          Celery,_Raw      35.8      358.0      1.0      0.0
##       4  Lettuce,Iceberg,Raw      95.4      954.0      1.0      0.0
##      15              Oranges       3.2       32.0      1.0      0.0
##      27         Poached_Eggs       0.1        1.0      1.0      0.0
##      28       Scrambled_Eggs       0.1        1.0      1.0      0.0
##      32         Kielbasa,Prk       0.1        1.0      1.0      0.0
##      48        Peanut_Butter       1.8       18.0      1.0      0.0
##      52   Popcorn,Air_Popped      13.4      134.0      1.0      0.0
```

Let's take a look at the results of Question 1 and Question 2.

For Question 1, we have a total cost value of $4.38, but we can see we have both broccoli and celery. Also we only have one protein, poached eggs.

For Question 2, we have a higher total cost of $4.52, but we only have celery (no broccoli). Also, we have 3 proteins: poached eggs, scrambled eggs, and kielbasa pork.

We can see that our program worked and managed to find solutions that fit all the constraints.