Destek Vektor Makinalari (Support Vector Machines)

En basit halleriyle SVM'ler risk minimize eden lineer siniflayicisidirlar.

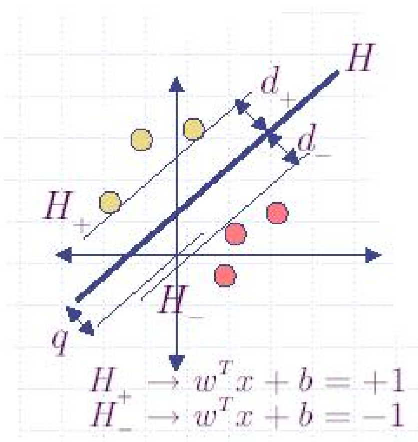
$$R(\Theta) \leq J(\Theta) = R_{emp}(\Theta) + \sqrt{\frac{h \times (log(\frac{2N}{h}) + 1) - log(\frac{\eta}{4})}{N}}$$

h: siniflayicinin kapasitesi

N: egitim verisinde kac veri noktasi oldugu

Vapnik ve Chernovenkis $1-\eta$ olasilikla ispaladi ki ustteki denklem dogrudur. SVM algoritmasi hem h degerini hem de sayisal, olcumsel riski ayni anda minimize etmektedir, ve bunu sinir noktalarini noktalarini ayirmakla yapmaktadir.

Turetelim



Karar duzlemi: $w^T x + b = 0$

Soyle bir tanim yapalim: $q = min_x \|x - 0\|$

- q, H^+ ve H^- formullerini ileride kullanacagiz.
- H icin: $q = min_x \|x - 0\|$ su sarta tabi $w^T x + b = 0$
- Lagrange: $min_x \frac{1}{2} \|x - 0\|^2 + \lambda(w^T x + b)$
- Gradyani alalim ($\frac{\partial}{\partial x}$) ve 0 degerine esitleyelim
- Biraz cebirsel numaradan sonra: $q = \frac{|b|}{||w||}$

- Tanim:
  - $H^+ = w^T x + b = +1$
  - $H^- = w^T x + b = -1$
- Bu tanimi genellikte bir kayip olmadan yapabiliyoruz; b & w degerlerini hala duzeltebiliriz.

- q^+ ve q^- degerlerinin hesapla
  - $q^+ = \frac{|b-1|}{||w||}$
  - $q^- = \frac{|-b-1|}{||w||}$
- Ayrac o zaman soyle
  - $m = q^+ + q^- = \frac{|b-1-b-1|}{||w||} = \frac{|-2|}{||w||} = \frac{2}{||w||}$

Ayraclarin olabildigince ayirmasini istiyorsak m'i arttiriz (yani $\frac{2}{||w||}$'i maksimize ederiz), ya da $||w||$ degerini minimize ederiz.

Sinirlar

Veri noktalarini oyle siniflamak istiyoruz ki + ve - noktalar hiperduzlemlerin dogru noktalarinda kalsinlar.

$w^T x + b \geq +1, \forall y_i = +1$

$w^T x + b \leq -1, \forall y_i = -1$

Bu iki denklemi birlestirelim

$y_i(w^T x + b) - 1 \geq 0$

Her seyi biraraya koyalim

$min \frac{1}{2} ||w||^2$ subject to $y_i(w^T x_i + b) - 1 \geq 0$

Bu form tanidik geliyor mu? Bu qp ile cozulebilecek karesel (quadratic) bir formul, programdir!

qp

- Python dilinde cvxopt paketi vardir
- Matlab Optimization Toolbox'da qp() var.
- Steve Gunn'in SVM Toolbox'i icinde C ile yazilmis bir qp var
- SVMLight icinde ayrica bir qp var
- qp fonksiyonlari problemleri genelde $\frac{1}{2}x^TPx + q^Tx$ formunda gormek isterler.
- Biraz once elde ettigimiz denklemi bu istenen formata dogru "masajlayabiliriz"

Ikiz (dual)

- SVM ihtiyaclari icin ikiz formul (dual) ile calismak daha rahattir
- Lagrange (tekrar) olusturalim, turevi alalim, ve sifira esitleyelim.
- Bunun sonucunda elimize KKT noktalari gececektir

$$L_p = \frac{1}{2}||w||^2 - \sum_i \alpha_i(y_i(w^Tx_i + b) - 1) \tag{1}$$

$$\frac{\partial}{\partial w}L_p = w - \sum_i \alpha_i y_i x_i = 0$$

$$w = \sum_i \alpha_i y_i x_i \tag{2}$$

$$\frac{\partial}{\partial b}L_p = -\sum_i \alpha_i y_i = 0 \tag{3}$$

2 ve 3 denklemini asal (primal) denklem olan 1 icine koydugumuz zaman

$$\text{Maksimize et } L_D = \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j x_i^T x_j \tag{4}$$

sinirlar

$$\sum_i \alpha_i y_i = 0$$

$$\alpha_i \geq 0$$

qp

- Bu yine qp() formunda bir problem! Sadece bu sefer cozecegimiz degiskenler α_i'lar, x'lar degil.
- Denklem 4 su forma $\frac{1}{2}x^T P x + q^T x$ masajlanabilir
- Bunun yapmak icin $P_{i,j}$'ye $-y_i y_j x_i^T x_j$ degerini atariz.
- Ve qp'yi cagiririz
- Sonuc bir α'lar listesi olacaktir.

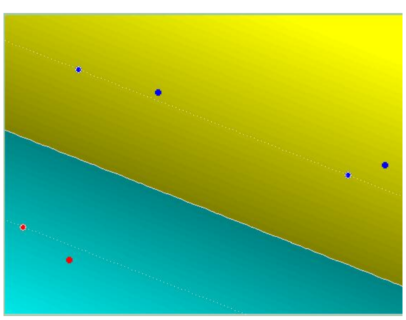
b degerini hesaplamak

- KKT kosulunun sebebiyle sifir olmayan her α_i icin ana problemde ona tekabul eden kisitlayici sart sıkıdır (tight), yani bir esitliktir.
- O zaman sifir olmayan her α_i icin b'yi $w^T x_i + b = y_i$ ifadesini kullanarak hesaplariz.
- Sifir olmayan her α_i'dan gelen b yaklasik olarak diger other b'lere esit olacaktir. Final b'yi hesaplamak icin tum b'lerin ortalamasini almak numerik olarak daha garantidir.

Siniflayici Tamamlandi

Her yeni x noktasi icin artik $sign(x^T w + b)$ ibaresini siniflayicimiz olarak kullanabiliriz. -1 ya da $+1$ olarak geri gelecek sonuc bize yeni noktanin hangi sinifa ait oldugunu soyleyecektir.

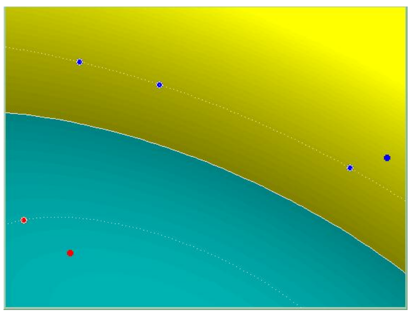
Ornek Çıktı



Kernels

- Simdiye kadar lineer ayraclardan bahsettik.
- SVM'ler lineer olmayan ayraclarla da calisabilir.
- Cok basit: Bir temel fonksiyon kullanarak girdiyi daha yuksek bir boyuta dogru bir onislemden gecirirsek bunu basarabiliriz.
- Algoritmanin geri kalani degismeden kalacaktir.

Gayri Lineer Cekirdek



Esneme Payı

- Bazen bir problem ayrilmaya musait olmayabilir.
- Cok uc noktalardaki bazi noktalar siniflayicinin calismasini imkansiz hale getirebilir
- Bunun cozumu icin siniflayiciya "esneme payı" dahil edebiliriz.
- Mesela $y_i = +1$ icin verinin yanlis tarafa dusmesini su durumda izin verebiliriz: $w^T + b \geq -0.03$
- Fakat eklemek gerekir ki bu tur noktalarin "cok fazla" olmasini da istemiyoruz, bu sebeple bu "yanlis" noktalarin sayisina da bir ceza getirebiliriz.

```
import numpy as np
from numpy import linalg
import cvxopt
import cvxopt.solvers

def svm(X, y):
    n_samples, n_features = X.shape

    # Gram matrix
    K = np.zeros((n_samples, n_samples))
    for i in range(n_samples):
        for j in range(n_samples):
            K[i,j] = np.dot(X[i], X[j])

    P = cvxopt.matrix(np.outer(y,y) * K)
    q = cvxopt.matrix(np.ones(n_samples) * -1)
    A = cvxopt.matrix(y, (1,n_samples))
    b = cvxopt.matrix(0.0)

    G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
    h = cvxopt.matrix(np.zeros(n_samples))

    # solve QP problem
    solution = cvxopt.solvers.qp(P, q, G, h, A, b)

    print solution

    # Lagrange multipliers
    a = np.ravel(solution['x'])

    print "a", a

    # Support vectors have non zero lagrange multipliers
    ssv = a > 1e-5
    ind = np.arange(len(a))[ssv]
    a = a[ssv]
    sv = X[ssv]
    sv_y = y[ssv]
```

```
    print "%d support vectors out of %d points" % (len(a), n_samples)
    print "sv", sv
    print "sv_y", sv_y

    # Intercept
    b = 0
    for n in range(len(a)):
        b += sv_y[n]
        b -= np.sum(a * sv_y * K[ind[n],ssv])
    b /= len(a)

    # Weight vector
    w = np.zeros(n_features)
    for n in range(len(a)):
        w += a[n] * sv_y[n] * sv[n]

    print "a", a
    return w, b, sv_y, sv, a

if __name__ == "__main__":

    def test():
        X = np.array([[3.,3.],[4.,4.],[7.,7.],[8.,8.]])
        y = np.array([1.,1.,-1.,-1.])
        w, b, sv_y, sv, a = svm(X, y)
        print "w", w
        print "b", b
        print 'test points'
        print np.dot([2.,2.], w) + b # > 1
        print np.dot([9.,9.], w) + b # < -1

    test()
```

Not: Ikizdeki L_d'yi maksimize ediyoruz, fakat hala `qp()`'deki minimize ediciyi cagiriyoruz. Bu sebeple tum α'larin toplamini temsil eden q'larin negatifini aliyoruz, `np.ones(n_samples) *-1` isleminde goruldugu gibi. Formuldeki karesel kisim icinde zaten $-\frac{1}{2}$ negatif ibaresi var, boylece geri kalan formulun degismesine gerek yok.

Kaynaklar

http://www.mblondel.org/journal/2010/09/19/support-vector-machines-in-python

Jebara, T., Machine Learning Lecture, Columbia University