

Naive Bayes

Reel sayılar arasında bağlantı kurmak için istatistikte regresyon kullanılır. Eğer 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.

Not: Daha ilerlemeden belirtelim, bu tekniğin ismi Naive Bayes ama bu tanım doğru değil, çünkü bu teknik Olasılık Teorisi'nden bilinen Bayes Teorisini kullanmıyor.

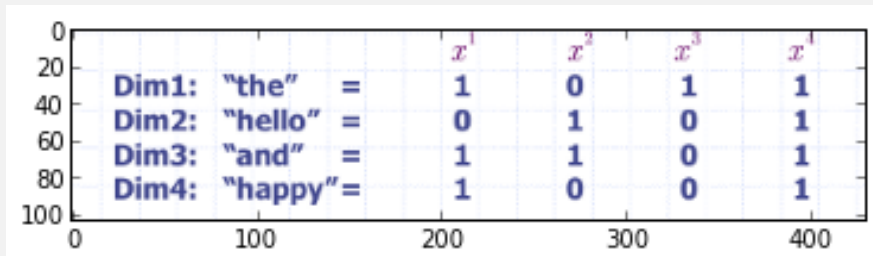
Öncelikle kategorik değerler ile ne demek istediğimizi belirtelim. Reel sayılar 0.3423, 2.4334 gibi değerlerdir, kategorik değerler ile ise mesela bir belge içinde 'a', 'x' gibi harflerin mevcut olmasıdır. Ya da, bir evin 'beyaz', 'gri' renkli olması.. Burada böyle kategorilerden bahsediyoruz ki istesek 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. Eğer 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

```
im=plt.imread("dims.png"); imshow(im)
```

<matplotlib.image.AxesImage at 0xb25bb4c>



Bir doküman ile içindeki kelimeler arasında nasıl bağlantı kuracağız? Burada olasılık teorisinden Çok Boyutlu Bernoulli (Multivariate Bernoulli) dağılımını kullanacağız. Üstteki resimde görüldüğü

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 tasivabilir. Iste bu vektorun her hucre, onceden tanimli bir kelimeye tekabul edecek, ve bu kelimeden bir dokuman icinde en az bir tane var ise, o hucre ‘1’ degerini tasivacak, yoksa ‘0’ degerini tasivacak. Ustteki ornekte 2. kelime “hello” ve 4. dokuman icinde bu kelimeden en az bir tane var, o zaman $x_2^4 = 1$. Tek bir dokumani temsil eden dagilimi matematiksel olarak soyle yazabiliriz:

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

Bu formilde her d boyutu bir tek boyutlu Bernoulli, ve bir dokuman icin tum bu boyutların ortak (joint) dagilimi gerekiyor, carpimin sebebi bu. Formildeki α_d bir dagilimi “tanımlayan” deger, α bir vektor, ve unutmayalım, her “sinif” icin NB ayrı ayrı egitilecek, ve her sınıf icin farklı α vektörü olacak. Yani Obama’nın kitapları icin $\alpha_2 = 0.8$ olabilir, Hawking kitabı icin $\alpha_2 = 0.3$ olabilir. Birinin kitabında “hello” kelimesi olma sansi fazla, diğlerinde pek yok. O zaman NB’yi “egitmek” ne demektir? Egitmek her sınıf icin yukarıdaki α degerlerini bulmak demektir.

Bunun icin istatistikteki “olurluk (likelihood)” kavramını kullanmak yeterli. Olurluk, bir dagilimdan geldiği farzedilen bir veri setini alır, tum veri noktalarını teker teker olasılığa gecerek olasılık degerlerini birbirine carpar. Sonuc ne kadar yuksek cikarsa, bu verinin o dagilimdan gelme olasılığı o kadar yuksek demektir. Bizim problemimiz icin tek bir sınıfın olurluğu, o sınıf icindeki tum (N tane) belgeyi kapsamalıdır, 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 kullanıldı, bu ornekte icinde sadece α var.

Devam edelim: Eger α ’nın ne olduğunu bilmiyorsak (ki bilmiyoruz -egitmek zaten bu demek-) o zaman maksimum olurluk (maximum likelihood) kavramını resme dahil etmek gerekli. Bunun icin ustteki olurluk formülünün α ’ya göre turevini alıp sifra esitlersek, bu formulden bir maksimum noktadaki α elimize gececektir. Iste bu α bizim aradığımız deger. Veriyi en iyi temsil eden α degeri bu demektir. Onu bulunca eğitim tamamlanır.

Turev almadan önce iki tarafın log’unu alalım, böylece carpımlar 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 alalım:

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

1- α_d ’ye göre turev alırken x_d^i ’ler sabit sayı gibi muamele görürler. 2- log’un turevi alırken log icindeki degerlerin turev alınmış hali bolumun ustune, kendisini olduğu gibi bolum altına alınır, örnek $d\log(-x)/dx = -1/x$ 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 formulu buldumuza gore artik kodu yazabiliriz. Ilk once bir dokumani temsil eden cok boyutlu Bernoulli vektorunu ortaya cikartmamiz lazim. Bu vektorun her hucresi belli bir kelime olacak, ve o kelimelerin ne oldugunu onceden kararlasmamiz lazim. Bunun icin her siniftaki tum dokumanlardaki tum kelimeleri iceren bir sozluk yaratiriz:

```
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')

```

```

hawking test
obama -34048.7734496 hawking -32192.3692113 obama False hawking True
hawking test
obama -33027.3182425 hawking -32295.7149639 obama False hawking True
obama test
obama -32531.9918709 hawking -32925.037558 obama True hawking False
obama test
obama -32205.4710748 hawking -32549.6924713 obama True hawking False

```

Test için yeni dokumani 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 dokumani 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 uğrasmamak 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 uğrasmak 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, dokumanın o yazardan gelmesi daha olasıdır, ve o seçiminiz o yazar olur.

Anahtarlama (Hashing) Numarasi

Ustteki kodda bir problem var, dokumani temsil eden ve içinde 1 ya da 0 hucrelili özellik vektorunu (feature vector) oluşturmak için tüm kelimelerin ne olduğunu bilmeliyiz. Yani veriyi bir kere bastan sonra tarayarak bir sözlük oluşturmaliyiz (ki öyle yapmaya mecbur kaldık) ve ancak ondan sonra her dokuman için hangi kelimenin olup olmadığını saptamaya ve onu kodlamaya başlayabiliriz. Halbuki

belgelere bakar bakmaz, teker teker giderken bile hemen bir ozellik vektörü oluşturabilseydik daha iyi olmaz mıydı?

Bunu basarmak için anahtarlama numarasını kullanmamız lazım. Bilindiği gibi temel yazılım bilimine göre bir kelimeyi temsil eden bir anahtar (hash) üretebiliriz, ki bu hash değeri bir sayıdır. Elimizde bir “sayı” olması bize faydalı olur yarar, bu sayının en fazla kaç olabileceğinden hareketle (hatta bu sayıya bir limit koyarak) özellik vektörümüzün boyutunu önceden saptamış oluruz. Sonra kelimeye bakarız, hash üretiriz, sonuç mesela 230 geldi, o zaman özellik vektöründeki 230’uncu kolonun değerini 1 yaparız.

```
d_input = dict()

def add_word(word):
    hashed_token = hash(word) % 127
    d_input[hashed_token] = d_input.setdefault(hashed_token, 0) + 1

add_word("obama")
d_input
```

```
{28: 1}
```

```
add_word("politics")
d_input
```

```
{28: 1, 123: 1}
```

Ustteki kodda bunun örneğini görüyoruz. Hash sonrası mod uyguladık (yüzde işareti ile) ve hash sonucunu en fazla 127 olacak şekilde sınırladık. Sözlük (dictionary) yavaş yavaş büyüyebiliyor. Potansiyel problemler ne olabilir? Hashing mükemmel değildir, çarpışma (collision) olması mümkündür yani nadiren farklı kelimelerin aynı numaraya eşlenebilmesi durumu. Bu problemleri iyi bir anahtarlama algoritması kullanarak, mod edilen sayıyı büyük tutarak çözmek mümkündür, ya da bu tür nadir çarpışmalar “kabul edilir hata” olarak addedilebilir.

Pandas kullanarak bir Dataframe’i otomatik olarak anahtarlamak istersek,

```
import pandas as pd
import numpy as np
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}

data = pd.DataFrame(data)
data
```

| | pop | state | year |
|---|-----|-------|------|
| 0 | 1.5 | Ohio | 2000 |
| 1 | 1.7 | Ohio | 2001 |
| 2 | 3.6 | Ohio | 2002 |

```
3 2.4 Nevada 2001
4 2.9 Nevada 2002
```

Simdi bu veri uzerinde sadece eyalet (state) icin bir anahtarlama numarasi yapalim

```
def hash_col(df, col, N):
    cols = [col + "_" + str(i) for i in range(N)]
    def xform(x):
        tmp = [0 for i in range(N)]
        tmp[hash(x) % N] = 1
        return pd.Series(tmp, index=cols)
    df[cols] = df[col].apply(xform)
    return df.drop(col, axis=1)

print hash_col(data, 'state', 4)
```

```
pop  year  state_0  state_1  state_2  state_3
0  1.5  2000         0         1         0         0
1  1.7  2001         0         1         0         0
2  3.6  2002         0         1         0         0
3  2.4  2001         0         0         0         1
4  2.9  2002         0         0         0         1
```

Kaynaklar

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

http://scikit-learn.org/dev/modules/feature_extraction.html