

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 kitle 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 kesildigini 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))
print ws
```

```
[ 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 gelelim.

Sirt Regresyonu (Ridge Regression)

Klasik regresyon ile

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

problemimizi 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 λ 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 sifira 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 λ katsayısı üzerinden minimizasyon problemine dahil ettik, böylece diğer parametreler ile büyüklük te minimize edilecek. Üstteki tanım sınırı tanımlanmamış (unconstrained) bir optimizasyon problemidir. Sınırlı olarak

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

Su kosula gore (subject to) $\|w\|^2 \leq \tau$

ki λ Lagrange carpanidir. Aslinda simdiye kadar ustteki cevrimin tersini gorduk cogunlukla (yani sinirli problemnden sinirsiza gitmeyi), bu gidis tarzini gormek te 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 λI toplama islemi var. Demek ki sirt regresyonunu

kullanmak için zaten yaptığımız hesaba, zaten bizim kendimizin karar verdiği bir λ üzerinden λI eklersek, geri kalan tüm işlemler aynı 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))
print wridge
```

```
[ 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 aynı hesabi yapalım

```
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 -51.790942
 -75.08327488 -192.45037659  123.60400024  387.91106403  105.55514774]
```

Bir yöntem daha var, bu yonteme Lasso ismi veriliyor. Lasso'ya göre cezalandırma

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

üzerinden olur. Bu yöntemin tüm detaylarına şimdiye değin değinmiştik.

Örnek olarak bir λ ile onun bulduğu katsayıları bakalım.

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

```
[ 0.          -0.          497.3407568   199.17441037  -0.          -0.
 -118.89291549   0.          430.93795945   0.          ]
```

Lasso bazı katsayıları sifıra indirdi! Bu katsayıların ağırlık verdiği değişkenleri, eğer Lasso'ya inanırsak, modelden tamamen atmak mümkündür.

Bu arada Sirt ve Lasso yöntemlerinin metodlarına "regularize etmek (regularization)" ismi de veriliyor.

k-Katlamalı Capraz Sağlama (k-fold Cross-Validation)

Bir yapay öğrenim algoritmasını kullanmadan önce veriyi iki parçaya ayırmak işe yarar; bu parçalar tipik olarak eğitim verisi (training set) diğeri ise test verisi (validation set) olarak isimlendirilir. İsimlerden belli olacağı üzere, algoritma eğitim

seti üzerinde eğitilir; ve başarısı test verisi üzerinden rapor edilir. Bir bakıma modelin oluşturulması bir set üzerindedir, sonra "al şimdi hiç görmediğin bir veri seti, bakalım ne yapacaksın" sorusunun cevabı, sağlaması bu şekilde yapılır.

k-Katlamalı Capraz Sağlama bu iki parçalı eğitim / test kavramını bir adım öteye taşır. Ufak bir k seçeriz, ki bu genellikle 5 ila 10 arasında bir sayı olur, ve tüm verimizi rasgele bir şekilde ama k tane ve esit büyüklükte olacak şekilde parçalara ayırırız. Bu parçalara "katlar (folds)" ismi verilir bazen (ki isim buradan geliyor). Sonra teker teker her parçayı test verisi yaparız ve geri kalan tüm parçaları eğitim verisi olarak kullanırız. Bu işlemi tüm parçalar için tekrarlarız.

Bu yaklaşım niye faydalıdır? Çünkü veriyi rasgele şekillerde bölüp, pek çok yondan eğitim / test için kullanınca verinin herhangi bir şekilde bizi yönlendirmesi / aldatması daha az mümkün hale gelir.

Ve iste bu özelliği, ek olarak, capraz sağlamayı "model seçmek" için vazgeçilmez bir araç haline getirir.

Model seçmek nedir? Model seçimi üstteki bağlamda optimal bir λ bulmaktır mesela, yani her modeli temsil eden bir λ var ise, en iyi λ 'yi bulmak, en iyi modeli bulmak anlamına geliyor, capraz sağlama bunu sağlıyor. Capraz sağlama için `scikit-learn`'un sağladığı fonksiyonlar vardır, önce katları tanımlarız, sonra bu değiştirilmiş regresyon fonksiyonlarına katlama usulünü geçeriz.

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

Katları üstteki gibi tanımladık. 420 tane veri noktasını 7 kata böl dedik. Şimdi bu katları kullanalım,

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

0.1

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

```
lasso_cv = linear_model.LassoCV(cv=k_fold)
print 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)
```

```
print lasso_cv.alpha_
```

0.00283958719118

```
print lasso_cv.score(diabetes_x_test, diabetes_y_test)
0.597090337358
```

Simdi veri setinin bir kısmi uzerinde teker teker hangi algoritmanin daha basarili oldugunu gorelim.

```
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, dtype: int64 lasso [ 122.2361344] ridge [ 127.1821212] linear [ 123.5]
```

```
438    104
```

```
Name: response, dtype: int64 lasso [ 101.85154189] ridge [ 108.89678818] linear [ 104.0]
```

```
437    178
```

```
Name: response, dtype: int64 lasso [ 192.95670241] ridge [ 189.58095011] linear [ 178.0]
```

```
436     48
```

```
Name: response, dtype: int64 lasso [ 52.8903924] ridge [ 57.66611598] linear [ 52.5]
```

```
433     72
```

```
Name: response, dtype: int64 lasso [ 60.42852107] ridge [ 66.3661042] linear [ 61.1]
```

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 yontemi daha basarili cikti.

Sonuclara bakınca bazen sirt, bazen normal regresyon basarili cikiyor. Hangi yontem kazanmis o zaman? Bir o, o bir bu ondeyse, hangi yontemi kullanacagimizi nasıl bilecegiz?

Aslında her seferinde tek bir metodu kullanmak gerekmiyor. Bu metodları bir takım (ensemble) halinde isletebiliriz. Her test noktasını, her seferinde tüm metotlara sorarız, gelen sonucların mesela.. ortalamasını alırız. Bu şekilde tek basına isleyen tüm metotlardan tutarlı olarak her seferinde daha iyi sonuca ulaşacak bir sonuc elde edebiliriz. Zaten Kaggle gibi yarışmalarda çoğunlukla birinciliği kazanan metotlar bu türden takım yöntemlerini kullanan metotlar, mesela Netflix yarışmasını kNN ve SVD metotlarını takım halinde isleten bir grup kazandı.

Kaynaklar

www.lx.it.pt/~mtf/Figueiredo_Linear_Regression.pdf

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