

Naive Bayes

Reel sayılar arasında bağlantı kurmak için istatistikte regresyon kullanılır. Eger reel değerleri, (mesela) iki kategorik grup arasında seçmek için kullanmak istenirse, bunun için lojistik regresyon gibi teknikler de vardır.

Fakat kategoriler / gruplar ile başka kategorik gruplar arasında bağlantılar kurulmak istenirse, standart istatistik yöntemleri faydalı olamıyor. Bu gibi ihtiyaçlar için makine öğrenimi (machine learning) dünyasından Naive Bayes gibi tekniklere bakmamız lazım.

Öncelikle kategorik değerler ile ne demek istediğimizi belirtelim. Reel sayılar 0.3423, 2.4334 gibi değerlerdir, kategorik değerler ile ise mesela bir belge içinde 'a', 'x' gibi harflerin mevcut olmasıdır. Ya da, bir evin 'beyaz', 'gri' renkli olması.. Burada böyle kategorilerden bahsediyoruz ki istesek te onları sayısal bir değere çeviremiyoruz; kıyasla mesela bir gunun 'az sıcak', 'orta', 'çok sıcak' olduğu verisini kategorik bile olsa regresyon amacıyla sayıya çevirip kullanabilirdik. Az sıcak = 0, orta = 1, çok sıcak = 2 değerlerini kullanabilirdik, regresyon hala anlamlı olurdu (çünkü arka planda bu kategoriler aslında sayısal sıcaklık değerlerine tekabül ediyor olurlardı). Fakat 'beyaz', 'gri' değerlere sayı atamanın regresyon açısından bir anlamı olmazdı, hatta bunu yapmak yanlış olurdu. Eger elimizde fazla sayıda 'gri' ev verisi olsa, bu durum regresyon sırasında beyaz evlerin beyazlığını mı azaltacaktır?

İşte bu gibi durumlarda kategorileri olduğu gibi işleyebilen bir teknik gerekiyor. Bu yazıda kullanacağımız örnek, bir belgenin içindeki kelimelere göre kategorize edilmesi. Elimizde iki türlü doküman olacak. Bir tanesi Stephen Hawking adlı bilim adamının bir kitabından 3 sayfa, diğeri başkan Barack Obama'nın bir kitabından 3 sayfa. Bu sayfalar ve içindeki kelimeler NB yöntemini "eğitmek" için kullanılacak, sonra NB tarafından hiç görülmemiş yeni sayfaları yöntemimize kategorize ettireceğiz.

Cok Boyutlu Bernoulli ve Kelimeler

			x^1	x^2	x^3	x^4
Dim1:	"the"	=	1	0	1	1
Dim2:	"hello"	=	0	1	0	1
Dim3:	"and"	=	1	1	0	1
Dim4:	"happy"	=	1	0	0	1

Bir doküman ile içindeki kelimeler arasında nasıl bağlantı kuracağız? Burada olasılık teorisinden Çok Boyutlu Bernoulli (Multivariate Bernoulli) dağılım-

ini kullanacagiz. Ustteki resimde goruldugu gibi her dokuman bir x^i rasgele degiskeniyle temsil edilecek. Tek boyutlu Bernoulli degiskeni '1' ya da '0' degerine sahip olabilir, cok boyutlu olani ise bir vektor icinde '1' ve '0' degerlerini tasiyabilir. Iste bu vektorun her hucre, onceden tanimli bir kelimeye tekabul edecek, ve bu kelimedenden bir dokuman icinde en az bir tane var ise, o hucre '1' degerini tasiyacak, yoksa '0' degerini tasiyacak. Ustteki ornekte 2. kelime “hello” ve 4. dokuman icinde bu kelimedenden en az bir tane var, o zaman $x_2^4 = 1$. Tek bir dokumani temsil eden dagilimi matematiksel olarak soyle yazabiliriz:

$$p(x_1, \dots, x_D) = \prod_{d=1}^D p(x_d) = \prod_{d=1}^D \alpha_d^{x_d} (1 - \alpha_d)^{1-x_d}$$

Bu formilde her d boyutu bir tek boyutlu Bernoulli, ve bir dokuman icin tum bu boyutlari birlesik (joint) dagilimi gerekiyor, carpimin sebebi bu. Formildeki α_d bir dagilimi “tanimlayan” deger, α bir vektor, ve unutmayalim, her “sinif” icin NB ayri ayri egitilecek, ve her sınıf icin farkli α vektoru olacak. Yani Obama’nin kitaplari icin $\alpha_2 = 0.8$ olabilir, Hawking kitabi icin $\alpha_2 = 0.3$ olabilir. Birinin kitabinda “hello” kelimesi olma sansi fazla, digerinde pek yok. O zaman NB’yi “egitmek” ne demektir? Egitmek her sınıf icin yukaridaki α degerlerini bulmak demektir.

Bunun icin istatistikteki “olurluk (likelihood)” kavramini kullanmak yeterli. Olurluk, bir dagilimdan geldigi farzedilen bir veri setini alir, tum veri noktalarını teker teker olasiliga gecerek olasilik degerlerini birbirine carpar. Sonuc ne kadar yuksek cikarsa, bu verinin o dagilimdan gelme olasiligi o kadar yuksek demektir. Bizim problemimiz icin tek bir sinifin olurlugu, o sınıf icindeki tum (N tane) belgeyi kapsamalidir, tek bir “veri noktası” tek bir belgedir, o zaman:

$$L(\theta) = \prod_{i=1}^N \prod_{d=1}^D p(x_d^i) = \prod_{i=1}^N \prod_{d=1}^D \alpha_d^{x_d^i} (1 - \alpha_d)^{1-x_d^i}$$

θ bir dagilimi tanımlayan her türlü degisken anlamında kullanildi, bu ornekte icinde sadece α var.

Devam edelim: Eger α ’nin ne oldugunu bilmiyorsak (ki bilmiyoruz -egitmek zaten bu demek-) o zaman maksimum olurluk (maximum likelihood) kavramini resme dahil etmek gerekli. Bunun icin ustteki olurluk formülünün α ’ya gore turevini alip sifra esitlersek, bu formulden bir maksimum noktasındaki α

elimize gececektir. Iste bu α bizim aradigimiz deger. Veriyi en iyi temsil eden α degeri bu demektir. Onu bulunca egitim tamamlanir.

Turev almadan once iki tarafın \log 'unu alalim, boylece carpimlar toplamlara donusecek ve turevin formülün icine nufuz etmesi daha kolay olacak.

$$\log(L) = \sum_{i=1}^N \sum_{d=1}^D x_d^i \log(\alpha_d) + (1 - x_d^i) \log(1 - \alpha_d)$$

Turevi alalim:

$$\frac{d\log(L)}{d\alpha_d} = \sum_{i=1}^N \left(\frac{x_d^i}{\alpha_d} - \frac{1 - x_d^i}{1 - \alpha_d} \right) = 0$$

1) α_d 'ye gore turev alirken x_d^i 'ler sabit sayi gibi muamele gorurler. 2) \log 'un turevi alirken \log icindeki degerlerin turev alinmis hali bolumun ustune, kendisini oldugu gibi bolum altina alinir, ornek $d\log(-x)/dx = -1/x$ olur ustteki eksi isaretinin sebebi bu.

Peki $\sum_{d=1}^D$ nereye gitti? Turevi α_d 'ye gore aliyoruz ve o turevi alirken tek bir α_d ile ilgileniyoruz, mesela α_{22} , bunun haricindeki diger tum α_d degerleri turev alma islemi sirasinda sabit kabul edilirler, turev sirasinda sifirlanirlar. Bu sebeple $\sum_{d=1}^D$ icinde sadece bizim ilgilendigimiz α_d geriye kalir. Tabii ki bu ayni zamanda her $d = 1, 2, \dots, D$, α_d icin ayri bir turev var demektir, ama bu turevlerin hepsi birbirine benzerler, yani tek bir α_d 'yi cozmek, hepsini cozmek anlamina gelir.

Devam edelim:

$$\sum_{i=1}^N \left(\frac{x_d^i}{\alpha_d} - \frac{1 - x_d^i}{1 - \alpha_d} \right) = \frac{N_d}{\alpha_d} - \frac{N - N_d}{1 - \alpha_d} = 0$$

$\sum_{i=1}^N x_d^i = N_d$ olarak kabul ediyoruz, N_d tum veri icinde d boyutu (kelimesi) '1' kac tane hucre oldugunu bize soyler. x_d^i ya '1' ya '0' olabildigine gore bir d icin, tum N hucrenin toplami otomatik olarak bize kac tane '1' oldugunu soyler. Sonra:

$$\frac{N_d}{\alpha_d} - \frac{N - N_d}{1 - \alpha_d} = 0$$

$$\frac{1 - \alpha_d}{\alpha_d} = \frac{N - N_d}{N_d}$$

$$\frac{1}{\alpha_d} - 1 = \frac{N}{N_d} - 1$$

$$\frac{1}{\alpha_d} = \frac{N}{N_d}$$

$$\alpha_d = \frac{N_d}{N}$$

Python Kodu

α_d 'nin formülünü buldumuza göre artık kodu yazabiliriz. İlk önce bir dokümanı temsil eden çok boyutlu Bernoulli vektörünü ortaya çıkartmamız lazım. Bu vektörün her hücresi belli bir kelime olacak, ve o kelimelerin ne olduğunu önceden kararlaştırmamız lazım. Bunun için her sınıftaki tüm dokümanlardaki tüm kelimeleri içeren bir sözlük yaratırız:

```
words = {}
```

```
# find all words in all files , creating a  
# global dictionary.
```

```
for file in [ 'a1.txt ', 'a2.txt ', 'a3.txt ',  
              'b1.txt ', 'b2.txt ', 'b3.txt ' ]:  
    f = open ( file )  
    s = f.read()  
    tokens = re.split( '\W+', s )  
    for x in tokens: words[x] = 0.
```

Dosyalar 'a1.txt', 'a2.txt', 'a3.txt' Hawking kitabından sayfalar, 'b1.txt', 'b2.txt', 'b3.txt' ise Obama sayfaları. Ana sözlüğü yarattığımıza göre artık her yazar için ayrı eğitim yapabiliriz. Hawking için

```
hawking_alphas = words.copy()  
for file in [ 'a1.txt ', 'a2.txt ', 'a3.txt ' ]:  
    words_hawking = set()  
    f = open ( file )  
    s = f.read()  
    tokens = re.split( '\W+', s )  
    for x in tokens:  
        words_hawking.add(x)  
    for x in words_hawking:  
        hawking_alphas[x] += 1.
```

Niye `set()` kullandik? Bilindigi gibi `set()` objesi bir kume demektir, kume mantigina gore bir eleman kumede var ise, onu bir daha eklemek kume icerigini degistirmez. Bu tam bize gore, cunku tek bir belgeye bakarken, bir kelime “en az bir kere” var ise o kelimenin hucresi '1' olacak, tek belge icinde kelimenin kac kere cikacagi bizi ilgilendirmiyor. Sonra bu kumeye bakiyoruz ve kumedeki her kelimeye tekabul eden hucreyi `hawking_alphas[x] += 1.` ile '1' arttiriyoruz. Obama sayfaları için yapılan işlem aynı. Sonra her iki alpha vektorunu toplam dokuman sayısına (3) boluyoruz cunku $\alpha_d = N_d/N$.

```
for x in hawking_alphas.keys():
    hawking_alphas[x] = hawking_alphas[x] / 3.
for x in obama_alphas.keys():
    obama_alphas[x] = obama_alphas[x] / 3.
```

Bu noktada hic gorulmemis yeni bir dokumani siniflamaya haziriz. Diyelim ki `a4.txt` adli bir dosyamiz var, biz bunun Hawking’in kitabindan bir sayfa oldugunu biliyoruz (ama program bilmiyor) ve `test('a4.txt')` cagrisini yaparak dogru sonucu almayi umuyoruz.

```
def prob(xd, alpha):
    return math.log(alpha*xd + 1e-10) + \
        math.log((1.-alpha)*(1.-xd) + 1e-10)

def test(file):
    test_vector = words.copy()
    words_test = set()
    f = open(file)
    s = f.read()
    tokens = re.split('\W+', s)
    for x in tokens:
        words_test.add(x)
    for x in words_test:
        test_vector[x] = 1.
    ob = 0.
    ha = 0.
    for x in test_vector.keys():
        if x in obama_alphas:
            ob += prob(test_vector[x], obama_alphas[x])
        if x in hawking_alphas:
```

```

ha += prob(test_vector[x], hawking_alphas[x])

print "obama", ob, "hawking", ha, "obama", ..

print "hawking_test"
test('a4.txt')

```

Test için yeni dokümanı kelimelerine ayırıyoruz, ve her kelimeye tekabül eden alpha vektorlerini kullanarak bir yazar için toplam olasılığı hesaplıyoruz. Nasıl? Her kelimeyi $\alpha_d^{x_d}(1 - \alpha_d)^{1-x_d}$ formülüne soruyoruz, yeni dokümanı temsilen elimizde bir $[1, 0, 0, 1, 0, 0, \dots, 1]$ şeklinde bir vektor olduğunu farz ediyoruz, buna göre mesela $x_1 = 1, x_2 = 0$. Eğer bir d kelimesi yeni belgede “var” ise o kelime için $x_d = 1$ ve bu durumda $\alpha_d^{x_d} = \alpha_d^1 = \alpha_d$ haline gelir, ama formülün oteki tarafı yok olur, $(1 - \alpha_d)^{1-x_d} = (1 - \alpha_d)^0 = 1$, o zaman $\alpha_d \cdot 1 = \alpha_d$.

Carpım diyoruz ama biz aslında sınıflama sırasında $\alpha_d^{x_d}(1 - \alpha_d)^{1-x_d}$ carpımı yerine yine $\log()$ numarasını kullandık; çünkü olasılık değerleri hep 1’e eşit ya da ondan küçük sayılardır, ve bu küçük değerlerin birbiriyle sürekli carpımı nihai sonucu asiri fazla küçültür. Asiri ufak değerlerle uğraşmamak için olasılıkların \log ’unu alıp birbirleri ile toplamayı seçtik, yani hesapladığımız değer $x_d \cdot \log(\alpha_d) + (1 - x_d) \cdot \log(1 - \alpha_d)$

Fonksiyon `prob` içindeki `1e-7` kullanımı neden? Bu kullanım \log numarisini yapabilmek için – sıfır değerinin \log değeri tanımsızdır, bir kelime olmadığı zaman \log ’a sıfır geleceği için hata olmaması için \log içindeki değerlere her seferinde yeterince küçük bir sayı ekliyoruz, böylece pur sıfırla uğraşmak zorunda kalmıyoruz. Sıfır olmadığı zamanlarda çok eklenen çok küçük bir sayı sonucta büyük farklar (hatalar) yaratmıyor.

Toparlarsak, yeni belge ‘a4.txt’ için iki tur alpha değerleri kullanarak iki farklı \log toplamını hesaplatıyoruz. Bu iki toplamı birbiri ile karşılastırıyoruz, hangi toplam daha büyükse, dokümanın o yazardan gelmesi daha olasıdır, ve o seçimimiz o yazar olur.

Kodun geri kalanı `naive.py` dosyasında bulunabilir, test çağrılarına bakılırsa, yeni test dokümanlarının doğru şekilde sınıflandığı görülecektir.

```

import re
import math

```

```

words = {}

# find all words in all files , creating a
# global dictionary.
for file in [ 'a1.txt', 'a2.txt', 'a3.txt',
               'b1.txt', 'b2.txt', 'b3.txt' ]:
    f = open (file)
    s = f.read()
    tokens = re.split( '\W+', s)
    for x in tokens: words[x] = 0.

hawking_alphas = words.copy()
for file in [ 'a1.txt', 'a2.txt', 'a3.txt' ]:
    words_hawking = set()
    f = open (file)
    s = f.read()
    tokens = re.split( '\W+', s)
    for x in tokens:
        words_hawking.add(x)
    for x in words_hawking:
        hawking_alphas[x] += 1.

obama_alphas = words.copy()
for file in [ 'b1.txt', 'b2.txt', 'b3.txt' ]:
    words_obama = set()
    f = open (file)
    s = f.read()
    tokens = re.split( '\W+', s)
    for x in tokens:
        words_obama.add(x)
    for x in words_obama:
        obama_alphas[x] += 1.

for x in hawking_alphas.keys():
    hawking_alphas[x] = hawking_alphas[x] / 3.
for x in obama_alphas.keys():
    obama_alphas[x] = obama_alphas[x] / 3.

```

```

def prob(xd, alpha):
    return math.log(alpha*xd + 1e-10) + \
           math.log((1.-alpha)*(1.-xd) + 1e-10)

def test(file):
    test_vector = words.copy()
    words_test = set()
    f = open(file)
    s = f.read()
    tokens = re.split('\W+', s)
    for x in tokens:
        words_test.add(x)
    for x in words_test:
        test_vector[x] = 1.
    ob = 0.
    ha = 0.
    for x in test_vector.keys():
        if x in obama_alphas:
            ob += prob(test_vector[x], obama_alphas[x])
        if x in hawking_alphas:
            ha += prob(test_vector[x], hawking_alphas[x])

    print "obama", ob, "hawking", ha, \
          "obama", ob > ha, "hawking", ha > ob

print "hawking_test"
test('a4.txt')
print "hawking_test"
test('a5.txt')
print "obama_test"
test('b4.txt')
print "obama_test"
test('b5.txt')

```

Kaynaklar

Jebara, T., Columbia U., *COMS 4771 Machine Learning* Lecture Notes, Lecture 7