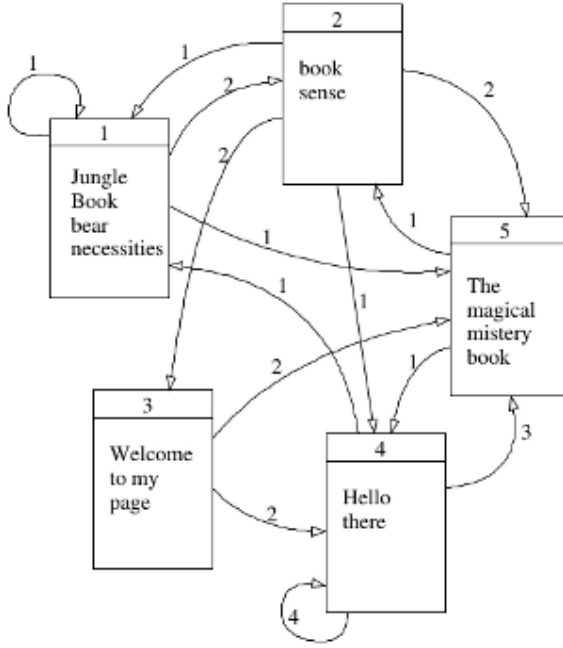


## Google Nasıl Isler?

Google arama motoruna bir kelime yazdigimizda geri gelen sonuclar nasıl kararlaştirilir? İlk akla gelen yontem tabii ki Web'deki tum sayfaların (milyarlarca sayfa) sayfalar üzerindeki kelimelerin o sayfa ile iliskilendirilmesi ve arama yapilınca kelimeye göre sayfa geri getirilmesi. Mesela alttaki ornekte “book (kitap)” yazınca geriye 1., 2. ve 5. sayfalar geri gelecek. Fakat hangi sirada? Bu sayfalardan hangisi digerlerinden daha onemli?



Google'in arama motorlarına getirdigi en büyük yeniliklerden biri PageRank algoritmasıdır. Bu algoritmanın temelinde daha fazla referans edilen sayfalar daha üstte cikması yatar. Hatta o referans eden sayfaların kendilerine daha fazla referans var ise bu etki ta en sondaki sayfaya kadar yansitilir, hatta bu zincir bastan sona her seviyede hesaplanabilir. Peki bu nasıl gerçekleştirilir?

PageRank Web sayfalarını bir Markov Zinciri olarak görür. Markov Zincirleri seri halindeki  $X_n, n = 0, 1, 2, \dots$  rasgele değişkenini modeller ve bu değişkenler belli sayıdaki konumların birinde olabilirler. Mesela konumları bir doğal sayı ile ilintilendirirsek  $X_n = i$  olabilir ki  $i = \{0, 1, \dots\}$  diye kabul edelim.

Markov Zincirlerinde (MZ)  $i$  konumundan  $j$  konumuna geçiş olasılığını,  $P_{ij}$ , biliriz ve bu  $P(X_{n+1} = j | X_n = i)$  olarak acilabilir. Acilimdan görüleceği üzere bir MZ sonraki adıma geçiş olasılığı için sadece bir önceki adıma bakar. Bu tür önce/sonra yapısındaki iki boyutlu hal, çok rahat bir şekilde matrisine çevirilebilir / gösterilebilir. Önceki konum satırlar, sonraki konum kolonlar olarak betimlenir mesela.

### Örnek

Bir sonraki günde yağmur yağmayacağını bir MZ olarak tasarlayalım. Bir sonraki günde yağmur yağmayacağını sadece bugün etkiliyor olsun. Eğer bugün yağmur

yagiyorsa yarın yağmur yağması 0.7, eğer bugün yağmıyor ise yarın yağması 0.4. MZ şöyle

$$P = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}$$

Gecis olasiliklarindan bahsettigimize gore ve elimizde sinirli / belli sayida konum var ise, bir MZ'nin her satirindaki olasiliklari toplami tabii ki 1'e esit olmalidir.

MZ'lerin ilginç bir özelliği  $n$  adım sonra  $i, j$  gecisinin  $P^n$  hesabıyla yapılabilmesidir. Yani  $P$ 'yi  $n$  defa kendisiyle carpip  $i, j$  kordinatina bakarsak  $n$  adım sonrasını rahatca görebiliriz. Bunun ispatını burada vermeyeceğiz.

Mesela üstteki örnekte, eğer bugün yağmur yagiyorsa 4 gün sonra yağmur yağma olasılığı nedir?

```
import numpy.linalg as lin
P = np.array([[0.7,0.3],[0.4,0.6]])
P4 = lin.matrix_power(P,4)
print P4

[[ 0.5749  0.4251]
 [ 0.5668  0.4332]]
```

Aradığımız gecis için kordinat 0,0'a bakıyoruz ve sonuç 0.5749. Numpy `matrix_power` bir matrisi istediğimiz kadar kendisiyle carpmamızı sağlıyor.

Duragan Dagilim (Stationary Distribution)

Eğer yağmur örneğindeki matrisi carpmaya devam edersek, mesela 8 kere kendisiyle carpsak sonuç ne olurdu?

```
import numpy.linalg as lin
P = np.array([[0.7,0.3],[0.4,0.6]])
P8 = lin.matrix_power(P,8)
print P8

[[ 0.57145669  0.42854331]
 [ 0.57139108  0.42860892]]
```

Dikkat edilirse, her satir bir deger yaklasmaya basladi. Bu deger MZ'nin duragan dagilimidir, belli kosullara uyan her MZ'nin boyle bir duragan dagilimi vardir. Bu kosullar MZ'nin periyodik olmayan (aperiodic) ve tekrar eden (recurrent) olmasidir. Bu sartlar çok “özel” sartlar degildir aslında, daha çok “normal” bir MZ'yi tarif ediyor diyebiliriz. Tüm konumları tekrar eden yapmak kolaydir, MZ tek bagli (singly connected) hale getirilir, yani her konumdan her diger konuma bir gecis olur, ve periyodik olmaması için ise MZ'ye olmadigi zamanlarda bir konumdan kendisine gecis saglanir (az bir gürültü üzerinden).

Nihayetinde duraganlik su denkleme sebebiyet verir,

$$\pi = \pi P$$

Burada duragan dagilim  $\pi$ 'dir. Bu denklem tanidik geliyor mu? Devrigini alarak soyle gosterelim, belki daha iyi taninir,

$$P^T \pi^T = \pi^T$$

Bir sey daha ekleyelim,

$$P^T \pi^T = 1 \cdot \pi^T$$

Bu ozdeger/vektor formuna benzemiyor mu? Evet! Bu form

$$Ax = \lambda x$$

MZ denklemi sunu soyluyor, 1 degerindeki ozdegere ait ozvektor bir MZ'nin duragan dagilimidir! Bu arada, MZ gecis matrisi  $P$ 'nin en buyuk ozdegerinin her zaman 1 oldugunu biliyoruz. Bu durumda en buyuk ozdegere ait ozvektoru hesaplamak yeterli olacaktır. Bunu yapmayi zaten *Lineer Cebir Ders 21*'de ogrenmistik, ust metot (power method) sayesinde bu hesap kolayca yapilabiliyor.

Simdi en basktaki Web sayfalarina ait gecisleri yazalim,

```
P = [[1./4, 2./4, 0, 0, 1./4],
      [1./6, 0, 2./6, 1./6, 2./6],
      [0, 0, 0, 2./4, 2./4],
      [1./8, 0, 0, 4./8, 3./8],
      [0, 1./2, 0, 1./2, 0]]
```

```
P = np.array(P)
```

```
print P
```

```
[[ 0.25      0.5      0.      0.      0.25      ]
 [ 0.16666667 0.      0.33333333 0.16666667 0.33333333]
 [ 0.         0.         0.         0.5      0.5      ]
 [ 0.125      0.         0.         0.5      0.375     ]
 [ 0.         0.5      0.         0.5      0.        ]]
```

Simdi ust metodu kullanarak duragan dagilimi hesaplayalim. Herhangi bir baslangic vektorunu  $P$  ile 20 kere carpmak yeterli olur.

```
import numpy.linalg as lin
x=np.array([.5, .3, .1, .1, 0]) # herhangi bir vektor
for i in range(20):
    x = np.dot(x,P)
print 'pi = ', x
```

```
pi = [ 0.10526316  0.18421053  0.06140351  0.38596491  0.2631579 ]
```

Not: Aslında cebirsel olarak  $P$ 'yi 20 kere kendisiyle carpma ve sonucu başlangıç vektörü ile bir kere carpma da düşünülebilirdi. Fakat 20 kere vektör / matris carpımları yapmak, 20 kere matris / matris carpımı yapmaktan daha verimli olacaktır. Büyük Veri ortamı için de bu söylenebilir.

Eğer kütüphane çağırısı kullanmak gerekirse,

```
import numpy.linalg as lin
evals, evec = lin.eig(P.T)
pi = evec[:,0] / np.sum(evec[:,0])
print np.abs(pi)
```

```
[ 0.10526316  0.18421053  0.06140351  0.38596491  0.26315789]
```

Sonuç gösteriyor ki “book” yazdığımızda Google bize 5. sayfayı en fazla olacak şekilde sonuç dondurmeli, çünkü onun duragan dağılımı 1,2,5 sayfalarının arasında en yüksek.

Duragan Dağılıma Bakış

MZ ve duragan dağılımın PageRank’le alakasını bir daha düşünelim. MZ ile  $n$  adım sonrasını hesaplayabiliyoruz, duragan dağılım ise sonsuz adım sonrasını ifade ediyor. Ve bu dağılım, bir anlamda, sonsuz yapılan adımlar sırasında *en fazla hangi konumlarda* zaman geçirileceğini gösteriyor. Konum yerine sayfa dersek duragan dağılımın niye en önemli sayfaları belirlemek için önemli olduğunu anlarız.

Kullanıcı herhangi bir sayfada iken hangi diğer sayfalara gideceği o sayfa üzerinde bağlantılar üzerinden anlaşılır, PageRank bu bağlantının mevcudiyetine bakar sadece, o mevcudiyet üzerinden bir geçiş olasılığı hesaplar, ve bu olasılığa göre (raslantısal şekilde) bağlantının takip edileceği düşünülür. Bu arada cöğünlükla sayfalar arasındaki bağlantıların ağırlığı 1 olacaktır, fakat örnek amaçlı 2,3 gibi sayılar da kullanılıyor.

Rasgele Sayfa Geçisi

Şimdi  $\pi^T$  yerine  $p$ ,  $P$  yerine  $N$  kullanalım, PageRank özdeğerli algoritması

$$p = N^T p$$

olarak gösterilebilir. Google veri temsili üzerinde bazı ekler yapmaktadır, mesela kullanıcının hiçbir bağlantı takip etmeyip tarayıcıya direkt URL girerek başka bir sayfaya zıplaması (teleporting) bir şekilde temsil edilmelidir. Ayrıca hiç dış bağlantı vermeyen sayfalar (olu noktalar) hesaba katılmalıdır.

Bu her iki durum için formül şu şekilde ikiye ayrılır,

$$p = (1 - d)N^T p + dN_f^T p$$

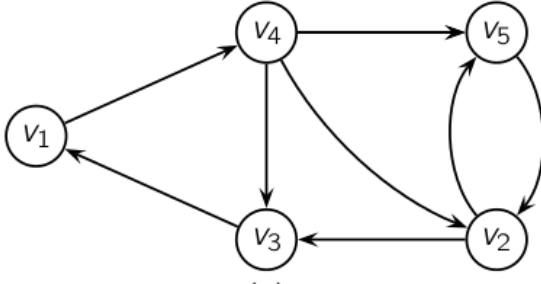
$$= ((1 - d)N^T + dN_f^T)p$$

$$= M^T p$$

ki,

$$M = (1 - d)N^T + dN_f^T$$

olacaktır.  $N_f$  bir normalize edilmiş “ziplama” matrisidir, yani her sayfadan her diğer sayfaya bir bağlantı “varmış gibi” yapar, mesela 5x5 boyutunda tüm öğeleri 0.20 olacaktır.  $d$  bir ağırlık sabitidir, Google’in bunu 0.85 olarak tanımladığı duyulmuştur, ve gerçek bağlantı matrisi ve rasgele ziplama matrisi arasında bir ağırlık tanımlar, her ikisinde de birazcık alarak (daha çok ana  $N$ ’den tabii) nihai matrisi oluşturur. Örnek olarak şu grafiğe bakalım,



```

N = [[0, 0, 0, 1., 0],
      [0, 0, 1./2, 0, 1./2],
      [1, 0, 0, 0, 0],
      [0, 1./3, 1./3, 0, 1./3],
      [0, 1, 0, 0, 0]]

```

```
N = np.array(N)
```

```
Nf = 0.20 * np.ones((5,5))
```

```
d = 0.85
```

```
M = d*N + (1-d)*Nf
```

```
x=np.array([.5, .3, .1, .1, 0]) # herhangi bir vektor
```

```
for i in range(20):
```

```
    x = np.dot(x,M)
```

```
print 'result = ', x
```

```
result = [ 0.18959094  0.24375097  0.18775335  0.19115138  0.18775335]
```

Sonuca göre  $v_2$  en yüksek PageRank değerine sahip.

[1] Murphy, K., CS340: Machine Learning Lecture Notes, [www.ugrad.cs.ubc.ca/~cs340](http://www.ugrad.cs.ubc.ca/~cs340)

- [2] Ross, S., Introduction to Probability Models, 8th Edition
- [3] Zaki, Maira, Fundamentals of Data Mining Algorithms