Méthode de l'entropie croisée pour la résolution du problème du sac à dos

Phong Lan Jean-Julien, Université Paris-Nanterre, UFR SEGMI

Juin 2022

Contents

1 Introduction

1.1 Méthode de l'entropie croisée

La méthode d'entropie croisée (CE) est une méthode de Monte Carlo[1] populaire pour l'optimisation en raison de sa simplicité et de son efficacité. Conçue par R. Rubinstein en 1997 [1] pour les simulations d'événements rares où la probabilité qu'un événement cible se produise est relativement faible, la méthode CE repose sur suffisamment d'appels de fonctions objectifs pour estimer avec précision les paramètres optimaux de la distribution sous-jacente. Les travaux ultérieurs de R. Rubinstein (1999 ; 2001) ont montré que de nombreux problèmes d'optimisation peuvent être traduits en un problème d'estimation d'événements rares. En conséquence, la méthode CE peut être utilisée comme algorithme aléatoire pour l'optimisation. L'essentiel de l'idée est que la probabilité de localiser une solution optimale ou quasi optimale à l'aide d'une recherche aléatoire naïve est une probabilité d'événement rare.

1.2 Le problème du sac à dos

Le problème du sac à dos (Kellerer et al. 2004) est défini comme étant le problème de maximisation sous contrainte suivant:

$$\begin{aligned} \underset{Xi}{\text{maximize}} \quad & \sum_{j=0}^{n-1} p[j]Xi[j], \qquad Xi[j] \in \{0,1\}, \quad j = 0, ..., n-1 \\ \text{subject to} \quad & \sum_{j=0}^{n-1} w[j]Xi[j] \leq c \end{aligned} \tag{1}$$

où $p[j]$ est la liste des profits des objets, $w[j]$ est la liste des poids des objets, c la capacité totale du sac et Xi est la i-ème variable (pour i variant de 0 à N-1) qui est une liste de n valeurs de simulation de variables de bernoulli. On désigne par $Xi[j]$ la j-ème valeur de la liste Xi.
Autrement dit ce problème revient à trouver la variable Xi qui maximise la somme des profits des objets sous contrainte que le poids des objets sélectionnés ne doivent pas dépasser la capacité totale du sac.

1. La première étape de l'algorithme est de définir une liste initiale $\hat{v_1} = [1/2, 1/2, ..., 1/2]$ de longueur n avec n le nombre d'objets total. On définit $rho = 0.1$ et $N = 10^3$ et on initialise Ne comme étant la partie entière supérieure de $rho \times N$. On initialise le temps à $t = 1$.

2. On génère ensuite Xi variables i.i.d pour i variant de 0 à N-1, chacune étant une liste de n valeurs de simulation de variables de bernoulli, suivant

[1] Les méthodes de Monte Carlo , ou expériences de Monte Carlo , sont une large classe d' algorithmes de calcul qui reposent sur un échantillonnage aléatoire répété pour obtenir des résultats numériques.

la probabilité $\hat{v_t}$. On calcule ensuite la performance $S(Xi)$ de chacune de ces variables avec la fonction objectif suivante:

$$S(Xi) = -\sum_{j=0}^{n-1} p[j] \sum_{i=0}^{n-1} \mathbf{1}_{\{\sum_{j=0}^{n-1} w[j]Xi[j]>c\}} + \sum_{j=0}^{n-1} p[j]Xi[j]$$

On ordonne ensuite du plus petit au plus grand les performances comme ceci: $S(X_1) \leq ... \leq S(X_N)$. On désigne ensuite par $\hat{\gamma_t}$ la performance $S(X_{N-Ne+1})$. C'est le $(1 - rho)$-quantile des performances. Autrement dit, c'est le score au temps t.

3. On utilise ensuite le même échantillon $X_1, ...X_N$ pour résoudre le programme stochastique suivant:

$$\max_v \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{1}_{\{S(Xi)\geq\hat{\gamma_t}\}} \ln f(Xi; v)$$

avec f la densité d'une loi de bernoulli multivariée. On note la liste solution du programme par $\hat{v_t}$.
Cela revient à calculer la j-ème observation de la liste $\hat{v_t}$ comme ceci:

$$\hat{v_t}[j] = \frac{\sum_{i=0}^{N-1} \mathbf{1}_{\{\hat{S}(Xi)\geq\hat{\gamma_t}\}} Xi[j]}{\sum_{i=0}^{N-1} \mathbf{1}_{\{\hat{S}(Xi)\geq\hat{\gamma_t}\}}}$$

4. On réitère depuis la deuxième étape jusqu'à t=50 et affecte à t $t = t + 1$ à chaque itération.

2 Algorithme CE pour l'optimisation du problème du sac à dos

2.1 Partie 1: Définition des paramètres dans la fonction principale

1. Définir n = le nombre d'objets total.
2. Définir la liste $\hat{v_1}$ = [1/2, 1/2, ... , 1/2] de longueur n.
3. Définir le paramètre $N = 10^3$. Plus le paramètre N est grand, et plus l'algorithme va converger vers la solution optimale.
4. Définir le paramètre $rho = 0.1$.
5. Définir le paramètre $Ne = \lceil rho \times N \rceil$, càd la partie entière supérieure de $rho \times N$.
6. Définir la liste w comme étant la liste des poids des objets.

7. Définir la liste p comme étant la liste des profits des objets.
8. Définir capacity = la capacité maximum du sac à dos.
9. Définir un tableau vide X à N lignes et n colonnes .
10. Définir une liste vide des scores $\hat{\gamma}_t\_list$.
11. Initialiser le compteur t à $t = 1$.
12. Définir une liste vide des temps t_list .

2.2 Partie 2: Définition des fonctions

1. - Définir la fonction "simulation" qui permet de simuler une variable de bernoulli de probabilité le j-ème élément de la liste $\hat{v}_t$. La fonction "simulation" prend en paramètre la liste $\hat{v}_t$.
   - Déclarer la variable U qui prend une valeur suivant une loi uniforme sur [0; 1].

   On déclare ensuite la variable x et on retourne sa valeur suivant l'algorithme suivant:

Algorithm 1 Calcul de la variable x

Function $simulation(\hat{v}_t)$:
U = nombre suivant une loi uniforme sur $[0; 1]$.
if $U < \hat{v}_t$ **then**
  $x = 1$
else
  $x = 0$
end if
return x

2. - Définir la fonction "indicatrice1" qui prend en paramètres w, Xi, capacity. Xi est la i-eme variable (pour i variant de 0 à N-1) qui est une liste de n valeurs de simulation de variables de bernoulli. On désigne par $Xi[j]$ la j-ème valeur de la liste Xi.
   - La fonction indicatrice1 renvoie

$$\sum_{j=0}^{n-1} \mathbf{1}_{\{w[j]Xi[j]>c\}}$$

     où wi[j] est le poids du j-ème objet pour j variant de 1 à n et ci la capacité totale du sac.
   - Définir $a = Somme(w[j] \times Xi[j])$ pour j variant de 0 à n-1.

   On déclare la fonction indicatrice selon l'algorithme suivant:

Algorithm 2 Calcul de la variable *indicatrice*

Function $indicatrice1(w, Xi, capacity)$:
$a = Somme(w[j] \times Xi[j])$ pour j variant de 0 à n-1
if $a > capacity$ **then**
  $indicatrice = 1$
else
  $indicatrice = 0$
end if
return *indicatrice*

3.
   - Définir la fonction "$fonction\_objectif$" qui prend en paramètres n, p, w, Xi. S est la fonction objectif.
   - Définir SI une liste vide.

Algorithm 3 Calcul de la fonction objectif $S(Xi)$

Function $fonction\_objectif(n, p, w, Xi)$:
$SI \leftarrow \varnothing$
for (j=0 ; j ≤ n-1; j++) **do**
  Ajouter à SI : indicatrice1(w, Xi, capacity)
end for
$S = (-Somme(p[j] \times Somme(SI[j]) + Somme(p[j] \times Xi[j])$ pour j variant de 0 à n-1
return S

4.
   - Définir la fonction "$performance\_variable$" qui calcule la performance de la i-ème liste Xi pour i variant de 1 à N. Cette fonction prend en paramètres N, n, X.
   - Définir une liste vide S_list qui est la liste des résultats de la fonction objectif pour chacune des listes Xi.

Algorithm 4 Calcul des valeurs de S_list

Function $performance\_variable(N, n, X)$:
$S\_list \leftarrow \varnothing$
for (i=0 ; i ≤ N-1; i++) **do**
  $S = fonction\_objectif(n, p, w, X[i])$
  Ajouter à Slist : S
end for
$S = (-Somme(p[j] \times Somme(Si[j]) + Somme(p[j] \times Xi[j])$ pour j variant de 0 à n-1
return S_list

5. Définir la fonction "tri" qui classe les performances de chaque variable Xi du plus petit au plus grand. Elle prend en paramètre S_list.

Algorithm 5 Classement des performances S_list_tri

Function tri(S_list):
S_list_tri = Liste du tri des valeurs de S_list du plus petit au plus grand.
$S\_list\_tri\_index$ = Liste des indices du tri des valeurs de Slist du plus petit au plus grand.
return $S\_list\_tri, S\_list\_tri\_index$

6.
   - Définir la fonction "indicatrice2" qui renvoie

$$\sum_{i=0}^{N-1} \mathbf{1}_{\{\hat{S}(Xi) \geq \hat{\gamma}_t\}}$$

     . Elle prend en paramètre N, S_list_tri, $\hat{\gamma}_t$.
   - Définir une liste vide $s\_indicatrices$.

Algorithm 6 Calcul de la somme des indicatrices $S\_indicatrices$

Function indicatrice2($N, S\_list\_tri, \hat{\gamma}_t$):
$s\_indicatrices \leftarrow \varnothing$
for (i=0 ; i ≤ N-1; i++) **do**
  if

$$S\_list\_tri[k] \geq \hat{\gamma}_t$$

  then
    Ajouter à $s\_indicatrices$: 1
  else
    Ajouter à $s\_indicatrices$: 0
  end if
end for
return $s\_indicatrices$

7. - Définir la fonction "*Poids_total_sac*" qui renvoie le poids du sac selon la liste $\hat{v_t}$ (si la j-ème valeur de la liste $\hat{v_t}$ est égale à 1, alors le poids du j-ème objet est ajouté dans le sac). Elle prend en paramètres wa, n avec wa le tableau de n lignes et 2 colonnes, la première colonne contenant le poids de chaque objet et la deuxième colonne étant la liste $\hat{v_t}$.
   - Définir $TW = 0$.

Algorithm 7 Calcul de la somme des indicatrices *S_indicatrices*

Function $Poids\_total\_sac(wa, n)$:
$TW \leftarrow 0$
for (j=0 ; j ≤ n-1; j++) **do**
  if
$$wa[1][i] == 1$$
  then
    $TW \leftarrow TW + wa[0][i]$
  else
    $TW = TW$
  end if
end for
return TW

8. - Définir la fonction "*calcul_vt*" qui renvoie la liste v au temps t+1 ($v_{t+1}^{\hat{}}$). Elle prend en paramètres $v_{t+1}^{\hat{}}, op2, s\_indicatrices, X$.
   - Définir une liste vide $op1\_list$.
   - Définir dans la première boucle une liste vide $op1$.

Algorithm 8 Calcul du vecteur $v_{t+1}^{\hat{}}$

Function $calcul\_vt(v_{t+1}^{\hat{}}, op2, s\_indicatrices, X)$:
$op1\_list \leftarrow \varnothing$
for (j=0 ; j ≤ n-1; j++) **do**
  $op1 \leftarrow \varnothing$
  for (i=0 ; i ≤ N-1; i++) **do**
    Ajouter à op1 : $s\_indicatrices[i] * X[i][j]$
  end for
  Ajouter à op1 : Somme(op1[i]) pour i variant de 0 à N-1
  Ajouter à $v_{t+1}^{\hat{}}$: $op1\_list[j]/op2$
end for
return $v_{t+1}^{\hat{}}$

2.3 Partie 3: Définition de la fonction principale

1. Définir les paramètres de la partie 2.1 .
2. Écrire l'algorithme suivant qui retourne le meilleur score, le vecteur du meilleur score, le poids total du sac pour ce vecteur et l'indice du temps du meilleur score:

Algorithm 9 Fonction principale

Function $main(capacity, N, n, rho, w, p)$:
for (t=1 ; t $\leq$ 50; t++) **do**
  for (i=0 ; i $\leq$ N-1; i++) **do**
    $Xi \leftarrow \varnothing$
    for (j=0 ; j $\leq$ n-1; j++) **do**
      Ajouter à Xi : simulation($\hat{v_t}[j]$)
    end for
    Ajouter à X[i] la liste Xi.
  end for
  $S\_list = performance\_variable(N, n, X)$
  $S\_list\_tri, S\_list\_tri\_index = tri(S\_list)$
  $index\_\hat{\gamma_t} = N - Ne + 1$
  $\hat{\gamma_t} = S\_list\_tri[index\_\hat{\gamma_t}]$
  Ajouter à $\hat{\gamma_t}\_list[i]$: $\hat{\gamma_t}$
  $s\_indicatrices = indicatrice2(N, S\_list, \hat{\gamma_t})$
  $op2 = sum(s\_indicatrices[i])$ pour i variant de 0 à N-1
  $\hat{v_{t+1}} \leftarrow \varnothing$
  $\hat{v_{t+1}} = calcul\_vt(\hat{v_{t+1}}, op2, s\_indicatrices, X)$
  $t = t + 1$
  Ajouter à t_list: t
  $\hat{v_t} = \hat{v_{t+1}}$
  $wa \leftarrow$ Ajouter la colonne $\hat{v_t}$ au vecteur w.
  TW =Poids_total_sac(wa, n)
end for
$t\_max\_\hat{\gamma_t}\_index$ = indice du score maximum dans $\hat{\gamma_t}\_list$
$t\_max\_\hat{\gamma_t} = t\_list[t\_max\_\hat{\gamma_t}\_index]$
Afficher l'évolution des scores contenus dans la liste $\hat{\gamma_t}\_list$ selon les temps contenus dans la liste t_list
return $\hat{\gamma_t}, v_t, TW, t\_max\_\hat{\gamma_t}\_index$

References

[1] Zdravko I. Botev, Department of Computer Science and Operations Research, Université de Montréal - P. Kroese, School of Mathematics and Physics, The University of Queens-land - Y. Rubinstein, Faculty of Industrial Engineering and Management - Pierre L'Ecuyer, Department of Computer

Science and Operations Research,Université de Montréal. *The Cross-Entropy Method for Optimization*. La dernière modification a été faite en Décembre 2013.

[2] Robert J. Moss – Stanford University, Departement of Computer Science Stanford, *Cross-Entropy Method Variants for Optimization* (English), La dernière modification a été faite en août 2020.

[3] Wikipedia (English). *Cross-entropy method*.