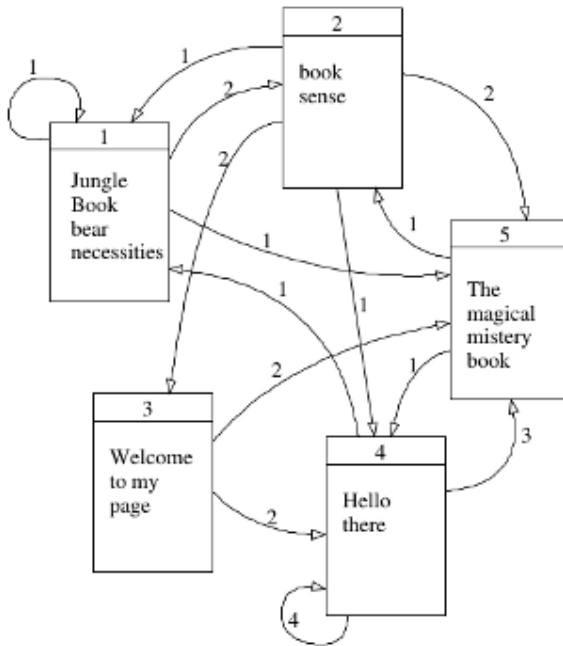


Google Nasıl Isler?

Lineer Cebir hocaları Google'a mutessekkir olmalı, çünkü bu unlu arama motorunun kullandığı PageRank tekniğinin özü aslında lineer cebirin temelini oluşturan kavramlardan özdeğer / özvektor hesabı. Öğrenciler “niye bu kavramları öğreniyoruz hocam?” diye sorunca artık cevap kolay: “bu yöntemi Google da kullanıyor!”.

Şimdi arama motorunun yapması gerekenleri düşünelim: Google'a bir kelime yazdığımızda geri gelen sonuçlar nasıl kararlaştırılır? İlk akla gelen yöntem tabii ki Web'deki tüm sayfaların (milyarlarca sayfa) sayfalar üzerindeki kelimelerin o sayfa ile ilişkilendirilmesi ve arama yapılırken kelimeye göre sayfa geri getirilmesi. Mesela alttaki örnekte “book (kitap)” yazınca geriye 1., 2. ve 5. sayfalar geri gelecek. Fakat hangi sırada? Bu sayfalardan hangisi diğerlerinden daha önemli?



PageRank'in temelinde daha fazla referans edilen sayfaların daha üstte çıkması yatar. Hatta o referans eden sayfaların kendilerine daha fazla referans var ise bu etki ta en sondaki sayfaya kadar yansıtılır, 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 acılabilir. Acılimdan 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 çevrilebilir / gösterilebilir. Önceki konum satırlar, sonraki konum kolonlar olarak betimlenir mesela.

Ornek

Bir sonraki gunde yagmur yagmayacagini bir MZ olarak tasarlayalim. Bir sonraki gunde yagmur yagmayacagini sadece bugun etkiliyor olsun. Eger bugun yagmur yagiyorsa yarın yagmur yagmasi 0.7, eger bugun yagmıyor ise yarın yagmasi 0.4. MZ soyle

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

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

MZ'lerin ilginç bir ozelligi n adim sonra i, j gecisinin P^n hesabiyla yapilabilmesidir. Yani P 'yi n defa kendisiyle carpip i, j kordinatina bakarsak n adim sonrasini rahatca gorebiliriz. Bunun ispatini burada vermeyecegiz.

Mesela ustteki ornekte, eger bugun yagmur yagiyorsa 4 gun sonra yagmur yagma olasiligi 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]]
```

Aradigimiz gecis icin kordinat 0,0'a bakiyoruz ve sonuc 0.5749. Numpy `matrix_power` bir matrisi istedigimiz kadar kendisiyle carpmamizi sagliyor.

Duragan Dagilim (Stationary Distribution)

Eger yagmur ornegindeki matrisi carpmaya devam edersek, mesela 8 kere kendisiyle carpsak sonuc 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 degere 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 cok "ozel" sartlar degildir aslinda, daha cok "normal" bir MZ'yi tarif ediyor diyebiliriz. Tum konumlari 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 olmadığı zamanlarda bir konumdan kendisine gecis sağlanır (az bir gürültü üzerinden).

Nihayetinde duraganlık şu denkleme sebebiyet verir,

$$\pi = \pi P$$

Burada duragan dağılım π 'dir. Bu denklem tanıdık geliyor mu? Devriginin alarak şöyle gösterelim, belki daha iyi tanınır,

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

Bir şey daha ekleyelim,

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

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

$$Ax = \lambda x$$

MZ denklemi sunu söylüyor, 1 değerindeki özdeğere ait özvektor bir MZ'nin duragan dağılımıdır! Bu arada, MZ gecis matrisi P 'nin en büyük özdeğerinin her zaman 1 olduğunu biliyoruz (çünkü üstteki tarif ettiğimiz özel şartlara sahip olan türden matrisler böyle özdeğerlere sahip olmalı). Bu durumda en büyük özdeğere ait özvektörü hesaplamak yeterli olacaktır. Bunu yapmayı zaten *Lineer Cebir Ders 21*'de öğrenmiştik, üst metot (power method) sayesinde bu hesap kolayca yapılabilir.

Şimdi en basktaki Web sayfalarına ait gecisleri yazalım,

```
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: Aslinda cebirsel olarak P 'yi 20 kere kendisiyle carpmak ve sonucu baslangic vektoru ile bir kere carpmak ta dusunulebilirdi. Fakat 20 kere vektor / matris carpimlari yapmak, 20 kere matris / matris carpimi yapmaktan daha verimli olacaktir. Buyuk Veri ortami icin de bu soylenebilir.

Neyse, eger ozvektor hesabini kendimiz elle yapmak yerine direk kutuphane cagrisi kullansaydik,

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

[ 0.10526316  0.18421053  0.06140351  0.38596491  0.26315789]
```

Ayni sonuca ulastik. Bu sonuc gosteriyor ki “book” yazdigizda Google bize 5. sayfayi en basta olacak sekilde sonuc dondurmeli, cunku onun duragan dagilimi 1,2,5 sayfalarinin arasinda en yuksek.

Duragan Dagilima Bakis

MZ ve duragan dagilimin PageRank'le alakasini bir daha dusunelim. MZ ile n adim sonrasini hesaplayabiliyoruz, duragan dagilim ise sonsuz adim sonrasini ifade ediyor. Ve bu dagilim, bir anlamda, sonsuz yapilan adimlar sirasinda *en fazla hangi konumlarda* zaman gecirilecegini gosteriyor. Konum yerine sayfa dersek duragan dagilimin niye en onemli sayfalari belirlemek icin onemli oldugunu anlariz.

Kullanici herhangi bir sayfada iken hangi diger sayfalara gidecegi o sayfa uzerinde baglantilar uzerinden anlasilir, PageRank bu baglantinin mevcudiyetine bakar sadece, o mevcudiyet uzerinden bir gecis olasiligi hesaplar, ve bu olasiliga gore (raslantisal sekilde) baglantinin takip edilecegi dusunulur. Bu arada cogunlukla sayfalar arasindaki baglantilarin agirligi 1 olacaktir, fakat ornek amaclı 2,3 gibi sayılar da kullaniliyor.

Rasgele Sayfa Gecisi

Google veri temsili uzerinde bazi ekler yapmaktadir, mesela kullanicinin hicbir baglanti takip etmeyip tarayiciya direk URL girerek baska bir sayfaya ziplamasi (teleporting) bir sekilde temsil edilmelidir. Ayrica hic disa baglanti vermeyen sayfalar (olu

noktalar) hesaba katılmalıdır. Simdi π^T yerine p , P yerine N kullanalım, PageRank ozyineli algoritması

$$p = N^T p$$

olarak gosterilebilir.

Bu her iki durum icin formül su sekilde ikiye ayirilir,

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

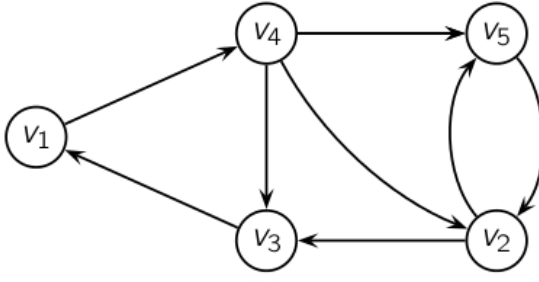
$$= ((1 - d)N^T + dN_f^T)p$$

$$= M^T p$$

ki,

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

olacaktır. N_f bir normalize edilmiş “ziplama” matrisidir, yani her sayfadan her diger sayfaya bir baglanti “varmis gibi” yapar, mesela 5x5 boyutunda tum ogeleri 0.20 olacaktır. d bir agirlik sabitidir, Google’in bunu 0.85 olarak tanımladigi duyulmustur, ve gercek baglanti matrisi ve rasgele ziplama matrisi arasinda bir agirlik tanımlar, her ikisinde de birazcik alarak (daha cok ana N ’den tabii) niahi matrisi olusturur. Ornek olarak su grafige bakalim,



```

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 gore v_2 en yuksek PageRank degerine sahip.

Mimari

Google tabii ki arama sonuclarini iyilestirmek icin yillar icinde diger ek fonksiyonlari motoruna ekledi. Duyumlarimiza gore artik PageRank gibi onlarca ek kriter kullanilmaktadir; fakat PageRank hala cok onemli ve sirketin kurulusu baglaminda Google'i Google yapan algoritmaydi, onun diger motorlara nazaran elindeki avantajı, en buyuk ilerlemesiydi.

Sistem kodlamasi acisindan PageRank'e tum Web sayfalari ve onların arasindaki iliskiler verilmelidir, bu milyarlarca sayfa ve onların arasindaki baglantilar demektir. Google bunu yapabilmek icin Web “agini” orumcek (spider) programlari ile surekli geziyor, ve bu veriyi alip, PageRank'e hesap icin aktariyor.

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

[2] Ross, S., Introduction to Probability Models, 8th Edition

[3] Zaki, Maira, Fundamentals of Data Mining Algorithms