Monte Carlo, MCMC
Applications en Machine Learning

Satya V.Sarma

8 Décembre 2016

Table des matières

Intégration

On considère des problèmes où il faut calculer :

$$\mathbb{E}_f[h(X)] = \int_\Omega h(x)f(x)dx \tag{1}$$

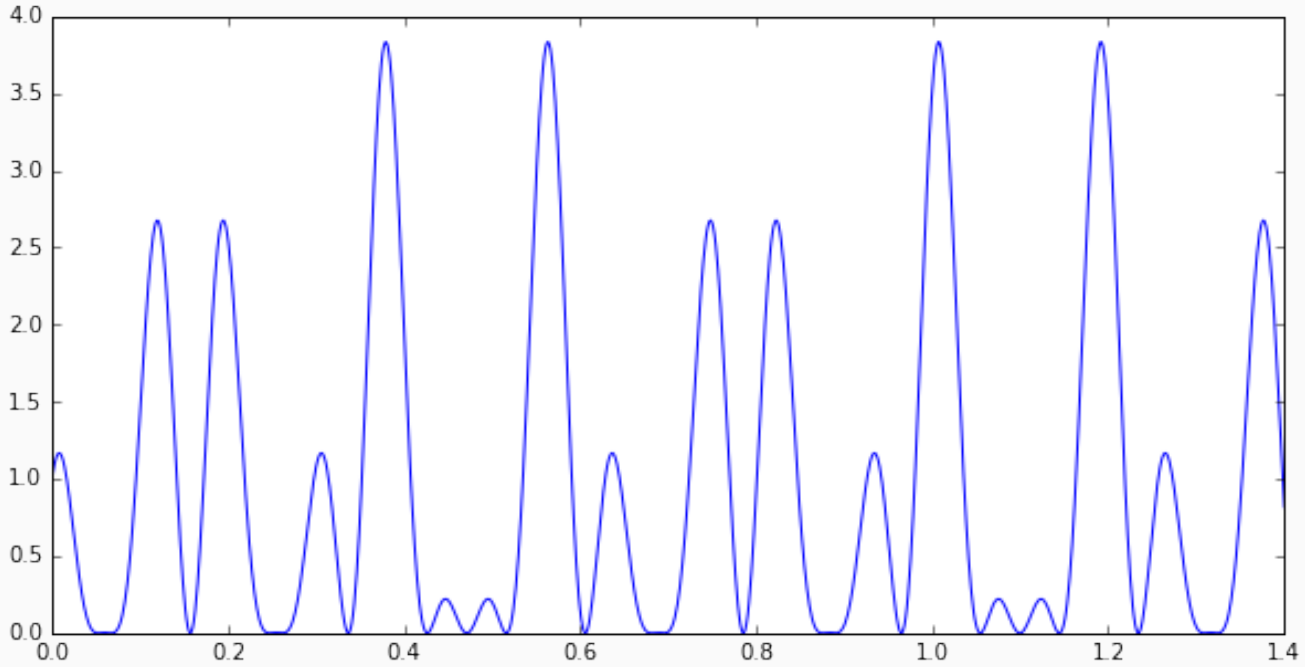
où X est une variable aléatoire sur Ω avec fonction de répartition F et densité f.

Quantité potentiellement impossible à calculer par méthodes déterministes !

Optimisation

Ou bien encores des problèmes où on doit optimiser des fonctions avec plusieurs extrema locaux, comme par exemple :

Courbe de la fonction $f(x) = [cos(50x) + sin(20x)]^2$



La descente de gradient usuelle n'est plus efficace !

Définition

- **Méthodes de Monte-Carlo** : famille de méthodes algorithmiques visant à calculer une valeur numérique approchée en utilisant des procédés aléatoires
- Utilisées pour la première fois dans le cadre du projet Manhattan, puis formalisées dans les années 50
- Référence aux jeux de hasard pratiqués à Monte-Carlo

Table des matières

Idée générale

On utilise la simulation de variables aléatoires iid pour l'estimation et l'optimisation.

Dans le cas de l'intégration, pour estimer $\mathbb{E}_f[h(X)]$ on utilise la quantité suivante :

$$\bar{h}_N = \frac{1}{N} \sum_{i=1}^{n} h(x_i) \tag{2}$$

où les x_i sont simulées selon la loi de fonction de répartition F associée à la densité f.

Par application directe de la Loi des Grands Nombres, cet estimateur converge vers la vraie valeur.

Table of contents

Comment simuler des variables aléatoires ?

La méthode la plus simple se base sur le résultat suivant :

Théorème

Soit une variable aléatoire $X \sim \pi$ avec pour fonction de répartition F. Alors la variable aléatoire $F(X) \sim \mathcal{U}([0,1])$.

En effet $\mathbb{P}(F(X) < t) = \mathbb{P}(X < F^{-1}(t)) = F(F^{-1}(t)) = t$

Comment simuler des variables aléatoires ?

- Simulation d'une variable selon la loi π : on tire des nombres u_i au hasard dans $[0, 1]$ et $F^{-1}(u_i)$ fournit une réalisation de variable aléatoire suivant la loi π.
- Méthode peu pratique lorsque F n'est pas continue
- D'autres méthodes existent dans ces cas : simulation par acceptation-rejet, méthode de Box-Muller
- Python : des générateurs sont implémentés dans *scipy* et *numpy* pour beaucoup de lois usuelles. De même sur R

Application : intégration

Exemple 1

Soit $Z \sim \mathcal{N}(0,1)$. Estimer $\mathbb{P}(Z > 2.5)$.

Puisque $\mathbb{P}(Z > 2.5) = \mathbb{E}(\mathbb{1}_{Z>2.5})$, on peut utiliser l'estimateur de Monte-Carlo suivant :

$$\bar{h}_N = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{Z_i>2.5} \tag{3}$$

où $(Z_i)_i$ simulés selon une loi $\mathcal{N}(0,1)$. Voir code Python.

Table of contents

Variante pour l'intégration : importance sampling

Si f n'est pas associée à une loi usuelle, il peut être compliqué (voire totalement impossible) de simuler des variables selon cette loi. Dans ce cas, on utilise *l'importance sampling* :
$\mathbb{E}_f[h(X)] = \int_\Omega h(x)f(x)dx = \int_\Omega h(x)\frac{f(x)}{g(x)}g(x)dx$ est approximée par la quantité

$$\bar{h}_N = \frac{1}{N}\sum_{i=1}^{n} h(x_i)\frac{f(x_i)}{g(x_i)} \tag{4}$$

où les x_i sont simulés selon la loi associé à g, que l'on sait simuler, et plus à f

Optimisation stochastique

Méthode 1 : Exploration aléatoire (*random search*)

- Consiste à tirer aléatoirement des nombres sur le domaine de définition de la fonction que l'on veut optimiser, puis à calculer leurs images et d'en prendre le max ou le min.
- Fonctionne bien sur les ensembles de faible dimension et quand l'évaluation de la fonction est peu coûteuse.

Optimisation stochastique

Méthode 2 : Descente de gradient stochastique (SGD)

On modifie la séquence de la descente de gradient classique :

$$x_{j+1} = x_j + \frac{\alpha_j}{2\beta_j}\Delta f(x_j; \beta_j\zeta_j)\zeta_j \tag{5}$$

avec $(\alpha_j)_j$ et $(\beta_j)_j$ des suites déterministes décroissantes, $(\zeta_j)_j$ une suite de variables aléatoires telles que $\|\zeta\| = 1$

Idée : la direction dans laquelle on se déplace de j à $j+1$ est aléatoire !

$\Delta f(x, y) = f(x+y) - f(x-y)$ approxime $2\|y\|\nabla f(x)$

Application : optimisation

Exemple 2

Déterminer le maximum global de la fonction :

$$f(x) = [cos(50x) + sin(20x)]^2$$

L'exploration aléatoire fonctionne bien ici. On peut aussi utiliser la descente de gradient stochastique. Voir code Python.

Table of contents

Algorithme Expectation-Maximization

Algorithme EM : méthode permettant d'estimer l'estimateur du maximum de vraisemblance d'un modèle.

Utile lorsque la vraisemblance du modèle est trop compliquée pour obtenir directement l'EMV.

2 étapes à chaque itération de l'algorithme :

- **E step** : on évalue l'espérance de la log-vraisemblance du modèle
- **M step** : on maximise cette quantité en les paramètres à estimer

Algorithme Expectation-Maximization

Le Monte-Carlo peut servir à chacune des deux étapes :

- **E step** : estimation de l'espérance de la log-vraisemblance par moyenne empirique basée sur des simulations aléatoires
- **M step** : optimisation de cette quantité par descente de gradient stochastique ou autre procédé

La variante de l'algorithme EM utilisant les méthodes de Monte-Carlo est appelée algorithme MCEM.

Modèles basés sur le MCEM : la quasi-totalité des algorithmes de clustering probabilistes (Gaussian mixture, LDA & co.), de nombreux modèles de NLP.

Table des matières

Idée générale

Principal désavantage des méthodes de Monte-Carlo : **fléau de la dimension**

Lorsque des problèmes à traiter relèvent d'espaces dont la dimension est trop élevée, on est forcé d'abandonner le cadre des simulations iid.

Alors, pour échantillonner selon une loi π, **on utilise une chaîne de Markov ergodique dont la loi stationnaire est π.**

Table of contents

Metropolis-Hastings

Principe du MCMC : on part d'une valeur arbitraire x_0 puis on génère une chaine $(X_t)_t$ ergodique dont la loi stationnaire est f. Mais comment générer cette chaîne ?

Algorithme de Metropolis-Hastings (version *random walk*) :

Algorithm 1 Metropolis-Hastings

1: **Input** : $x^{(0)}$ and number of iterations T

2: **Repeat** : For t in $\{0, ..., T\}$

Simulate $y_{t+1} \sim g(y - x^{(t)})$

Take

$$x^{(t+1)} = \begin{cases} y_t & \text{with probability } min\left(1; \frac{f(y_t)}{f(x^{(t)})}\right) \\ x^{(t)} & \text{otherwise} \end{cases}$$

3: **Output** : A sequence $(x^{(t)})_t$

avec f la densité cible (que l'on souhaite simuler) et g la densité instrumentale dont on se sert pour la simulation, qui doit vérifier $g(x) = g(-x)$ dans la version marche aléatoire de l'algorithme.

Metropolis-Hastings

Exemple 3

Simuler la densité $\mathcal{N}(0,1)$ par randow walk Metropolis-Hastings en utilisant comme densité instrumentale la loi $\mathcal{U}([-x-1.70;-x+1.70])$.

Table of contents

Gibbs-Sampler

Le Gibbs Sampler est un algorithme de type MCMC, principalement utilisé pour l'inférence dans les modèles bayésiens ou faisant intervenir des lois dans des espaces à grande dimension.

Soit X et Y deux variables aléatoires. Le Gibbs-Sampler à deux étapes génère la chaîne de Markov (X_t, Y_t) de la façon suivante :

Gibbs-Sampler

Soit $X_0 = x_0$ fixé. Pour $t \in \{1, 2, ...\}$ générer :

- $Y_t \sim f_{Y|X}(.|x_{t-1})$
- $X_t \sim f_{X|Y}(.|y_t)$

Gibbs Sampler

Le Gibbs-Sampler est utile lorsque l'on ne sait pas simuler selon la loi du vecteur (X_t, Y_t) mais qu'il est facile de simuler seloin les lois conditionnelles.

Exemple 4

Estimation de la taille d'une population par capture-recapture en 2 étapes

Le Gibbs-Sampling dans sa version généralisée à n étapes est couramment utilisé en vision par ordinateur et dans certains modèles bayésiens.

Référence : Robert C. P., Monte carlo methods. *John Wiley and Sons, Ltd*, 2004.