

Destek Vektor Makinalari (Support Vector Machines)

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

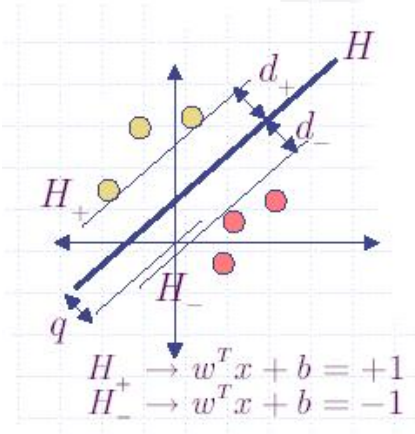
$$R(\Theta) \leq J(\Theta) = R_{emp}(\Theta) + \sqrt{\frac{h \times (\log(\frac{2N}{h}) + 1) - \log(\frac{\eta}{4})}{N}}$$

$h$ : siniflayicinin kapasitesi

$N$ : egitim verisinde kac veri noktası olduğu

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

Turetelim



Karar düzlemi:  $w^T x + b = 0$

Soyle bir tanım yapalım:  $q = \min_x \|x - 0\|$

- $q$ ,  $H^+$  ve  $H^-$  formüllerini ilerde kullanacağız.
- $H$  için:  $q = \min_x \|x - 0\|$  su sarta tabi  $w^T x + b = 0$
- Lagrange:  $\min_x \frac{1}{2} \|x - 0\|^2 + \lambda(w^T x + b)$
- Gradyani alalım  $(\frac{\partial}{\partial x})$  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 hiperdüzlemlerin 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 denklemleri birlestirelim

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

Her sey biraraya koyalim

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

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

qp

- Python dilinde cvxopt paketi vardır
- 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  $\frac{1}{2}x^T Px + q^T x$  formunda gormek isterler.
- Biraz once elde ettigimiz denklemi bu istenen formata dogru “masajlayabiliriz”

Ikiz (dual)

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

$$L_p = \frac{1}{2}||w||^2 - \sum_i \alpha_i (y_i (w^T x_i + b) - 1) \quad (1) \quad (\text{eq:primal})$$

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

$$w = \sum_i \alpha_i y_i x_i \quad (2) \quad (\text{eq:wdual})$$

$$\frac{\partial}{\partial b} L_p = - \sum_i \alpha_i y_i = 0 \quad (3) \quad (\text{eq:cdual})$$

~~eq:wdual~~

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

$$\text{Maksimize et } L_D = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j \quad (4) \quad (\text{eq:svm})$$

sinirlar

$$\sum_i \alpha_i y_i = 0$$

$$\alpha_i \geq 0$$

qp

- Bu yine qp() formunda bir problem! Sadece bu sefer cozecegimiz degiskenler  $\alpha_i$ 'lar,  $x$ 'lar degil.
- Denklem <sup>eq:svm</sup>  $\frac{1}{2}x^T Px + q^T x$  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  $\alpha$ 'lar listesi olacaktır.

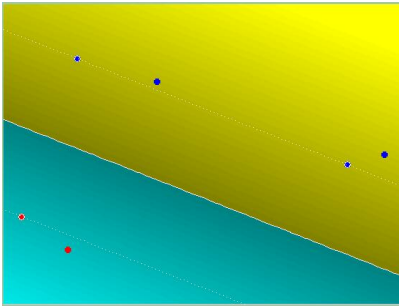
$b$  degerini hesaplamak

- KKT kosulunun sebebiyle sifir olmayan her  $\alpha_i$  icin ana problemde ona tekabul eden kisitlayici sart sikidir (tight), yani bir esitliktir.
- O zaman sifir olmayan her  $\alpha_i$  icin  $b$ 'yi  $w^T x_i + b = y_i$  ifadesini kullanarak hesaplariz.
- Sifir olmayan her  $\alpha_i$ 'dan gelen  $b$  yaklasik olarak diger other  $b$ 'lere esit olacaktır. 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.

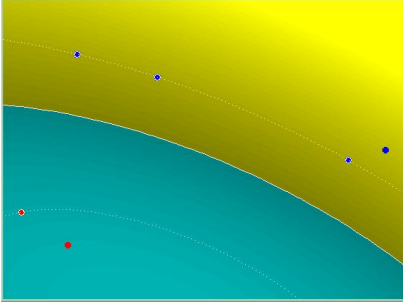
Ornek Çıktı



## 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 onisleminden gecirirsek bunu basarabiliriz.
- Algoritmanin geri kalani degismeden kalacaktır.

## Gayri Lineer Cekirdek



## Esneme Payı

- Bazen bir problem ayrilmaya musait olmayabilir.
- Cok uc noktalardeki 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” olmasını da istemiyoruz, bu sebeple bu “yanlis” noktalarin sayısına 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: İkizdeki  $L_d$ 'yi maksimize ediyoruz, fakat hala `qp()`'deki minimize ediciyi çağırıyoruz. Bu sebeple tüm  $\alpha$ 'ların toplamını temsil eden  $q$ 'ların negatifini alıyoruz, `np.ones(n_samples) * -1` işleminde görüldüğü gibi. Formüldeki karesel kısım içinde zaten  $-\frac{1}{2}$  negatif ibaresi var, böylece geri kalan formülün degismesine gerek yok.

Kaynaklar

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

Jebara, T., Machine Learning Lecture, Columbia University