

Yaklasiksal SVD ile Tavsiye Sistemleri

Gecmis verilere bakarak bir kullanicinin hic seyretmedigi bir filme nasil not verecegini tahmin etmek unlu Netflix yarismasinin konusuydu. Onceki bir yazi *SVD, Toplu Tavsiye*'de benzer bir veri seti Movielens üzerinde SVD uygulayarak once boyut azaltmistik, azaltilmis boyut uzerinden yeni (o film icin notu bilinmeyen) bir kullanicinin diger mevcut kullanicilara mesafesini hesaplamis, ve boylece begeni acisindan en cok benzedigi diger kullaniciyi bulmustuk (birkac tane de bulunabilir). Bu kullanicinin bir film icin verdigi notu yeni kullanici icin tahmin olarak baz almistik.

SVD uygulamanin tek yontemi bu degil. Netflix yarismasinda kullanilan [1] bir yaklasim soyle; alttaki SVD ayristirmasina bakalim,

$$\begin{matrix} & n \\ \begin{matrix} m \\ \end{matrix} & \begin{matrix} \blacksquare \\ A \end{matrix} \end{matrix} = \begin{matrix} & r \\ \begin{matrix} m \\ \end{matrix} & \begin{matrix} \blacksquare \\ P \end{matrix} \end{matrix} \begin{matrix} & r \\ \begin{matrix} r \\ \end{matrix} & \begin{matrix} \blacksquare \\ S \end{matrix} \end{matrix} \begin{matrix} & n \\ \begin{matrix} r \\ \end{matrix} & \begin{matrix} \blacksquare \\ Q \end{matrix} \end{matrix}$$

1. kullanicini 1. filme verdigi not ustte koyu gosterilen satirlarin carpimi ile oluyor, eger ufak harfler ve kullanıcı (user) için u , film için i indisini kullanırsak, ve q, p vektorlerini Q, P matrislerinin sirasiyla kolon ve satirlarini gosterme için kullanırsak, ayristirma sonrası begeni değeri (onemli bir kısmı daha dogrusu) $q_i^T p_u$ carpimindadir. Carpim icinde S 'ten gelecek tekil değeri (singular value) ne olacak? Simdi formülasyonu biraz degistirelim, bu değeri carpim disina alarak birkac toplam olarak gosterebiliriz. Bu toplamlar mesela bir kullanicinin ne kadar yanlı (bias) not verdigini, ya da bir filmin kabaca, ortalama nasil not almaya meyilli oldugunu modelleyebilirler (ki bu da bir yanlılık olcusu). Ayrica tum filmlere verilen notların yanlıligi da olculebilir. Tum bunlari bir araya koyarsak, bir begeni notunu tahmin edecek formül soyle gosterilebilir,

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u$$

μ bir skalar, tum filmlere verilen ortalamayı gosteriyor, ki tum begenilerin sayisal ortalaması uzerinden basit bir sekilde hizla hesaplanabilir. \hat{r}_{ui} 'ya bir tahmin dedik cunku modelimizdeki vektorlerin değerlerini bulduktan sonra (egitim verisiyle bu hesabi yapacagiz) modeli kullanarak gercek not r_{ui} için bir tahmin yapmaya ugrasacagiz.

Yanlılık hakkında bazı örnekler vermek gerekirse, diyelim ki kullanıcı Bob not verirken yüksek seviyede oy vermeye meyilli. Bu durumda bu kullanicinin ortalama hatta dusuk oy vermesi onun bir film den hakikaten hic hoslanmadigini sinyallebilir. Ya da mesela bir film, genellikle ortalama oy almaktadır, bu du-

rumda ona cok iyi not veren bir kisinin bu filmi cok begendigi ortaya cilar. Modeldeki yanlilik parametreleri bu durumu saptayabilirler.

Egitim

Egitim icin ne yapmali? Minimize edecegimiz bir hedef fonksiyonu kuralim, ki cogunlukla bu karesi alinmis hata ile olur. Mesela gercek not r_{ui} degerinden tahmin notu \hat{r}_{ui} 'yi cikartip karesini alabiliriz. Bu islemi tum u, i 'ler icin yaparak sonuclari toplariz, ve bu toplami minimize etmeye ugrasabiliriz. Yani

$$\begin{aligned} & \min_{b^*, q^*, p^*} \sum_{u, i} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2) \\ &= \min_{b^*, q^*, p^*} \sum_{u, i} (r_{ui} - \mu - b_i - b_u - q_i^T p_u)^2 + \lambda(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2) \end{aligned}$$

Kisaltma olarak e_{ui} tanımlayalım, bu faydalı olabilir, formüldeki ilk parantez içindeki kismda e_{ui} kullanmak uzere,

$$e_{ui} := r_{ui} - \hat{r}_{ui}$$

λ ile carpilan bolum regularizasyon icin. Istatistik, yapay ogrenim, optimizasyon alanlarında modelimizin asiri uygunluk (overfitting) yapmasini engellemek icin regularizasyon kullanilir, bunun icin istedigimiz degiskenlerin fazla buyumesini cezalandiririz, ustteki minimizasyon modelinde bu ceza icin tum degerlerin buyuklugunu (magnitude) hesapladik -skalar degerlerin karesini, vektor degerlerinin kare norm'unu alarak- ve bu buyuklukleri bizim disaridan set edebilecegimiz bir sabitle carpilmasi uzereinden minimizasyon problemine direk dahil ettik. Boylece bu buyuklukler formulasyona dahil oldular ve azaltılma hedefinin bir parcasi haline geldiler. Yani hem e_{ui}^2 hem de hatayi olusturan degerlerin kendileri minimize edilecek.

Rasgele Gradyan Inisi (Stochastic Gradient Descent -SGD-)

Modeli nasıl minimize ederiz? Bu model konveks (convex) degil, ki konvekslik bilindigi gibi fonksiyonun duzgün bir cukur gibi oldugu problemlerdir. Boyle cukur fonksiyonlarında herhangi bir noktadan baslarsiniz, gradyani hesaplarsiniz, ve bu gradyan hep optimal inis noktasini (daha dogrusu tersini) gosterir, ve yolda giderken takilip kalabileceginiz yerel minimumlar mevcut degildir, ve sonunda cukur dibine ulasilir. Bizim problemimizde $q_i^T p_u$ var, bu degiskenlerin ikisi de bilinmiyor, ve bu carpimin karesi alindigi icin genel karesellligi (quadratic) kaybetmis oluyoruz. Fakat yine de SGD bu problemi cozebiliyor. Bunun sebeplerini, SGD SVD'nin hikayesiyle beraber yazinin sonunda bulabilirsiniz.

SGD icin gradyanlar lazim, her degisken icin minimizasyon toplami icindeki kismın (bu kisma E diyelim) ayrı ayrı kısmi turevini almak lazim. Mesela b_u icin

$$\frac{\partial E}{\partial b_u} = -2e_{ui} + 2\lambda b_u$$

Gradyan her zaman en yuksek cikisi gosterir, o zaman hesapsal algoritma onun tersi yonune gitmelidir. Bu gidisin adim buyuklugunu kontrol etmek icin disari-dan bizim belirledigimiz bir γ sabiti ile carpim yapabiliriz, ve bir numara daha, sabit 2 degerlerinin γ icinde eritilebilecegini farzederek onlari sileriz. Yani adim $\gamma(e_{ui} - \lambda b_u)$ haline geldi. Bir dongu icinde eski b_u bulunacak, gordugumuz yonde adim atilacak, yani adim onceki degere toplanacak, ve yeni deger elde edilecek. Diger degiskenler icin turev alip benzer islemleri yaparsak, sonuc soyle,

$$b_u \leftarrow b_u + \gamma(e_{ui} - \lambda \cdot b_u)$$

$$b_i \leftarrow b_i + \gamma(e_{ui} - \lambda \cdot b_i)$$

$$q_i \leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda \cdot q_i)$$

$$p_u \leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda \cdot p_u)$$

Her degisken icin baslangic noktasi rasgele olarak secilebilir, ki γ, λ sabitleri ile beraber bu baslangic noktaları icin en iyi degerler deneme/ yanilma ya da capraz saglama (crossvalidation) ile bulunabilir.

Rasgelelik, aynen *Lojistik Regresyon* orneginde oldugu gibi verinin rasgeliliginden geliyor, her veri noktasini teker teker sirayla isliyoruz aslinda fakat bu “siranin” rasgele oldugunu farzettigimiz icin ozyineli algoritmamiz rasgelelik elde ediyor. Python kodu altta, egitim icin kod sadece bir kere verinin uzerinden geciyor. Basa donup birkac kere (hatta yuzlerce) veriyi isleyenler de oldu.

```
from numpy.linalg import linalg as la
import numpy as np
import random
import pandas as pd, os

def create_training_test(df, collim=2, rowlim=200):
    test_data = []
    df_train = df.copy()
    for u in range(df.shape[0]):
        row = df.ix[u]; idxs = row.index[row.notnull()]
        if len(idxs) > collim:
            i = random.choice(idxs); val = df.ix[u,i]
            test_data.append([u,i,val])
            df_train.ix[u,i] = np.nan
        if len(test_data) > rowlim: break
    return df_train, test_data
```

```

def ssvd(df_train,rank):
    print 'rank',rank
    gamma = 0.02 # regularization
    lam = 0.05

    mu = df_train.mean().mean()
    m,n = df_train.shape
    c = 0.03
    b_u = np.ones(m) * c
    b_i = np.ones(n) * c
    p_u = np.ones((m, rank)) * c
    q_i = np.ones((rank, n)) * c
    r_ui = np.array(df_train)
    for u in range(m):
        row = df_train.ix[u]; idxs = row.index[row.notnull()]
        for i in idxs:
            i = int(i)
            r_ui_hat = mu + b_i[i] + b_u[u] + np.dot(q_i[:,i].T,p_u[u,:])
            e_ui = r_ui[u,i] - r_ui_hat
            b_u[u] = b_u[u] + gamma * (e_ui - lam*b_u[u])
            b_i[i] = b_i[i] + gamma * (e_ui - lam*b_i[i])
            q_i[:,i] = q_i[:,i] + gamma * (e_ui*p_u[u,:].T - lam*q_i[:,i])
            p_u[u,:] = p_u[u,:] + gamma * (e_ui*q_i[:,i].T - lam*p_u[u,:])
    return mu,b_u,b_i,q_i,p_u

```

Kodun önemli bir özelliği sudur, boş yani nan değeri içeren notlar eğitim sırasında atlanır. SGD seyrek verilerle de işleyebilen bir eğitim yöntemidir. Bu durumda verinin seyrekliği (sparsity) bizim için çok faydalı, çünkü o veri noktalarına bakılmayacak, `row.notnull()` ile boş olmayan öğelerin indis değerlerini alıyoruz.

Basit bir örnek

```

import pandas as pd
import ssvd
d = np.array(
[[ 5., 5., 3., nan, 5., 5.],
 [ 5., nan, 4., nan, 4., 4.],
 [ nan, 3., nan, 5., 4., 5.],
 [ 5., 4., 3., 3., 5., 5.],
 [ 5., 5., nan, nan, nan, 5.]
])
data = pd.DataFrame (d, columns=['0','1','2','3','4','5'],
                    index=['Ben','Tom','John','Fred','Bob'])
mu,b_u,b_i,q_i,p_u = ssvd.ssvd(data,rank=3)
print mu
print 'b_u',b_u
print 'b_i',b_i
print 'q_i',q_i
print 'p_u',p_u
u = 4; i = 2 # Bob için tahmin yapalım
r_ui_hat = mu + b_i[i] + b_u[u] + np.dot(q_i[:,i].T,p_u[u,:])
print r_ui_hat

```

```

rank 3
4.31388888889
b_u [ 0.05129388  0.01927226  0.0206893   0.0065487   0.06568321]
b_i [ 0.07820389  0.01958841 -0.03217881  0.01561187  0.04071886  0.07140383]
q_i [[ 0.03132989  0.02957741  0.02802317  0.02951804  0.0301854   0.03108419]
      [ 0.03132989  0.02957741  0.02802317  0.02951804  0.0301854   0.03108419]
      [ 0.03132989  0.02957741  0.02802317  0.02951804  0.0301854   0.03108419]]
p_u [[ 0.03053543  0.03053543  0.03053543]
      [ 0.0295772   0.0295772   0.0295772 ]
      [ 0.02963018  0.02963018  0.02963018]
      [ 0.02921864  0.02921864  0.02921864]
      [ 0.03100583  0.03100583  0.03100583]]
4.34999993855

```

Test Etmek

Test verisi olusturmak icin egitim verisinde rasgele olarak bazi notlari sectik, bunlari bir kenara kaydederek onlari ana matris icindeki degerini sildik (yerine nan koyarak), ve bir kısmi silinmiş yeni bir eğitim matrisi yarattık, `create_training_test` islevinde bu gorulebilir. Bu islevde her kullanicidan sadece bir tane not verisi aliyoruz, ve bunu sadece belli bir sayida, `collim` kadar, not vermiş kullanıcılar icin yapıyoruz, ki böylece az sayida not vermiş kullanıcıların verisini azaltmamış oluyoruz. Ayrıca belli miktarda, `rowlim` kadar test noktası elde edince iş bitti kabul ediyoruz. Test verisi yaratmak için %80-%20 gibi bir ayırım yapmadık, yani eğitim verisindeki tüm kullanıcıları ve onların neredeyse tüm verisini eğitim için kullanıyoruz.

Movielens verisine gelelim. *SVD, Toplu Tavsiye* yazısındaki `movielens_prep.py` ile gerekli eğitim dosyası üretildiğini farzederek,

```

import pandas as pd, os
df = pd.read_csv("%s/Downloads/movielens.csv" % os.environ['HOME'], sep=';')
print df.shape
df = df.ix[:,1:3700] # id kolonunu atla,
df.columns = range(3699) # kolon degerlerini tekrar indisle
print df.shape

(6040, 3731)
(6040, 3699)

```

Eğitim ve test verisi yaratıyoruz,

```

import ssvd
df_train, test_data = ssvd.create_training_test(df, rowlim=500, collim=300)
print len(test_data)

501

mu, b_u, b_i, q_i, p_u = ssvd.ssvd(df_train, rank=25)
print 'mu', mu

rank 25
mu 3.23835007474

```

Test

```
rmse = 0; n = 0
for u,i,real in test_data:
    r_ui_hat = mu + b_i[i] + b_u[u] + np.dot(q_i[:,i].T,p_u[u,:])
    rmse += (real-r_ui_hat)**2
    n += 1
print "rmse", np.sqrt(rmse / n)

rmse 0.903347878156
```

Sonuc oldukca iyi.

Formulasyonun Hikayesi

SGD SVD'nin hikayesi soyle. Yil 2009, Netflix Yarismasi [11] katilimcilarindan Simon Funk (gercek adi Brandyn Webb) SGD SVD yaklasimini kodlayip veri uzerinde isletince birden bire siralamada ilk 3'e firlar; Webb artik cok unlu olan blog yazisinda [1] yaklasimi detayıyla paylasip forum'da haberini verince bu haber tam bir bomba etkisi yaratir. Pek cok kisi yaklasimi kopyalar, hatta kazanan BellKor urununde Webb'in SVD yaklasiminin kullanildigi biliniyor.

Bu metotun kesfi hangi basamaklardan gecti? Beni meraklandiran minimizasyon formulasyonun konveks olmamasiydi – genellikle optimizasyon problemlerinde konveksligin mevdudiyeti aranir, cunku bu durumda sonuca yaklasmak (convergence) icin bir garanti elde edilir. Bu durumda konvekslik yoktu. Biraz arastir-inca Bottou ve LeCunn gibi arastirmacilarin yazilarina ulastik [4]. Onlara gore konvekslik olmamasi yapay ogrenim arastirmacilarini korkutmamali, eger say-isal (empirically) isleyen bir algoritma var ise, teorik ispat gelene kadar bu meto-tun kullanilmasinda sakınca yoktur.

Fakat boyle buluslarda yine de bazi garantiler temel alinmis olabilir, arastirmaci tamamen baliklama atlayis yapmaz. Webb'in kendisine bu sorulari sorduk ve bize bulusun hangi seviyelerden gectigini anlatti. Geriye sariyoruz, Webb Net-flix'den cok once yapay sinir aglarini arastirmaktadir, ve Sanger, Oja'nin [5,6] yayinlarini baz alarak kurdugu bir YSA icin bir cozum buldugunu farkedir. Say-isal cozumde ozdeger/vektor bulmaya yarayan Ustel Metotun (power method) bir seklini kullanmistir, ki Sanger'in Genel Hebbian Algoritmasinin (GHA) ustel metot ile baglantilari var, ve bu GHA yayininda "egitilince" ozdeger/vektor ve PCA hesabi yapabilen bir YSA'dan bahsediliyor. Daha onemlisi GHA 1 olasilikla (yani kesin) bu sonuclara erisebiliyor.

Daha sonra Webb bu cozumu arkadasi Gorrell ile tartisirken Gorrell ona problem formulasyonunun SVD olarak gorulebilecegini soyler. Bilindigi gibi ozdeger/vektor hesabi ile SVD yakin akraba sayilir. Ikili bu baglamda birkac yayin da yaparlar. Daha sonra Netflix yarismasi basladiginda Webb cozum icin gradyan baz alarak SGD kullanabilecegini farkediyor, ki SGD ile ustel metot arasinda teorik baglanti var [7]. Ve sonuc olarak SGD SVD metotu ortaya cikiyor.

Tabii ki "SGD SVD ne kadar SVD sayilir?" gibi bir soru sorulabilir. Evet, regular-izasyon bazi gayri-lineerlikleri probleme sokar, zaten bu cozumu "yaklasiksal"

yapan kisim da budur. Fakat belli sartlarda, regularizasyon olmasa cozum tam SVD olacaktır. Bu bulusun puf noktası bu bilgide, ve ustteki teorik benzerliklerde, onlari biliyor olmakta yatiyor. Eger bunlar biliniyor ise, ve saglam lineer cebir bilgisi ile gerektigi zaman onlari ne kadar esnetebilecegimizi biliriz. Konu hakkındaki daha fazla detay surada [10] bulunabilir.

Kaynaklar

- [1] <http://sifter.org/~simon/journal/20061211.html>
- [2] Koren, Bell, *Recommender Systems Handbook*, http://www.cs.bme.hu/nagyadat/Recommender_systems_handbook.pdf
- [3] <http://www2.research.att.com/~volinsky/papers/ieeecomputer.pdf>
- [4] <http://www.cs.nyu.edu/~yann/talks/lecun-20071207-nonconvex.pdf>
- [5] http://courses.cs.washington.edu/courses/cse528/09sp/sanger_pca_nn.pdf
- [6] <http://users.ics.aalto.fi/oja/Oja1982.pdf>
- [7] <http://arxiv.org/pdf/1308.3509>
- [8] http://www.maths.qmul.ac.uk/~wj/MTH5110/notes/MAS235_lecturenotes1.pdf
- [9] <http://heim.ifi.uio.no/~tom/powerandqrslides.pdf>
- [10] <http://math.stackexchange.com/questions/649701/gradient-descent-on-non-convex-function-works-but-how>
- [11] Netflix Odulu, <http://www.netflixprize.com>