Modelli Decisionali e Ottimizzazione

Gaetano Settembre n. matricola: 720703

January 11, 2021

1 Programmazione Lineare (PL)

La programmazione lineare (PL) è una branca della Ricerca Operativa che si occupa di studiare algoritmi per la risoluzione di problemi di ottimizzazione lineari. La programmazione lineare è un caso speciale di programmazione matematica (nota anche come ottimizzazione matematica). Un problema è detto lineare se, sia la **funzione obiettivo** che le funzioni che definiscono i **vincoli**, sono lineari. Un problema di programmazione lineare consiste quindi nel ricercare i valori delle variabili decisionali che minimizzano o massimizzano una funzione lineare e che soddisfino contemporaneamente un insieme di vincoli lineari.

In base alla natura o dominio delle variabili decisionali, si parla di:

- Programmazione Lineare (in senso stretto, PL) se tutte le variabili possono assumere valori reali;
- Programmazione Lineare Booleana (PLB) se le variabili assumono i soli valori {0,1};
- Programmazione Lineare Intera (PLI) se tutte le variabili assumono valori interi;
- Programmazione Lineare Intera Mista (PLIM) se alcune variabili possono assumere valori reali e altre valori interi.

Problemi di questo tipo posso essere formulati nel seguente modo:

$$\begin{array}{c} \min c^T x \\ \text{sog . a } Ax \geq b \\ x \geq 0 \end{array} \tag{1}$$

dove $c \in \mathbb{R}^n$ rappresenta il vettore dei costi, $x \in \mathbb{R}^n$ il vettore delle variabili decisionali, $A \in \mathbb{R}^{mxn}$ la matrice dei vincoli e $b \in \mathbb{R}^m$ il vettore dei termini noti. Il prodotto $c^T x = \sum_{i=1}^{n} c_i x_i$ da minimizzare o massimizzare prende il nome di funzione obiettivo o funzione costo.

La programmazione lineare, anche per la sua relativa semplicità, è applicata in diversi settori del mondo reale: dall'industria al commercio, dai trasporti alle telecomunicazioni. Sono esempi di programmazione lineare:

- Problema dello zaino (Knapsack problem);
- Problema di assegnazione;
- Problema del commesso viaggiatore (Traveling salesman problem)
- Problema della dieta.

Di seguito poniamo l'attenzione all'ultimo problema elencato.

2 Problema della dieta

L'obiettivo di questo problema è selezionare una serie di alimenti che soddisfino una serie di esigenze nutrizionali giornaliere al minimo costo. Il problema è formulato come un problema di Programmazione Lineare in cui l'obiettivo è quello di ridurre al minimo i costi e soddisfare i vincoli che rappresentano le esigenze nutrizionali. I vincoli del problema della dieta in genere regolano il numero di calorie, la quantità di grassi, carboidrati, fibre, proteine, ecc assumibili in un determinato giorno.

2.1 Cenni storici

Il problema della dieta è uno dei primi problemi di ottimizzazione studiati negli Stati Uniti negli anni '30 e '40 del secolo scorso per far fronte alle richieste nutritive del personale dell'esercito, minimizzando contemporaneamente le spese derivanti dalla loro alimentazione pur fornendo una dieta sana.

Uno dei primi ricercatori ad affrontare il problema fu George Stigler, che lo studiò e lo formulò nel 1939, identificando una soluzione con un metodo euristico. La sua ipotesi per il costo di una dieta ottimale era di $39.93 all'anno.

In seguito, nell'autunno del 1947, Jack Laderman usando il Metodo del Simplesso (sviluppato nello stesso anno da George Dantzig) risolse il problema che comprendeva 9 equazioni e 77 variabili, impiegando circa 120 giorni/uomo per eseguire i calcoli manualmente, e scoprendo che la soluzione trovata precedentemente da Stigler era sbagliata di soli $0.24.

2.2 Formulazione del problema

Dato un elenco di alimenti, relativi valori nutrizionali e costo per porzione di ogni alimento, l'obiettivo del problema è selezionare il numero di porzioni di ogni alimento da acquistare e consumare in modo da ridurre al minimo il costo della spesa soddisfacendo al contempo i requisiti nutrizionali specificati. Spesso i requisiti nutrizionali sono espressi con minimo e massimo consentito per ogni componente nutrizionale. In un problema di questo tipo possiamo avere diversi tipologie di vincoli:

- Vincoli di qualità;
- Vincoli di non negatività;
- Vincoli logici;

Matematicamente una prima versione del problema della dieta può essere espresso come:

$$\begin{aligned} &\text{Min } \textstyle\sum c_i \cdot x_i \\ &\text{sogg. a } \textstyle\sum \mathrm{q}_{ij} \cdot x_i \geq \text{MinNutriente}_j \\ &\text{sogg. a } \textstyle\sum \mathrm{q}_{ij} \cdot x_i \leq \text{MaxNutriente}_j \\ &\text{sogg. a } x_i \geq 0 \end{aligned} \tag{2}$$

con:
c: vettore dei costi unitari degli alimenti
q_{ij}: quantità di nutriente j presente nell'alimento i

$MinNutriente_j$ e $MaxNutriente_j$: valore assunzione minima e massima giornaliera per il nutriente j

2.3 Risoluzione

L'algoritmo più usato per risolvere problemi di Programmazione Lineare (come quello della dieta) è il **Metodo del Simplesso**, ideato dal matematico George Dantzig, considerato il padre della programmazione lineare.

Senza addentrarci nel dettaglio, in seguito sono riportate alcune informazioni essenziali di questo algoritmo:

- Il metodo del simplesso parte da una qualunque soluzione di base ammissibile (una tale soluzione di partenza viene trovata con la cosiddetta Fase I del metodo del simplesso).
- Ad ogni iterazione, il metodo del simplesso si muove dalla soluzione di base ammissibile corrente ad una nuova soluzione di base ammissibile, ottenuta sostituendo un unico indice della base corrente con un nuovo indice.
- Ad ogni iterazione, la nuova soluzione di base ammissibile ottenuta ha un valore della funzione obiettivo non peggiore di quello della soluzione precedente.
- Dopo un numero finito di iterazioni, il metodo del simplesso restituisce una soluzione di base ottima (oppure fornisce una prova dell'illimitatezza del problema).

Al giorno d'oggi esistono diverse soluzioni software per risolvere un problema di Programmazione Lineare. Si è deciso di usare Python e in particolare una libreria denominata **PuLP**, un progetto open source sviluppato da COIN-OR Foundation, una fondazione educativa e scientifica senza scopo di lucro.
La missione della Fondazione COIN-OR (COmputational INfrastructure for Operations Research) è quella di creare e diffondere conoscenze relative a tutti gli aspetti della ricerca operativa computazionale promuovendo e supportando lo sviluppo guidato di software open scource, implementando un processo accademico di revisione e aumentando la consapevolezza dell'utilizzo di software per risolvere problemi. https://www.coin-or.org/

2.3.1 PuLP

PuLP è una libreria scritta in Python che permette di modellare in maniera facile un PL e sfrutta diversi package come GLPK (GNU Linear Programming Kit) o COIN-OR CBC (Coin-or branch and cut) per la loro risoluzione. Di seguito sono riportati i link al relativo repository GitHub e documentazione: https://github.com/coin-or/Pulp, https://coin-or.github.io/pulp/index.html
Per il corretto funzionamento richiede Python 2.7 o una versione superiore.

Le principali classi che ritroviamo nella libreria sono:

- LpProblem: classe per la creazione di un nuovo PL;
- LpVariable: classe per la modellizzazione delle variabili decisionali;
- LpContraint: classe per la modellizzazione dei vincoli.

2.4 Esempio

Nel file Excel "Dati_dieta.xls" allegato è data una tabella di esempio di valori nutrizionali dove è riportato l'alimento, il costo e le quantità di sostanze nutritive per unità che verrà utilizzata per risolvere il seguente Problema della Dieta:

	Calorie	Colest	Grassi	Sodio	Carbo	Fibre	Proteine	Vit A	Vit C	Calcio	Ferro
Min	1500	30	20	800	130	125	60	1000	400	700	10
Max	2500	24	70	2000	450	250	100	10000	5000	1500	40

[1]:
```
# Importiamo Pandas e PuLP
import pandas as pd
from pulp import *

# Importiamo la tabella nutrizionale dal file excel allegato e creaiamo il␣
 ↪DataFrame
DataFrame = pd.read_excel("Dati_dieta.xls")
DataFrame
```

	Alimenti	Costo	Dim. Porzione	Calorie	Colesterolo (mg)	Grassi (g)	Sodio (mg)	Carboidrati (g)	Fibre (g)	Proteine (g)	Vitamina A (IU)	Vitamina C (IU)	Calcio (mg)	Ferro (mg)
0	Broccoli surgelati	0.16	10 Oz Pkg	73.8	0.0	0.8	68.2	13.6	8.5	8.0	5867.4	160.2	159.0	2.3
1	Carote crude	0.07	1/2 Cup Shredded	23.7	0.0	0.1	19.2	5.6	1.6	0.6	15471.0	5.1	14.9	0.3
2	Sedano crudo	0.04	1 Stalk	6.4	0.0	0.1	34.8	1.5	0.7	0.3	53.6	2.8	16.0	0.2
3	Mais surgelato	0.18	1/2 Cup	72.2	0.0	0.6	2.5	17.1	2.0	2.5	106.6	5.2	3.3	0.3
4	Lattuga Iceberg cruda	0.02	1 Leaf	2.6	0.0	0.0	1.8	0.4	0.3	0.2	66.0	0.8	3.8	0.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
59	Zuppa di vongole	0.75	1 C (8 Fl Oz)	175.7	10.0	5.0	1864.9	21.8	1.5	10.9	20.1	4.8	82.8	2.8
60	Zuppa di pomodoro	0.39	1 C (8 Fl Oz)	170.7	0.0	3.8	1744.4	33.2	1.0	4.1	1393.0	133.0	27.6	3.5
61	Brodo di pesce	0.99	1 C (8 Fl Oz)	163.7	22.3	6.6	992.0	16.6	1.5	9.5	163.7	3.5	186.0	1.5
62	Crema di funghi	0.65	1 C (8 Fl Oz)	203.4	19.8	13.6	1076.3	15.0	0.5	6.1	153.8	2.2	178.6	0.6
63	Zuppa di faglioli	0.67	1 C (8 Fl Oz)	172.0	2.5	5.9	951.3	22.8	8.6	7.9	888.0	1.5	81.0	2.0

64 rows × 14 columns

Come prima cosa creiamo una lista di tutti gli alimenti presenti nel dataset e un dizionario che associa ad ogni alimento il suo prezzo unitario.

[2]:
```
lista_alimenti = list(DataFrame['Alimenti'])
costi = dict(zip(lista_alimenti,DataFrame['Costo']))
#print(lista_alimenti)
#print(costi)
```

Ora creiamo una oggetto di tipo LpProblem istanziando la classe omonima e specificando come parametri il nome e la tipologia del problema, nel nostro caso un problema di minimizzazione.

[3]:
```
problema_dieta = LpProblem("ProblemaDieta", LpMinimize) #LpMaximize se volessimo␣
 ↪creare un problema di massimo
```

Nel seguente snippet sono riportate delle righe di codice finalizzate alla creazione di dizionari in modo tale da avere una corrispondenza tra alimento e la quantità di nutriente contenuta. Creiamo un dizionario per ogni nutriente presente nel dataset in quanto ci saranno utili successivamente per l'istanziazione dei vincoli.

[4]:
```
calorie = dict(zip(lista_alimenti,DataFrame['Calorie']))
colesterolo = dict(zip(lista_alimenti,DataFrame['Colesterolo (mg)']))
grassi = dict(zip(lista_alimenti,DataFrame['Grassi (g)']))
sodio = dict(zip(lista_alimenti,DataFrame['Sodio (mg)']))
carboidrati = dict(zip(lista_alimenti,DataFrame['Carboidrati (g)']))
fibre = dict(zip(lista_alimenti,DataFrame['Fibre (g)']))
proteine = dict(zip(lista_alimenti,DataFrame['Proteine (g)']))
calcio = dict(zip(lista_alimenti,DataFrame['Calcio (mg)']))
ferro = dict(zip(lista_alimenti,DataFrame['Ferro (mg)']))
vitamina_A = dict(zip(lista_alimenti,DataFrame['Vitamina A (IU)']))
vitamina_C = dict(zip(lista_alimenti,DataFrame['Vitamina C (IU)']))
```

Dichiariamo adesso un dizionario di alimenti che corrispondono alle nostre variabili decisionali. Usiamo il metodo presente nella libreria PuLP per specificare il dominio delle nostre variabili e la tipologia. Sfruttiamo questo metodo per inserire il vincolo di non negatività delle nostre variabili, indicando con 0 il limite inferiore. Particolare rilevanza è ricoperta dal parametro "cat" che può assumere il valore di 'Continuous' o 'Integer', indicando la tipologia della soluzione, se reale o intera.

[5]:
```
# La specifica del metodo è la seguente:
# dicts(name,indexs,lowBound=None,upBound=None,cat='Continuous',indexStart=[])

var_alimenti = LpVariable.dicts("Alimento",lista_alimenti, lowBound=0,␣
 ↪cat='Continuous') #o cat='Integer'
```

Adesso abbiamo tutti i dati necessari per aggiungere al nostro problema di programmazione linare la funzione obiettivo.

[6]:
```
problema_dieta += lpSum([costi[i]*var_alimenti[i] for i in var_alimenti])
```

Definiamo i valori minimi e massimi giornalieri di ogni nutriente presente nel dataset.

[7]:
```
Min_Calorie = 1500; Max_Calorie = 2500
Min_Colesterolo = 30; Max_Colesterolo = 240
Min_Grassi = 20; Max_Grassi = 70
Min_Sodio = 800; Max_Sodio = 2000
Min_Carboidrati = 130; Max_Carboidrati = 450
Min_Fibre = 125; Max_Fibre = 250
Min_Proteine = 60; Max_Proteine = 100
Min_Calcio = 700; Max_Calcio = 1500
Min_Ferro = 10; Max_Ferro = 40
Min_VitaminaA = 1000; Max_VitaminaA = 10000
Min_VitaminaC = 400; Max_VitaminaC = 5000
```

Fatto ciò possiamo aggiungere i vincoli di qualità al nostro problema:

[8]:
```
problema_dieta += lpSum([calorie[i] * var_alimenti[i] for i in lista_alimenti])␣
 ↪>= Min_Calorie
problema_dieta += lpSum([calorie[i] * var_alimenti[i] for i in lista_alimenti])␣
 ↪<= Max_Calorie

problema_dieta += lpSum([colesterolo[i] * var_alimenti[i] for i in␣
 ↪lista_alimenti]) >= Min_Colesterolo
problema_dieta += lpSum([colesterolo[i] * var_alimenti[i] for i in␣
 ↪lista_alimenti]) <= Max_Colesterolo

problema_dieta += lpSum([grassi[i] * var_alimenti[i] for i in lista_alimenti])␣
 ↪>= Min_Grassi
problema_dieta += lpSum([grassi[i] * var_alimenti[i] for i in lista_alimenti])␣
 ↪<= Max_Grassi

problema_dieta += lpSum([sodio[i] * var_alimenti[i] for i in lista_alimenti]) >=␣
 ↪Min_Sodio
problema_dieta += lpSum([sodio[i] * var_alimenti[i] for i in lista_alimenti]) <=␣
 ↪Max_Sodio

problema_dieta += lpSum([carboidrati[i] * var_alimenti[i] for i in␣
 ↪lista_alimenti]) >= Min_Carboidrati
problema_dieta += lpSum([carboidrati[i] * var_alimenti[i] for i in␣
 ↪lista_alimenti]) <= Max_Carboidrati

problema_dieta += lpSum([fibre[i] * var_alimenti[i] for i in lista_alimenti]) >=␣
 ↪Min_Fibre
problema_dieta += lpSum([fibre[i] * var_alimenti[i] for i in lista_alimenti]) <=␣
 ↪Max_Fibre

problema_dieta += lpSum([proteine[i] * var_alimenti[i] for i in lista_alimenti])␣
 ↪>= Min_Proteine
problema_dieta += lpSum([proteine[i] * var_alimenti[i] for i in lista_alimenti])␣
 ↪<= Max_Proteine

problema_dieta += lpSum([calcio[i] * var_alimenti[i] for i in lista_alimenti])␣
 ↪>= Min_Calcio
problema_dieta += lpSum([calcio[i] * var_alimenti[i] for i in lista_alimenti])␣
 ↪<= Max_Calcio

problema_dieta += lpSum([ferro[i] * var_alimenti[i] for i in lista_alimenti]) >=␣
 ↪Min_Ferro
problema_dieta += lpSum([ferro[i] * var_alimenti[i] for i in lista_alimenti]) <=␣
 ↪Max_Ferro
```

```
problema_dieta += lpSum([vitamina_A[i] * var_alimenti[i] for i in␣
 ↪lista_alimenti]) >= Min_VitaminaA
problema_dieta += lpSum([vitamina_A[i] * var_alimenti[i] for i in␣
 ↪lista_alimenti]) <= Max_VitaminaA

problema_dieta += lpSum([vitamina_C[i] * var_alimenti[i] for i in␣
 ↪lista_alimenti]) >= Min_VitaminaC
problema_dieta += lpSum([vitamina_C[i] * var_alimenti[i] for i in␣
 ↪lista_alimenti]) <= Max_VitaminaC
```

A questo punto utilizzando il metodo solve() possiamo risolvere il problema di programmazione lineare appena formulato e verificare se:

- il PL ammette soluzione e la soluzione fornita è una soluzione ottima;
- il PL è inammissibile, cioè l'insieme delle soluzioni è vuoto;
- il PL è illimitato

Il metodo che la libreria PuLP usa di default come Solver di un PL generico è l'algoritmo CBC (Coin-or Branch and Cut che sfrutta l'Algoritmo del Simplesso) ma possiamo facilmente scaricare, implementare e usare altri Solver. Nel nostro caso si è proceduto nel download tramite Anaconda di GLPK (GNU Linear Programming Kit), un pacchetto scritto in linguaggio C che include diverse versioni del Simplesso, Branch and Bound, ecc.

[9]:
```
# Qui è riportata la righa di codice che permette, anche da un semplice Jupyter␣
 ↪Notebook
# o dal terminale Anaconda, di installare il pacchetto GLPK con il relativo␣
 ↪Solver compatibile con PuLP

# Decommentare la righa seguente per installare
#!conda install -c conda-forge glpk
```

[10]:
```
# Dalla libreria PuLP invochiamo il metodo 'listSolvers' per visualizzare tutti␣
 ↪i solver compatibili
# e con il parametro onlyAvailable=True solamente quelli installati e␣
 ↪disponibili per l'uso
lista_solver = listSolvers(onlyAvailable=True)
print(lista_solver)
```

```
['GLPK_CMD', 'PULP_CBC_CMD', 'PULP_CHOCO_CMD']
```

[11]:
```
problema_dieta.solve()
# Decommentare la riga di codice successiva se vogliamo risolvere il problema␣
 ↪formulato utilizzando il solver GLPK
# problema_dieta.solve(solver = GLPK_CMD())

# L'attributo status dell'oggetto LpProblem restituisce lo status del problema
```

```
print("Status soluzione:", LpStatus[problema_dieta.status])
```

```
Status soluzione: Optimal
```

A questo punto il vettore delle variabili decisionali x conterrà le quantità di ciascun alimento. Il vettore soluzione x^* è un vettore $\mathbb{R}^n$ che conterrà valori non negativi. In quanto siamo interessati a scoprire quali alimenti comprare/mangiare in questo determinato giorno, andiamo a visualizzare solamente le componenti (alimenti) positive.
Inoltre, il valore della nostra funzione obiettivo, calcolata nella soluzione x^* appena trovata, corrisponderà al costo della dieta.

[12]:
```
print("Una dieta equilibrata ottimale, minimizzando il costo, dovrebbe␣
 ↪comprendere:\n")
for var in problema_dieta.variables():
    if var.varValue > 0:
        print(var.name, "=", round(var.varValue, 3))

print(f'\nIl costo totale di questa dieta è: {round(value(problema_dieta.
 ↪objective),2)}')
```

```
Una dieta equilibrata ottimale, minimizzando il costo, dovrebbe comprendere:

Alimento_Arancia = 2.293
Alimento_Broccoli_surgelati = 0.26
Alimento_Lattuga_Iceberg_cruda = 63.989
Alimento_Popcorn = 13.869
Alimento_Sedano_crudo = 52.644
Alimento_Uova_in_camicia = 0.142

Il costo totale di questa dieta è: 4.34
```

Come possiamo notare il vettore soluzione contiene componenti reali, che in un dominio come questo non è molto pratico. Nonostante ciò bisogna tener conto della colonna 'Dim. Porzione' del dataset iniziale. Nel caso non volessimo avere tale problema basterà semplicemente modificare il dominio delle variabili decisionali al momento della loro dichiarazione (vedi snippet [5]).
In questo caso si parlerà, come detto all'inizio, di un problema di Programmazione Lineare Intera (PLI) che potrà essere comunque risolto con il codice già scritto.

Utilizzando il parametro cat='Integer' lo snippet [5] diventerebbe:

[ ]:
```
var_alimenti = LpVariable.dicts("Alimento",lista_alimenti, lowBound=0,␣
 ↪cat='Integer')
```

e la soluzione, in questo caso del nostro PL Intero, diventerebbe:

```
Alimento_Arancia = 2
Alimento_Kiwi = 1
Alimento_Lattuga_Iceberg_cruda = 91
Alimento_Popcorn = 14
Alimento_Sedano_crudo = 41
```

```
Alimento_Uova_in_camicia = 1

Il costo totale di questa dieta è: 4.89
```

Sebbere la formulazione matematica di questa tipologia di problemi è relativamente semplice, la soluzione proposta potrebbe essere non molto invitante :(