Cok Degiskenli Normal Numaralari (Multivariate Normal Tricks)

Cok degiskenli normal dagilimlarla is yaparken, mesela Gaussian karisimlari kullanirken, bazi numaralari bilmek faydali olabiliyor. Bunlardan birincisi $(x-\mu)^T\Sigma^{-1}(x-\mu)$ hesabini yapmaktir, diger log-toplam-exp numarasi (logsumexp trick) diye bilinen hesaptir.

Birinciden baslayalim, daha kisalastirmak icin $y = x-\mu$ diyelim, yani $y^T\Sigma^{-1}y$ olsun. Simdi bu formulde bir ters alma (inversion) isleminin oldugunu goruyoruz. Fakat bu islem oldukca pahali bir islem olarak bilinir, hele hele boyutlarin yukseldigi durumlardan (binler, onbinler), kovaryansi temsil eden Σ, $n \times n$ olacaktir. Acaba tersini almayi baska bir sekilde gerceklestiremez miyiz?

Σ matrisi bir kovaryans matrisi oldugu icin simetrik, pozitif yari kesin bir matristir. Bu tur matrislerin Cholesky ayristirmasinin oldugunu biliyoruz ve bu islem cok hizli yapilabiliyor. O zaman

$$\Sigma = LL^T$$

ki L matrisi alt-ucgensel (lower triangular) bir matristir,

$$\Sigma^{-1} = (LL^T)^{-1}$$

$$= L^{-T}L^{-1}$$

Bunu temel alarak iki taraftan y'leri geri koyalim,

$$y^T\Sigma^{-1}y = y^TL^{-T}L^{-1}y$$

Bilindigi gibi lineer cebirde istedigimiz yere parantez koyabiliriz,

$$= (y^TL^{-T})L^{-1}y$$

Parantezden bir seyin devrigi gibi temsil edersek, parantez icindekilerin sirasi degisir ve tek tek devrigi alinir,

$$= (L^{-1}y)^TL^{-1}y$$

$$= |L^{-1}y|^2$$

Ustteki ifadede $|\cdot|$ icindeki kisim $Ax = b$ durumundaki x'in en az kareler cozumu olan $A^{-1}b$'ye benzemiyor mu? Evet. Gerci $n \times n$ boyutunda bir matris oldugu icin elimizde "bilinmeyenden fazla denklem" yok, yani bu sistem artık belirtilmiş (overdetermined) degil, yani en az kareler degil direk lineer sistem cozumu yapiyoruz. Her neyse, bu durumda her standart lineer cebir kutuphanesinde mevcut bir cagri ile $L^{-1}y$ hesabini yapabiliriz, mesela `solve` ile, ve bu cagrilar perde arkasinda ters alma isleminden kacinarak bir suru optimizasyon yaparak sonuca erismektedirler. Ustune ustluk L durumunda bu cok daha hizli olacaktir, cunku L alt-ucgensel oldugu icin cozum geriye deger koymak (back substitution) ile aninda bulunabilir. Geriye deger koymayi hatirlarsak, mesela

$$
\begin{bmatrix} 2 & 0 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 8 \end{bmatrix}
$$

En ust satirda her zaman tek bir bilinmeyen olacak, cunku matris alt ucgensel, en ust satir her zaman en bos satirdir. Bu tek bir esitlik demektir, yani $2x_1 = 6$, ki $x_1 = 3$. Bunu alip bir sonraki satira gideriz, artik x_1'i biliyoruz, sonraki satirda sadece x_2 bilinmeyen kaliyor, $3 \cdot x_1 + 4 \cdot x_2 = 8$, yani $x_2 = -1/4$. Sonuca ulastik. Daha fazla boyut olsaydi durum degismezdi, ayni islem daha fazla tekrarlanirdi. Bu arada bu turden bir cozumun ne kadar hizli olacagini belirtmemize gerek yok herhalde.

Demek ki $y^T \Sigma^{-1} y$ hesabi icin once Σ uzerinde Cholesky aliyoruz, sonra $L^{-1}y$ cozduruyoruz. Elde edilen degerin noktasal carpimini alinca Σ'nin tersini elde etmis olacagiz.

Ornek (once uzun yoldan),

```
import numpy.linalg as lin
Sigma = np.array([[10., 2.],[2., 5.]])
y = np.array([[1.],[2.]])
print np.dot(np.dot(y.T,lin.inv(Sigma)),y)
```

```
[[ 0.80434783]]
```

Simdi Cholesky ve `solve` uzerinden

```
L = lin.cholesky(Sigma)
x = lin.solve(L,y)
print np.dot(x.T,x)
```

```
[[ 0.80434783]]
```

Ayni sonuca eristik.

log-toplam-exp

Bu numaranin ilk kismi nisbeten basit. Bazi yapay ogrenim algoritmalari icin olasilik degerlerinin birbiriyle carpilmasi gerekiyor, mesela

$$
r = p_1 \cdot p_2 \dots p_n
$$

Olasiliklar 1'den kucuk oldugu icin 1'den kucuk degerlerin carpimi asiri kuculebilir, ve küçüklügun tasmasi (underflow) ortaya cikabilir. Eger carpim yerine log alirsak, carpimlar toplama donusur, sonra sonucu exp ile tersine ceviririz, ve log'u alinan degerler cok kuculmez, carpma yernie toplama islemi kullanildigi icin de nihai deger de kucukluge dogru tasmaz.

$$\log r = \log p_1 + \log p_2 + \ldots + \log p_n$$

$$r = \exp(\log p_1 + \log p_2 + \ldots + \log p_n)$$

Bir diger durum icinde *exp* ifadesi tasiyan bir olasilik degerinin cok kucuk degerler tasiyabilmesidir. Mesela cok degiskenli Gaussian karisimlari icin alttaki gibi bir hesap surekli yapilir,

$$= \sum_i w_i \frac{1}{(2\pi)^{k/2} \det(\Sigma)^{1/2}} \exp \left\{ -\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu) \right\}$$

ki $0 \leqslant w_i \leqslant 1$ seklinde bir agirlik degeridir. Ustteki formulun cogunlukla log'u alinir, ve, mesela bir ornek uzerinde gorursek (ve agirliklari bir kenara birakirsak),

$$\log(e^{-1000} + e^{-1001})$$

gibi hesaplar olabilir. Ustteki degerler tamamen uyduruk denemez, uygulamalarda pek cok kez karsimiza cikan degerler bunlar. Her neyse, eger ustteki ifadeyi kodla hesaplarsak,

```
print np.log(np.exp(-1000) + np.exp(-1001))
```

```
-inf
```

Bu durumdan kurtulmak icin bir numara sudur; exp ifadeleri arasinda en buyuk olanini disari cekersiniz, ve log'lar carpimi toplam yapar,

$$\log(e^{-1000}(e^0 + e^{-1}))$$

$$-1000 + \log(1 + e^{-1})$$

Bunu hesaplarsak,

```
print -1000 + np.log(1+np.exp(-1))
```

```
-999.686738312
```

Bu numaranin yaptigi nedir? Maksimumu disari cekerek en az bir degerin kucuklugu tasmamasini garantilemis oluyoruz. Ayrica, bu sekilde, geri kalan terimlerde de asiri ufalanlar terimler kalma sansi azaliyor.

Numerical Recipes, 3rd Edition

`http://makarandtapaswi.wordpress.com/2012/07/18/log-sum-exp-trick/`