expose_rwr_recommandation

January 8, 2019

1 3A.mr - Random Walk with Restart (système de recommandations)

Si la méthode de factorisation de matrices est la méthode la plus connue pour faire des recommandations, ce n'est pas la seule. L'algorithme Random Walk with Restart s'appuie sur l'exploration locale des noeuds d'un graphe et produit des résultats plus facile à intepréter.
Une des façons d'expliquer le PageRank est de modéliser Internet comme une immense chaîne de Markov. Le score PageRank correspond alors à la probabilité de rester dans un état ou un site Internet dans ce cas-là. Ce score est relié à la probabilité d'atterrir sur un site en suivant une marche aléatoire à travers les hyperliens. Et pour éviter les problèmes numériques lors du calcul, la formule fait apparaître un terme d qui correspond à la probabilité qu'un surfer a de continuer son chemin ou d'aller voir ailleurs sur un site qui n'a rien à voir : il fait un bond avec une probabilité $(1-d)$. Je ne réécris pas les formules, elles sont disponibles sur Wikipedia.
Maintenant, si on considère qu'un surfeur se ballade sur Internet de façon aléatoire mais qu'au lieu d'arrêter sa marche et d'aller n'importe ou ailleurs, il revient à son point de départ. Cela revient à étudier toutes les marches aléatoires partant du même noeud. On obtient alors des probabilités de rester dans des états qui dépendent de ce point de départ qu'on utilise pour faire des recommandations Fast Random Walk with Restart and Its Applications).
Je suppose qu'on a un graphe (G, V, E), V pour les noeuds, E pour les arcs. P représente la matrice de transition d'un noeud à l'autre. La somme des coefficients sur la même ligne fait 1 : $\sum_j P_{ij} = 1$. e est un vecteur avec que des 0 sauf pour une coordonnées i où i est le noeud de départ. c est la probabilité de revenir au point de départ. L'objectif est de trouver le régime transition π qui vérifie l'équation suivante :

$$\pi = (1-c)P'\pi + ce \Longleftrightarrow \pi = c(I-(1-c)P')^{-1}e$$

Le vecteur π qui en résulte donne un poids à chaque noeud du graphe et c'est ce poids dont on se sert pour ordonner les recommandations. Autrement dit, si un surfeur est sur un site i, on lui recommandera comme autre site ceux dont le poids est le plus fort dans le vecteur π. Le tout est savoir de le calculer. Exemple, on choisit pour P :

```
In [1]: import numpy as np
        from numpy.linalg import det
        P = np.matrix ( [[ 0,0.5,0,0.5],[0.5,0,0.5,0],[1./3,1./3,0,1./3],[0.1,0.9,0,0]])
        P
```

```
Out[1]: matrix([[ 0.        ,  0.5       ,  0.        ,  0.5       ],
                [ 0.5       ,  0.        ,  0.5       ,  0.        ],
                [ 0.33333333,  0.33333333,  0.        ,  0.33333333],
                [ 0.1       ,  0.9       ,  0.        ,  0.        ]])
```

```
In [2]: c = 0.15
        I = np.identity(4)
        e = np.matrix( [[ 0., 1., 0., 0. ]]).T
        pi = ((I-P.T*(1-c))).I * e * c
        pi
```

```
Out[2]: matrix([[ 0.24355828],
                [ 0.42249412],
                [ 0.17956   ],
                [ 0.1543876 ]])
```

Une autre de procéder est de considérer que le vecteur π est un point fixe de la suite : $\pi^t = (1-c)P\pi^{t-1} + ce$.

```
In [3]: pi = e
        for i in range(0,10):
            pi = P.T * pi * (1-c) + e * c
        pi
```

```
Out[3]: matrix([[ 0.24350522],
                [ 0.42215404],
                [ 0.17924793],
                [ 0.15509282]])
```

On retrouve sensiblement la même chose. Une dernière façon est d'utiliser des marches aléatoires avec restart ou Random Walk with Restart. Mais pour ce faire, on doit générer des marches aléatoires partant de i avec la probabilité de revenir au début égale à c.

```
In [4]: import random
        from numpy.random import multinomial

        def marche_alea(P,c,i):
            marche = [ i ]
            while True:
                r = random.random()
                if r <= c: return marche
                vect = P[i,:].tolist()[0]
                i = multinomial(1,vect,size=1).tolist()[0].index(1)
                marche.append(i)

        def aggregation(marches):
            count = {}
            for marche in marches:
                for i in marche : count[i] = count.get(i,0)+1.0
            s = sum( _ for _ in count.values())*1.0
            for i in count: count[i] /= s
            return [ count.get(i,0) for i in range(0,max(_ for _ in count.keys())+1) ]

        marches = [ marche_alea(P,c,1) for k in range(0,1000) ]
        count = aggregation(marches)
        count
```

```
Out[4]: [0.24443757725587145,
         0.4207354758961681,
         0.1773794808405439,
         0.15744746600741658]
```

On retrouve sensiblement les mêmes résultats. A quoi ça sert ? Selon les contextes, il est préférable d'utiliser tel ou tel algorithme pour calculer le vecteur π : l'inversion de la matrice, la suite récursive, ou Monte Carlo voire une combinaison. Voici quelques questions qu'il faut se poser.

- la matrice est grande (plusieurs millions de noeuds) ou petite (quelques milliers) ?

- la matrice est sparse ou creuse (ses coefficients sont presque tous nuls) ?
- on a besoin des valeurs pour seulement quelques noeuds et seulement les premières valeurs ?
- les calculs se font en parallèle sur la même machine (mémoire partagée) ou en map/reduce ?

Sur Internet, les matrices sont souvent très grande, très sparses excepté pour quelques noeuds qui sont comme des hubs et qu'il faut parfois traiter à part.