reservoir_sampling

July 1, 2023

1 Reservoir Sampling distribué - énoncé

Reservoir Sampling sur Map/Reduce. Cet algorithme est un algorithme d'échantillonnage en streaming.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

1.1 sampling sur Hadoop / PIG

Voir SAMPLE pour *PIG*, takeSample sur *Spark*, ou [sample](https://spark.apache.org/docs/1.4.0/api/java/org/apache/spark
En pratique, il difficile d'écrire un algorithme plus efficace à partir d'un langage haut niveau si ce langage haut niveau implémente lui-même la fonctionnalité. C'est une façon de reconnaître que c'est un besoin fréquent et qu'il est recommandé d'utiliser la fonction native.

1.2 Reservoir Sampling

Le reservoir sampling est une façon de tirer un échantillon aléatoire de k nombres parmi N nombres qui ne nécessite pas de conserver ces N nombres en mémoire.

[2]:
```
ensemble = [ "%d%s" % (i, chr(i%26 + 97)) for i in range(0,10000)]
ensemble[:5]
```

[2]:
```
['0a', '1b', '2c', '3d', '4e']
```

[3]:
```
import random

def reservoir_sampling(ensemble, k):
    N = len(ensemble)
    echantillon = []
    for i, e in enumerate(ensemble):
        if len(echantillon) < k:
            echantillon.append(e)
        else:
            j = random.randint(0, i-1)
            if j < k:
                echantillon[j] = e
    return echantillon

reservoir_sampling(ensemble, 10)
```

[3]:
```
['7629l',
 '6747n',
 '1035v',
 '7538y',
 '1113v',
 '7565z',
 '2258w',
 '5101f',
 '1235n',
 '9965h']
```

Le *reservoir sampling* reproduit un tirage sans remise. L'algorithme habituel - on tire k éléments parmi N, la probabilité qu'un élément fasse partie de l'échantillon est de :

$$p = \frac{C_{N-1}^{k-1}}{C_N^k} = \frac{\frac{N-1!}{k-1!N-k!}}{\frac{N!}{k!N-k!}} = \frac{N-1!k!}{N!k-1!} = \frac{k}{N}$$

Dans le cas du *reservoir sampling*, on doit s'assurer que chaque élément à la même probabilité de faire partie de l'échantillon. On procède par récurrence. Le résultat est vrai pour $k = N$. On suppose que est vrai à l'ordre $N-1$. La probabilité qu'un élément de l'ensemble fasse partie de l'échantillon est $\frac{k}{N-1}$. A l'ordre N, il y a une probabilité $\frac{k}{N}$ de remplacer un des éléments de l'échantillon. L'élément N a donc une probabilité $\frac{k}{N}$ de faire partie de l'échantillon. Pour les éléments faisant déjà partie de l'échantillon, leur probabilité de rester est $\frac{N-k}{N} + \frac{k}{N}\frac{k-1}{k} = \frac{N-1}{N}$. La probabilité qu'un élément présent dans l'échantillon soit dans l'échantillon à l'itération N est de $\frac{N-1}{N}\frac{k}{N-1} = \frac{k}{N}$. L'échantillon produit par le *reservoir sampling* a les mêmes propriétés qu'un échantillon produit avec un tirage sans remise.
Cet algorithme est intéressent lorsqu'on veut échantillonner sur une grande base de données car il nécessite de ne garder en mémoire que l'échantillon.

1.3 Reservoir Sampling Distribué

Distribuer l'algorithme paraît simple : une partie des données ira sur une machine qui en tirera un échantillon, l'autre machine tirera un échantillon aléatoire sur l'autre partie des données. Par extension, sur m machines, on obtient m échantillons de tailles $k_1, ..., k_m$ tirés sur des ensembles de tailles $N_1, ..., N_m$. Il faut s'assurer que la version distribuée produit un échantillon avec les mêmes propriétés.

1.3.1 exercice 1 : combinaison

Comment recombiner ces échantillons aléatoires ? Comment choisir les k_i sachant qu'on doit tirer un échantillon de taille $k \leqslant \sum_{i=1}^m k_i$ parmi $N = \sum_{i=1}^m N_i$?

[4]:

1.3.2 exercice 2 : script PIG, Spark

Ecrire le script PIG correspondant à l'algorithme.

[5]:

1.4 Petit problème théorique

Un Générateur congruentiel linéaire imite seulement le hasard, ce n'est pas vraiment le hasard. La suite dépend d'une graine ou *seed*. Si deux machines partent de la même graine, elle produiront la même suite. Qu'en est-il du hasard ? La solution de l'exercice précédent est-elle correcte d'un point de statistique ?

1.5 Reservoir Sampling pondéré

On serait tenter d'écrire… On suppose qu'on dispose un ensemble de points (X_i) pondéré par $\omega_i \in \mathbb{R}$. On veut tirer un échantillon de k éléments. On procède de la même manière que pour le *reservoir sampling* non pondéré. A l'étape $n-1$, on dispose d'un échantillon $E_{n-1} = (X_{i_1}, ..., X_{i_k})$. On tire un nombre u selon une loi uniforme $[0,1]$. On ajoute l'élément X_n si :

$$u \leqslant \frac{k\omega_n}{\sum_{l=1}^{k} w_{i_l}}$$

On veut montrer que pour tout $j \leqslant n$:

$$\mathbb{P}(X_j \in E_n) = \frac{k\omega_j}{\sum_{l=1}^{n} w_l}$$

On procède par récurrence en supposant cela vrai à l'ordre $n-1$, donc $\mathbb{P}(X_j \in E_{n-1}) = \frac{k\omega_j}{\sum_{l=1}^{k} w_j}$. On vérifie que cela est vrai pour $n = k$. Mais a priori, il n'y a aucune raison que cela soit vrai.
Il faut faire autrement.
On utilise l'algorithme A-Res. Soit $E_{n-1} = (X_{i_1}, ..., X_{i_k})$ pondéré par $(r_1, ..., r_k)$ qui sont différents des $(\omega_{i_1}, ..., \omega_{i_k})$. On considère l'élément X_n pondéré par ω_n. On l'ajoute à E_{n-1} si :

$$\min_{1 \leqslant l \leqslant k} r_l \leqslant u^{\frac{1}{\omega_n}}$$

Si la condition est vérifié, on enlève l'observation X_{l^*} associé au minimum $r_{l^*} = \min_{1 \leqslant l \leqslant k} r_l$ et on le remplace par le point X_n pondéré par $u^{\frac{1}{\omega_n}}$.
Dans le cas uniforme, $\omega_n = 1$. Si on considère tous les nombres aléatoires u_i, et qu'on les trie par ordre croissant $(u_{\sigma(1)}, ..., u_{\sigma(n)})$. Dans ce cas, $\min_{1 \leqslant l \leqslant k} r_l = u_{\sigma(n-k+1)}$. Comme les nombres u_i sont tirés selon une loi uniforme, $\mathbb{P}(\min_{1 \leqslant l \leqslant k} r_l \leqslant u) = \frac{k}{n}$. On retrouve l'algorithme précédent. Cette probabilité ne change pas lorsque $\omega_n = C \ \forall n$ car les nombres u_i sont identiquement distribués.
On s'intéresse maintenant au cas où les poids ne sont pas identiques. On associe à chaque élément X_i de poids ω_i un nombre $r_i = u^{\frac{1}{\omega_i}}$ où u un nombre aléatoire appartient tiré selon une loi uniforme. On trie ces r_i par ordre croissant : $(r_{\sigma(1)}, ..., r_{\sigma(n)})$. A l'itération n, l'échantillon est $X_{\sigma(n-k+1)}, ..., X_{\sigma(n)}$.
La fonction de répartition d'une variable aléatoire est définie par $F(x) = \mathbb{P}(X \leqslant x)$. On suppose que $X \geqslant 0$, la loi de X^α est $F(X^\alpha) = \mathbb{P}(X^\alpha \leqslant x) = \mathbb{P}(X \leqslant x^{\frac{1}{\alpha}}) = F(x^{\frac{1}{\alpha}})$.

[6]: