

## Regresyon, Ridge, Lasso, Capraz Saglama, Regularize Etmek

Konumuz regresyon cesitleri, ve ornek veri olarak diyabet hastaligi olan kisilerden alinmis bazi temel verilerle hastaligin bir sene sonraki ilerleme miktarini kullanilacak. Regresyon sayesinde temel veriler ile hastaligin ilerlemesi arasinda bir baglanti bulunabilir, bu sayede hem veri aciklanir / daha iyi anlasilir (hangi degisken onemlidir, hangisi degildir), hem de baska bir hastanin temel verilerini kullanarak o hastanin diyabetinin bir sene sonra ne olacagini tahmin etmek mumkun olur. Kullanilan temel veriler kisinin yasi, cinsiyeti, vucut kutle endeksi (body mass index) ortalama tansiyonu ve alti kere alinmis kan serum olcumleridir.

```
from pandas import *
diabetes = read_csv("diabetes.csv",sep=',')
diabetes_y = diabetes['response']
diabetes_x = diabetes.drop("response",axis=1)
diabetes_x_train = diabetes_x[:-20]
diabetes_x_test = diabetes_x[-20:]
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]
```

Ilk once basit regresyonu hatirlayalim (ordinary least regression). Bu teknigi daha once pek cok yonden gorduk. *Lineer Cebir*, *Cok Degiskenli Calculus* ders notlarinda, ve bizim *Uygulamali Matematik* yazilarinin hepsinde bu teknigin turetilmesi mevcut. Formül

$$\hat{w} = (X^T X)^{-1} X^T y$$

Numerik olarak hemen bu hesabi yapabiliriz. Bir hatirlatma: veri setine y ekseninin nerede kesildiginin bulunabilmesi icin suni bir ekstra 'intercept' adli kolon ekleyecegiz, bu kolon iki boyutta  $y = ax + c$  formundeki  $c$ 'nin bulunabilmesi icindir. Pandas ile bu ekstra kolonu eklemek cok basit, ismen mevcut olmayan kolon erisildigi anda o kolon hemen yoktan yaratilir.

```
import numpy.linalg as la
x_tmp = diabetes_x_train.copy()
x_tmp['intercept'] = 1
xTx = np.dot(x_tmp.T,x_tmp )
ws = np.dot(la.inv(xTx),np.dot(x_tmp.T,diabetes_y_train))
ws
```

```
array([ 3.03499452e-01, -2.37639315e+02,  5.10530605e+02,
        3.27736981e+02, -8.14131711e+02,  4.92814589e+02,
        1.02848453e+02,  1.84606489e+02,  7.43519617e+02,
        7.60951724e+01,  1.52764307e+02])
```

Ayni hesabi bir de scikit-learn paketini kullanarak yapalim. Bu paketin LinearRegression cagrisi intercept isini otomatik olarak hallediyor, eger intercept olmasin isteseydik, fit\_intercept=False diyecektik.

```

from sklearn import linear_model, cross_validation
lin = linear_model.LinearRegression()
lin.fit(diabetes_x_train, diabetes_y_train)
print lin.coef_
print "score", lin.score(diabetes_x_test, diabetes_y_test),

```

```

[ 3.03499452e-01 -2.37639315e+02  5.10530605e+02  3.27736981e+02
 -8.14131711e+02  4.92814589e+02  1.02848453e+02  1.84606489e+02
  7.43519617e+02  7.60951724e+01]
score 0.585075302278

```

Sonular birbirine oldukça yakın. Şimdi diğer tekniklere geelim.

Sirt Regresyonu (Ridge Regression)

Klasik regresyon ile

$$\hat{w} = \arg \min_w \|y - Xw\|^2$$

problemini çözduğümüzü biliyoruz, ki  $\|\cdot\|^2$  Oklit normunun karesini temsil ediyor. Fakat bazı durumlarda  $X^T X$ 'in tekil (singular) olması mümkün ki böyle bir durumda  $(X^T X)^{-1}$ 'in tersini almamız mümkün olmazdı. Tekillik ne zaman ortaya çıkar? Eğer elimizde veri noktasından daha fazla boyut var ise mesela... Diyelim ki veri olarak 10 tane kolon var, ama sadece 9 tane veri satırı. Sirt Regresyonunun çıkış noktası budur.

Fakat ek olarak bu teknik kestirme hesaplarımıza (estimation) bir meyil / yanlılık (bias) eklemek için de kullanılabilir ve bu meyil tahminlerin / kestirme hesapların iyileşmesine faydalı olabilir.

Meyili nasıl ekleriz? Diyelim ki bizim tanımlayacağımız bir  $\lambda$  ile tüm  $w$ 'lerin toplamına bir üst sınır tanımlayabiliriz. Böylelikle regresyonun bulacağı katsayıların çok fazla büyümesine bir "ceza" getirmiş olacağız, ve bu cezayı içeren regresyon hesabı o cezadan kaçınmak için mecburen bulacağı katsayıları ufak tutacak, hatta bazılarını sıfıra indirebilecek. Bu azaltmaya istatistikte küçülme (shrinkage) ismi veriliyor.

Sirt regresyonu için bu küçültme şöyle

$$\hat{w}_{sirt} = \arg \min_w (\|y - Xw\|^2 + \lambda \|w\|^2)$$

Görüldüğü üzere  $w$ 'nin büyüklüğünü, bir  $\lambda$  katsayısı üzerinden minimizasyon problemine dahil ettik, böylece diğer parametreler ile büyüklük de minimize edilecek. Üstteki tanım sınırı tanımlanmamış (unconstrained) bir optimizasyon problemidir. Sınırlı olarak

$$\min_w \|y - Xw\|^2$$

*Su kusura göre (subject to)  $\|w\|^2 \leq \tau$*

ki  $\lambda$  Lagrange carpanıdır. Aslında şimdiye kadar üstteki çevrimin tersini gördük çoğunlukla (yani sınırlı problemden sınırsıza gitmeyi), bu gidis tarzını görmek de iyi oldu.

Neyse bastaki sinirsiz problemi cozmek icin ifadenin gradyanini alalim,

$$\nabla(\|y - Xw\|^2 + \lambda\|w\|^2)$$

$$\nabla((y - Xw)^T(y - Xw) + \lambda w^T w)$$

$$\nabla((y^T - w^T X^T)(y - Xw) + \lambda w^T w)$$

$$\nabla(y^T y - y^T Xw - w^T X^T y + w^T X^T Xw + \lambda w^T w)$$

$$-y^T X - X^T y + 2X^T Xw + 2\lambda w$$

$$-2X^T y + 2X^T Xw + 2\lambda w$$

$$2X^T Xw + 2\lambda w - 2X^T y$$

$$2(X^T X + \lambda I)w - 2X^T y$$

Minimizasyon icin ustteki ifadeyi sifira esitleyebiliriz

$$2(X^T X + \lambda I)w - 2X^T y = 0$$

O zaman

$$(X^T X + \lambda I)w = X^T y$$

$$\hat{w} = (X^T X + \lambda I)^{-1} X^T y$$

Bu son ifade en az kareler (least squares) yani normal regresyon cozum formulune cok benziyor, sadece ek olarak bir  $\lambda I$  toplama islemi var. Demek ki sirt regresyonunu kullanmak icin zaten yaptigimiz hesaba, zaten bizim kendimizin karar verdigi bir  $\lambda$  uzerinden  $\lambda I$  eklersek, geri kalan tum islemler ayni olacak.

Kontrol edelim

```
lam = 0.2
wridge = np.dot(la.inv(xTx+lam*np.eye(xTx.shape[0])),\
                 np.dot(x_tmp.T,diabetes_y_train))
wridge

array([ 16.70807829, -179.42288145,  447.64999897,  285.41866481,
        -51.7991733 , -75.09876191, -192.46341288,  123.61066573,
        387.91385823,  105.53294479,  152.7637018 ])
```

Simdi scikit-learn ile ayni hesabi yapalim

```
ridge = linear_model.Ridge(alpha=0.2)
ridge.fit(diabetes_x_train, diabetes_y_train)
print ridge.score(diabetes_x_test, diabetes_y_test), ridge.coef_

0.553680030106 [ 16.69330211 -179.414259  447.63706059  285.40960442 -51.79094255
 -75.08327488 -192.45037659  123.60400024  387.91106403  105.55514774]
```

Bir yontem daha var, bu yonteme Lasso ismi veriliyor. Lasso’ya gore cezalandirma

$$\sum_{k=1}^n w_k^2 \leq \lambda$$

uzerinden olur. Bu yontemin tum detaylarına simdilik inmeyecegiz.

Ornek olarak bir  $\lambda$  ile onun buldugu katsayılara bakalim.

```
lasso = linear_model.Lasso(alpha=0.3)
lasso.fit(diabetes_x_train, diabetes_y_train)
lasso.coef_

array([  0.          , -0.          , 497.3407568 , 199.17441037,
        -0.          , -0.          , -118.89291549,  0.          ,
        430.93795945,  0.          ])
```

Lasso bazi katsayilari sifra indirdi! Bu katsayilarin agirlik verdigi degiskenleri, eger Lasso’ya inanir-sak, modelden tamamen atmak mumkundur.

Bu arada Sirt ve Lasso yontemlerinin metotlarına “regularize etmek (regularization)” ismi de veriliyor.

k-Katlamali Capraz Saglama (k-fold Cross-Validation)

Bir yapay ogrenim algoritmasini kullanmadan once veriyi iki parcaya ayirmak ise yarar; bu parcalar tipik olarak egitim verisi (training set) digeri ise test verisi (validation set) olarak isimlendirilir. Isimlerden belli olacagi uzere, algoritma egitim seti uzerinde egilir; ve basarisi test verisi uzerinden rapor edilir. Bir bakima modelin olusturulmasi bir set uzerindendir, sonra “al simdi hic gormedigin bir veri seti, bakalim ne yapacaksin” sorusunun cevabi, saglamasi bu sekilde yapilir.

k-Katlamali Capraz Saglama bu iki parcali egitim / test kavramini bir adim oteye tasir. Ufak bir k seceriz, ki bu genellikle 5 ila 10 arasinda bir sayi olur, ve tum verimizi rasgele bir sekilde ama k tane ve esit buyuklukte olacak sekilde parcalara ayiririz. Bu parcalara “katlar (folds)” ismi verilir bazen (ki isim buradan geliyor). Sonra teker teker her parçayı test verisi yapariz ve geri kalan tum parcalari egitim verisi olarak kullaniriz. Bu islemi tum parcalar icin tekrarlariz.

Bu yaklasim niye faydalidir? Cunku veriyi rasgele sekillerde bolup, pek cok yonden egitim / test icin kullaninca verinin herhangi bir sekilde bizi yonlendirmesi / aldatmasi daha az mumkun hale gelir.

Ve iste bu ozelligi, ek olarak, capraz saglamayi “model secmek” icin vazgecilmez bir arac haline getirir.

Model secmek nedir? Model secimi ustteki baglamda optimal bir  $\lambda$  bulmaktir mesela, yani her modeli temsil eden bir  $\lambda$  var ise, en iyi  $\lambda$ ’yi bulmak, en iyi modeli bulmak anlamina geliyor, capraz saglama bunu sagliyor. Capraz saglama icin scikit-learn’un sagladigi fonksiyonlar vardır, once katlari tanimlariz, sonra bu degistirilmis regresyon fonksiyonlarına katlama usulunu geceriz.

```
k_fold = cross_validation.KFold(n=420, n_folds=7)
```

Katlari ustteki gibi tanımladık. 420 tane veri noktasını 7 kata böldük. Şimdi bu katları kullanalım,

```
ridge_cv = linear_model.RidgeCV(cv=k_fold)
ridge_cv.fit(np.array(diabetes_x), np.array(diabetes_y))
ridge_cv.alpha_
```

```
0.10000000000000001
```

Üstteki sonuç  $\lambda = 0.1$ 'i gösteriyor. Bu  $\lambda$  daha optimalmiş demek ki. Lasso için benzer şekilde

```
lasso_cv = linear_model.LassoCV(cv=k_fold)
lasso_cv.fit(diabetes_x, diabetes_y)
```

```
LassoCV(alphas=None, copy_X=True,
        cv=sklearn.cross_validation.KFold(n=420, n_folds=7), eps=0.001,
        fit_intercept=True, max_iter=1000, n_alphas=100, normalize=False,
        precompute='auto', tol=0.0001, verbose=False)
```

```
lasso_cv.alpha_
```

```
0.0028395871911771854
```

```
lasso_cv.score(diabetes_x_test, diabetes_y_test)
```

```
0.59709035088351892
```

Şimdi veri setinin bir kısmı üzerinde teker teker hangi algoritmanın daha başarılı olduğunu görelim.

```
def predict(row):
    j = row; i = row-1
    new_data = diabetes_x[i:j]
    print diabetes_y[i:j], "lasso", lasso_cv.predict(new_data), \
          "ridge", ridge_cv.predict(new_data), \
          "linear", lin.predict(new_data)

predict(-2) # sondan ikinci veri satiri
predict(-3)
predict(-4)
predict(-5)
predict(-8)
```

```
439      132
Name: response lasso [ 122.23614365] ridge [ 127.1821212] linear [ 123.56604986]
```

```
438      104
Name: response lasso [ 101.85154941] ridge [ 108.89678818] linear [ 102.5713971]
437      178
Name: response lasso [ 192.95669774] ridge [ 189.58095011] linear [ 194.03798086]
436       48
Name: response lasso [ 52.89039744] ridge [ 57.66611598] linear [ 52.5445869]
433       72
Name: response lasso [ 60.42852011] ridge [ 66.3661042] linear [ 61.19831285]
```

Ustteki sonuclara gore gercek degeri 132 olan 439. satirda lasso 122.2, sirt (ridge) 127.1, basit regresyon ise 123.5 bulmus. O veri noktası için sirt yöntemi daha başarılı çıktı.

Sonucılara bakınca bazen sirt, bazen normal regresyon başarılı çıkıyor. Hangi yöntem kazanmış o zaman? Bir o, o bir bu öndeyse, hangi yöntemi kullanacağımızı nasıl bileceğiz?

Aslında her seferinde tek bir metodu kullanmak gerekmiyor. Bu metodları bir takım (ensemble) halinde işletebiliriz. Her test noktasını, her seferinde tüm metotlara sorarız, gelen sonuçların mesela.. ortalamasını alırız. Bu şekilde tek başına işleyen tüm metotlardan tutarlı olarak her seferinde daha iyi sonuca ulaşacak bir sonuç elde edebiliriz. Zaten Kaggle gibi yarışmalarda çoğunlukla birinciliği kazanan metodlar bu türden takım yöntemlerini kullanan metodlar, mesela Netflix yarışmasını kNN ve SVD metodlarını takım halinde işleten bir grup kazandı.

Kaynaklar

<http://www.lx.it.pt/~mtf/Figueiredo.Linear.Regression.pdf>

[http://www.cs.nyu.edu/~mohri/mls/lecture\\_8.pdf](http://www.cs.nyu.edu/~mohri/mls/lecture_8.pdf)

Harrington, P., *Machine Learning in Action*

Shalizi, C., *Data Analysis from an Elementary Point of View*