

Perceptron tanulás

Líneárisan szeparálható osztályok esetén a perceptron tanulás véges iteráció után megáll:

Adott: T a tanulóponatok attribútumhalmaza (d dimenziós) és osztályváltozója

minden $t \in T$ vektort kiegészítjük egy $d+1$ -edik értékkel (mindig 1)

Legyen w : d dimenziós és $w=(0,0,...,0)$

while van helytelenül klasszifikált eleme T -nek **do**

for all minden $t \in T$ **do**

if t rosszul klasszifikált **do**

if t az első osztályba tartozik **do**

$w=w+t$

else

$w=w-t$

end if

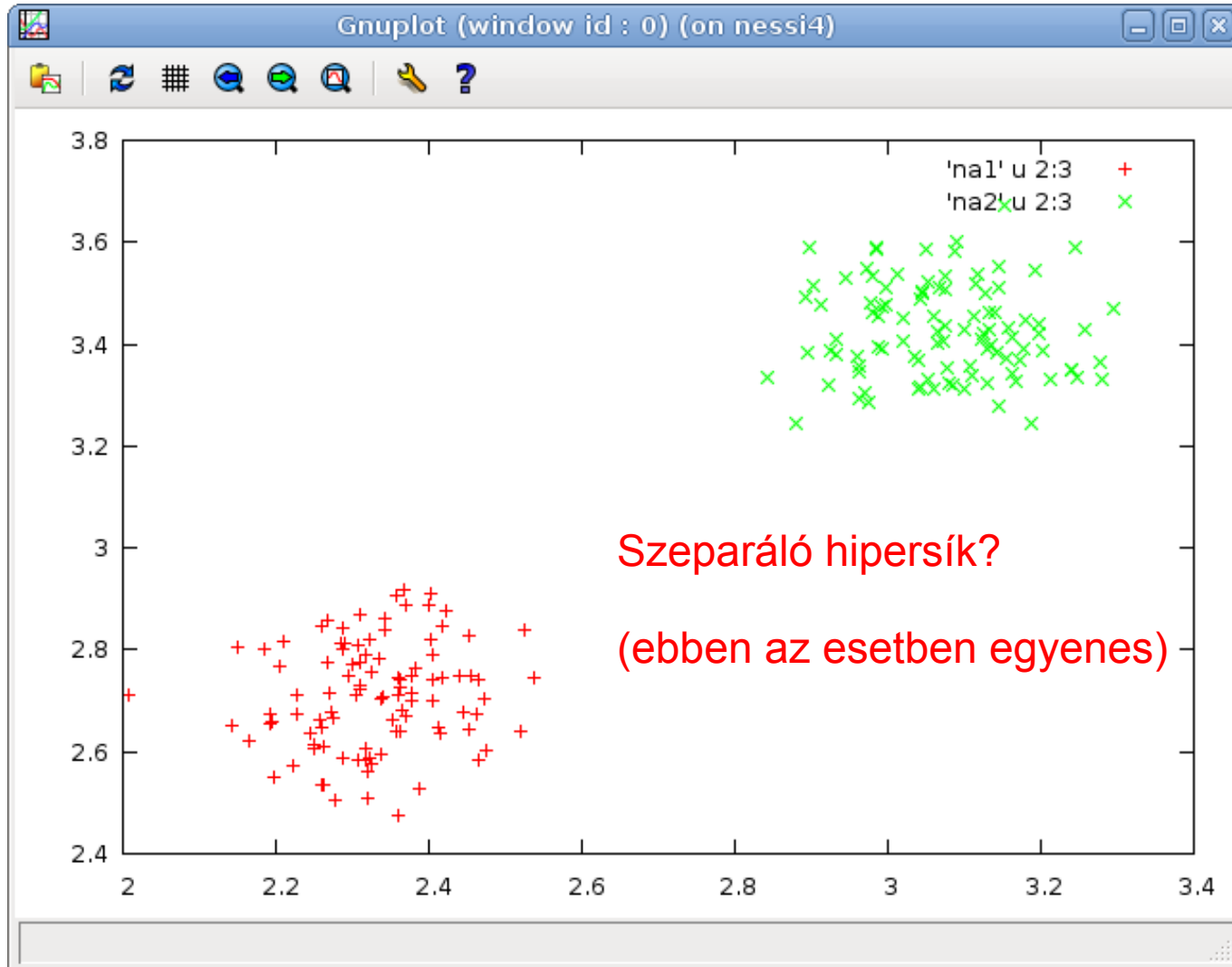
end if

end for

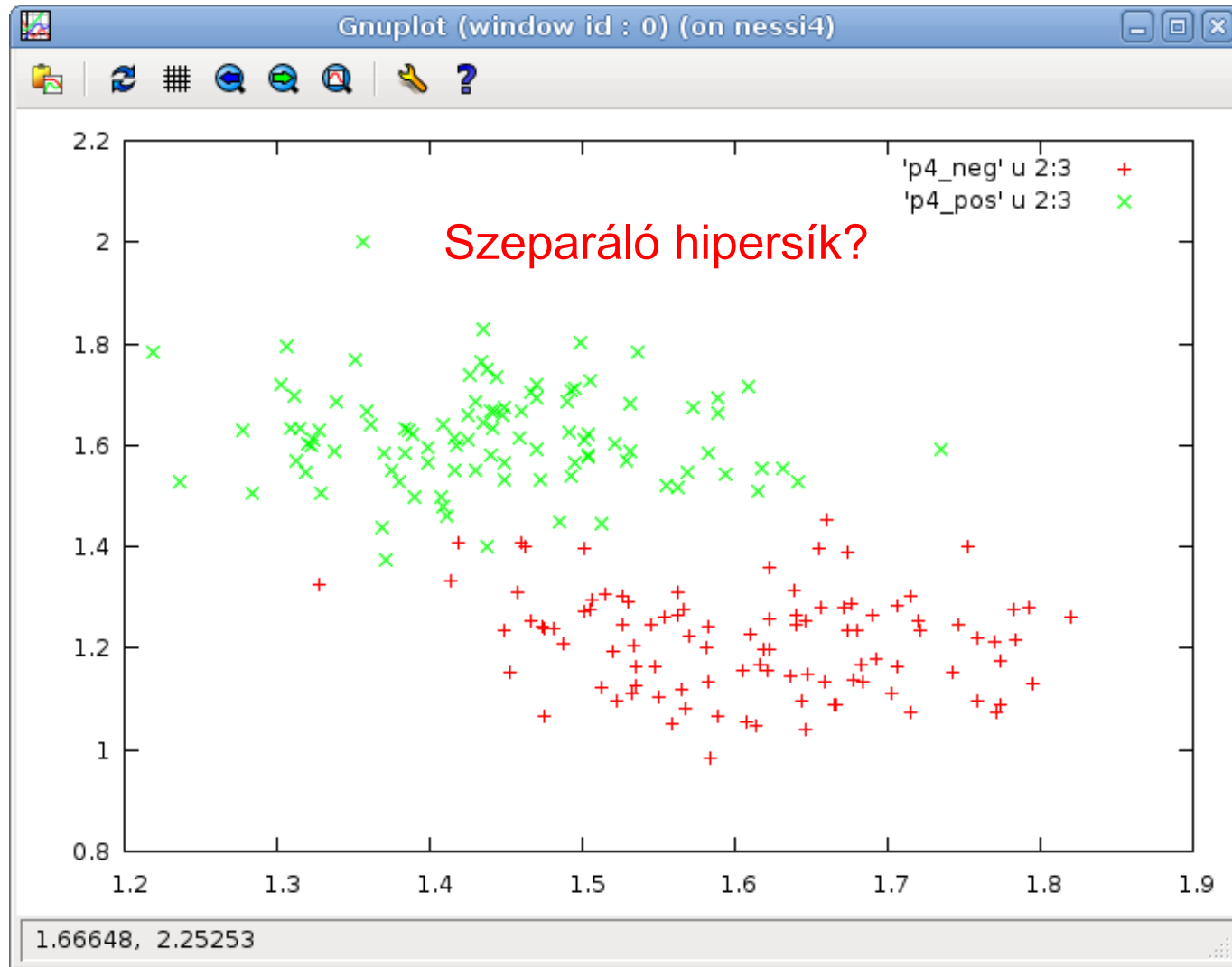
end while

Mikor áll meg?

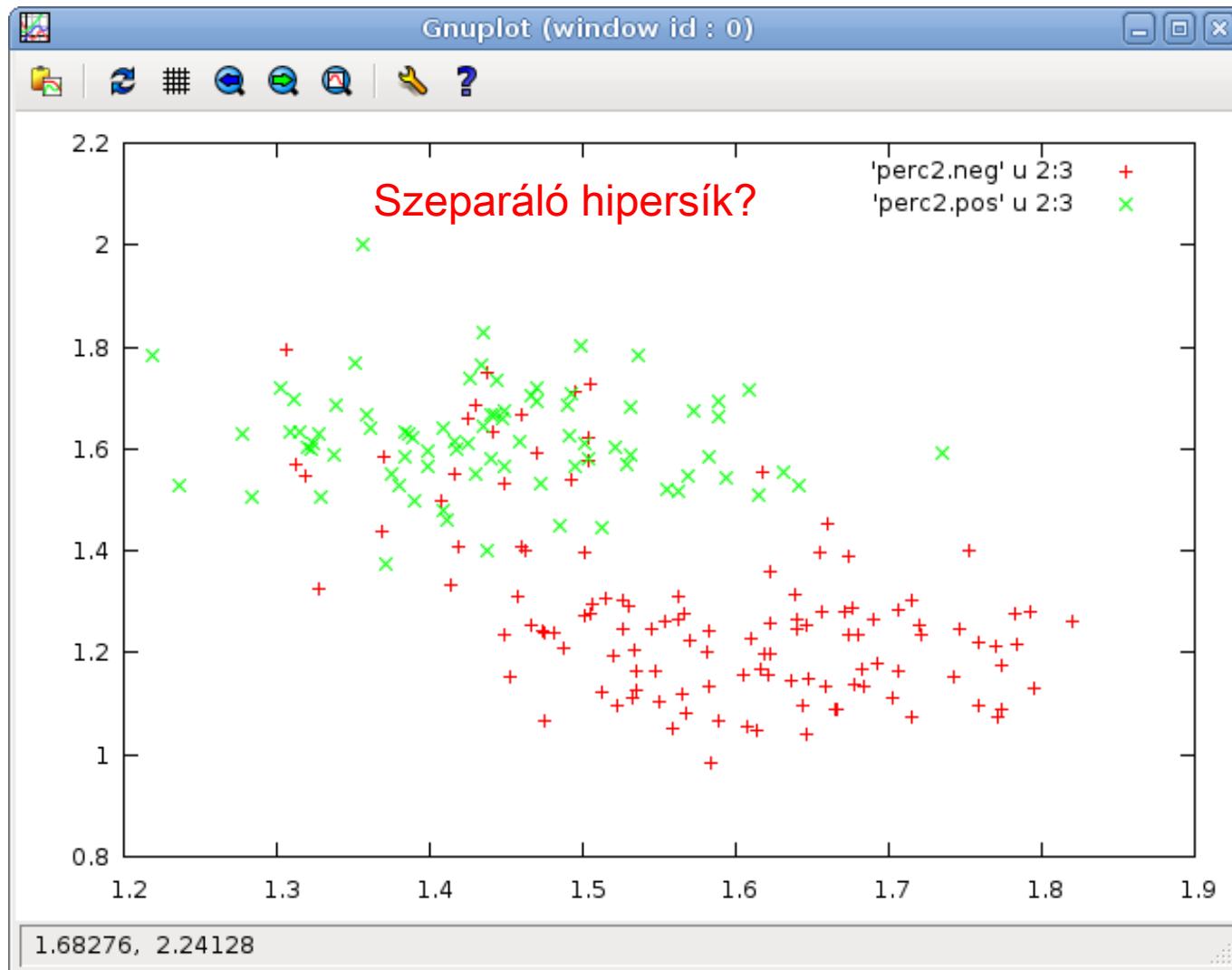
Perceptron



Perceptron



Perceptron



Líneáris regresszió

Tegyük fel újra, hogy a tanulóhalmaz attribútumait kiegészítettük megint egy bias változóval. (X a tanulóhalmaz (N elemű), Y az osztályváltozó) Ebben az esetben az előbbi líneáris kombináció :

$$Y = X^T w$$

Szeretnénk megtalálni a regressziós görbét, melynél a hibát négyzetesen mérjük (lehet más hibamértéket is alkalmazni):

$$error_{square}(f) = E[(Y - f(X))^2]$$

Mivel a tanulóhalmaz véges, a hiba :

$$error(f) = \sum_1^N (y_i - x_i^T w_i)^2$$

Líneáris regresszió

Mátrixábrázolásban (ahogy a bemenő adatot tároljuk):

$$\text{error}(f) = \sum_1^N (y_i - x_i^T w_i)^2 = (y - Xw)^T (y - Xw)$$

Be lehet bizonyítani, hogy mindig létezik minimuma és egyértelmű, így képezzük w szerinti deriváltját, s megkeressük hol 0-a:

$$-2X^T y + 2X^T Xw = 0 \qquad X^T (y - Xw) = 0$$

Melyből következik, hogy ha nem 0-a a determináns \rightarrow

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

Logisztikus regresszió

A lineáris regresszió során egy x pont jóslata nem 0-1 értékeket fog felvenni.

Épp ezért el kell döntenünk hogy mikor döntünk 1-es osztályra vagy 0-as osztályra (feltételezve, hogy eredetileg az osztályváltozók értékkészlete $\{0,1\}$). Amennyiben a jóslás nagyobb 0.5 akkor (ebben az esetben közelebb van 1-hez vagy nagyobb egynél) legyen a jóslat értéke 1, ha pedig kisebb 0-a.

$$y = x^T w - 0.5$$

$$f(y) = y_{osztály} = \frac{1 + \operatorname{sgn}(y)}{2}$$

Sajnos erre függvényre már nem lehet regressziót használni a w meghatározásához.

Logisztikus regresszió

Hogy meghatározhassuk w -t, keresnünk kell egy olyan $f(y)$ függvényt melyre a következő állítások igazak:

1. értéke 1-hez közelít, ha y értéke végtelenhez tart
2. értéke 0-hoz közelít, ha y értéke mínusz végtelenhez tart
3. $f(0) = 0.5$
4. szimmetrikus nullára nézve, tehát $f(y) + f(-y) = 1$ azaz $2f(0)$
5. $f(y)$ differenciálható minden pontban
6. ne legyenek lokális szélsőértékei (pl. monoton növekvő)

Az ilyen típusú függvényeket szigmoid függvényeknek nevezzük. Az irodalomban a következő típusú függvényeket ajánlják:

$$f(y) = 1 / (1 + a^{(-y)})$$

Be lehet bizonyítani, hogy ha $a > 1$, minden ilyen függvény megfelel a kívánalmaknak.

Logisztikus regresszió

Ha a helyére e -t helyettesítünk, megkapjuk a logisztikus függvényt

$$P(Y=1 | X) = 1 / (1 + e^{(-y)})$$

Inverze: $\ln(x/(1+x))$

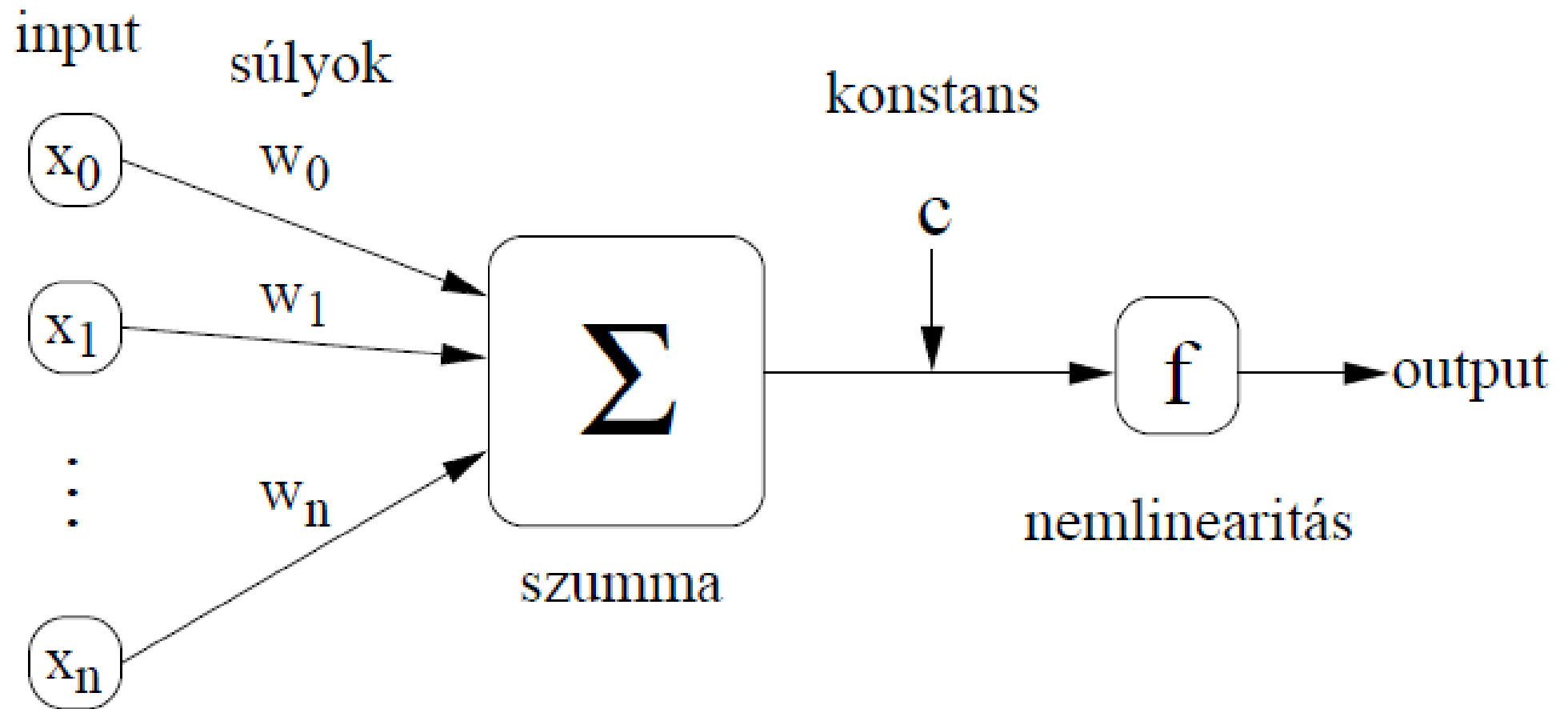
Deriváltja pedig $f(y)(1-f(y))$, mely $P(Y=1 | X)P(Y=0 | X)$ -el egyezik meg (back propagation).

Ebben az esetben logisztikus regresszióról beszélünk. A lineáris regresszióhoz képest, most nem X és Y kapcsolatát szeretnénk egy hibafüggvénnyel meghatározni, hanem $\mathbf{x}^T \mathbf{w}$ és $\ln(P(Y=1 | X)/P(Y=0 | X)) = \ln(P(Y=1 | X)/(1-P(Y=1 | X)))$ közti kapcsolatot.

$$P(Y=1 | X) = 1 / (1 + e^{(-x^{T_w})}) \quad \text{és} \quad P(Y=0 | X) = 1 - 1 / (1 + e^{(-x^{T_w})})$$

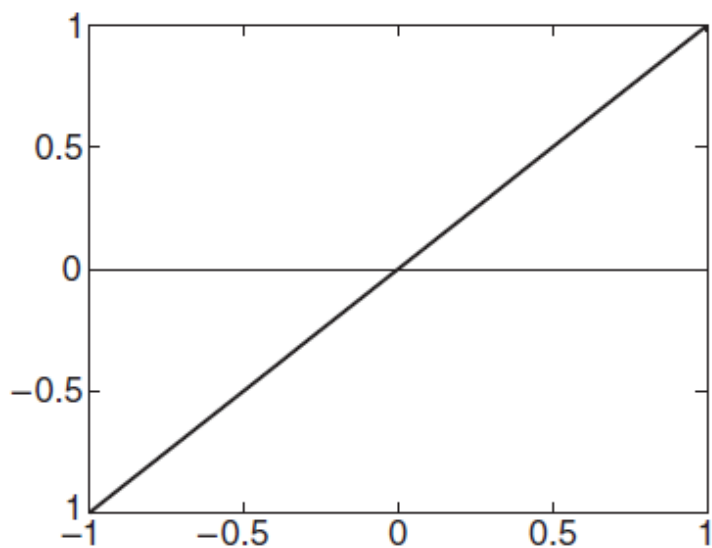
$$= e^{(-x^{T_w})} / (1 + e^{(-x^{T_w})})$$

Logisztikus regresszió

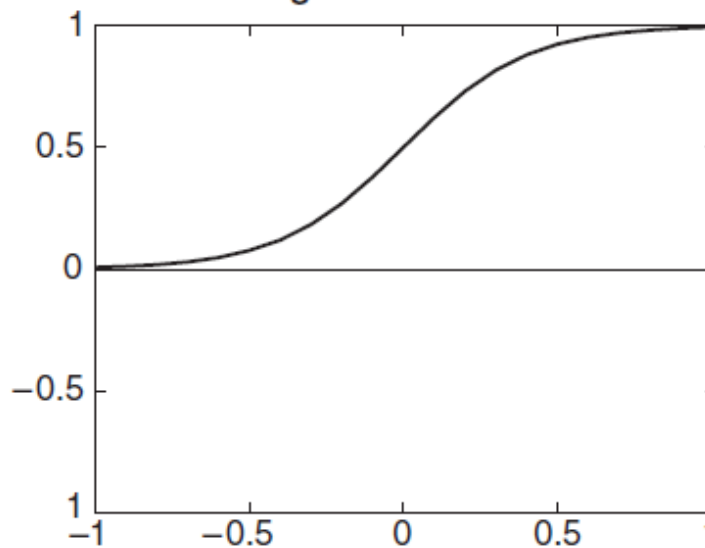


Logisztikus regresszió (aktiválási függvények)

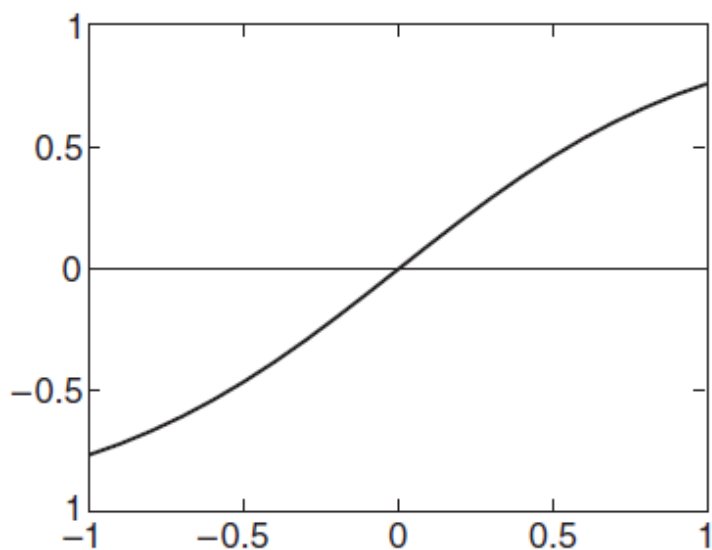
Linear function



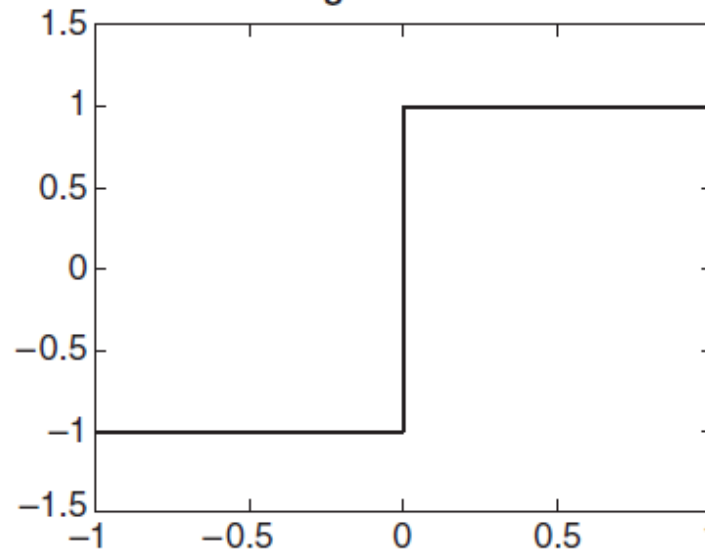
Sigmoid function



Tanh function



Sign function



Logisztikus regresszió

Iteratív módszerekkel szokás meghatározni w -t, melynek alapja, hogy a feltételes valószínűségeket szeretnénk maximalizálni:

$$w = \operatorname{argmax}_w \sum \ln(P(y_i | x_i, w))$$

Viszont felhasználhatjuk, hogy $y=0$ vagy $y=1$:

$$L(w) = \sum y_i \ln(P(y_i = 1 | x_i, w)) + (1 - y_i) \ln(P(y_i = 0 | x_i, w))$$

Kiindulunk $w^{(0)}$ -ból, majd minden iterációban $w^{(1)}$ -hez hozzáadjuk $dL(w)/dw$ (parciális deriváltak) \wedge szorosát (mely egy előre meghatározott konstans): (w_j -re)

$$\sum x_{ij} (y_i - P(y_i | x_{ij}, w_j))$$

Logisztikus regresszió

Gradiens módszer:

Feltételezzük, hogy az osztályváltozóink: $\{-1, 1\}$

$$\frac{dL}{dw} \quad \text{ahol} \quad \frac{dL}{dw_k} = \sum y_i x_i f(-y_i \sum x_i w) = \sum x_i ((y_i + 1)/2 - f(x_i))$$

Minden iterációban szeretnénk a gradiens irányába módosítani a w vektorunkat, megfelelő tanulási növekménnyel.

$$w_k^{(new)} = w_k^{(old)} + \lambda \sum (y_i - f(x_i)) x_{ik}$$

Az algoritmust megállítjuk ha elérjük a maximális iterációszámot, vagy ha az össz változás elér egy előre meghatározott küszöböt.

Mekkora legyen a növekmény? Sajnos adatfüggő, így paraméterként érdemes változtatni.

Logisztikus regresszió

Gradiens módszer előnyei:

- könnyen számolható
- a dL/dw monoton csökkenő (numerikus hibák esetén nem feltétlenül)

Stochasztikus gradiens:

Mivel a hipersíkunk minden egyes iterációban az adatok sorrendjétől függően változik, erősen függ a végső eredmény az adott tanulóponatok sorrendjétől.

Ennek kiküszöbölésére a legjobb módszer, ha minden egyes iterációban véletlen módon határozzuk meg a tanulóponatok sorrendjét. (minden iterációban minden adatpont részt vesz, csak mindig más sorrendben)

- SGD módszerek általában hamarabb konvergálnak -> gyorsabbak
- kevésbé érzékenyek a mintahalmaz eloszlására

A tapasztalat azt mutatja hogy elképzelhető, hogy a gradiens módszer konvergál, míg a sztochasztikus nem.

Logisztikus regresszió

Kiértékelés hiányosságai:

- **túltanulás** veszélye
 - elkerülhető valamilyen szinten, ha **cross-validációt** használunk:
 - a tanuló halmazt felbontjuk N részre, tanulni a komplementer halmazon, a jóslás jóságát azonban a részhalmazon számítjuk
- adott mértékre optimalizált tanulás

Ezen mérték lehet:

- ROC (AUC)
 - MAP
 - P@10 (a találati lista eleje érdekel bennünket)
- stb.

Klasszifikáció liblinear-al

Liblinear:

- open source

<http://www.csie.ntu.edu.tw/~cjlin/liblinear/>

- linear és logistic regression

- C++ library

Nem arff formátumot vár: (ritka mátrix formátum)

<-1,+1> <dim>:x^d

`./train -s 0 <data> <model>`

paraméterek: c, eps

`./predict -b 1 <test_data> <model> <prediction>`

Csak precision-re lehet optimalizálni, ha a beépített CV-t használjuk! → Próbáljuk ki a `perc1.dat` adatunkra!

Órai feladat 1: Klasszifikacio weka-val

Próbáljuk ki a **segment_5.arff** adatot **weka**-val (adatb5.zip)! Hasonlítsuk össze az ismert klasszifikátorokat (K-NN/DT/Bayes/Logistic)!

Az adat 2310 textúrából áll, a 19 attribútum pedig:

1. region-centroid-col: the column of the center pixel of the region.
2. region-centroid-row: the row of the center pixel of the region.
3. region-pixel-count: the number of pixels in a region = 9.
4. short-line-density-5: the results of a line extraction algorithm that counts how many lines of length 5 (any orientation) with low contrast, less than or equal to 5, go through the region.
5. short-line-density-2: same as short-line-density-5 but counts lines of high contrast, greater than 5.
6. vedge-mean: measure the contrast of horizontally adjacent pixels in the region. There are 6, the mean and standard deviation are given. This attribute is used as a vertical edge detector.
7. vegde-sd: (see 6)
8. hedge-mean: measures the contrast of vertically adjacent pixels. Used for horizontal line detection.
9. hedge-sd: (see 8).
10. intensity-mean: the average over the region of $(R + G + B)/3$
11. rawred-mean: the average over the region of the R value.
12. rawblue-mean: the average over the region of the B value.
13. rawgreen-mean: the average over the region of the G value.
14. exred-mean: measure the excess red: $(2R - (G + B))$
15. exblue-mean: measure the excess blue: $(2B - (G + R))$
16. exgreen-mean: measure the excess green: $(2G - (R + B))$
17. value-mean: 3-d nonlinear transformation of RGB. (Algorithm can be found in Foley and VanDam, Fundamentals of Interactive Computer Graphics)
18. saturatoin-mean: (see 17)
19. hue-mean: (see 17)

7 osztály:

1. tégl
2. ég
3. fa
4. beton
5. üveg
6. földút
7. fű

Support Vector Machine

A logisztikus regresszió esetében egy szeparáló hipersíkot kerestünk, amely reményeink szerint optimálisan (pl. a tanulóhalmaz pontjaitól minél távolabb) helyezkedett el.

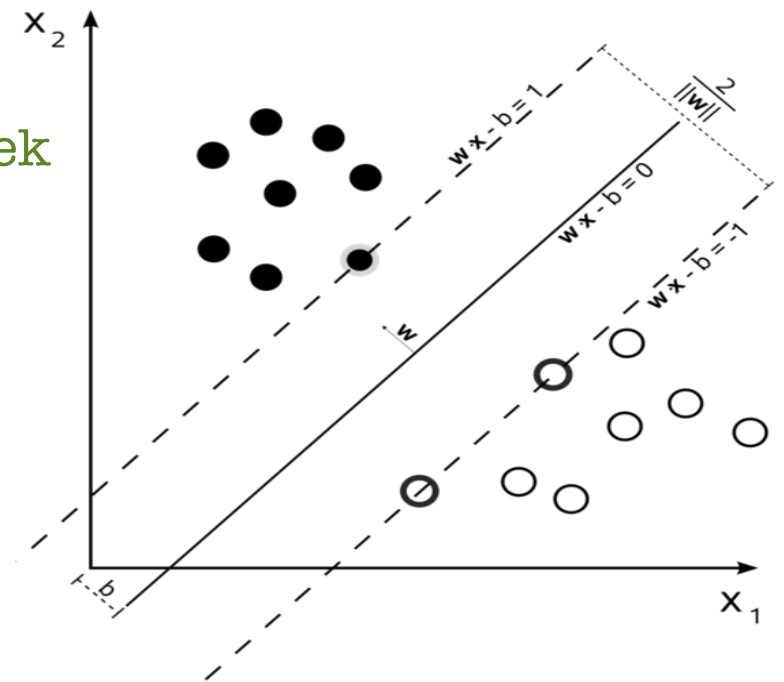
A $w \cdot x - b > 0$ (ahol b az origótól vett távolság) akkor az első osztályba soroljuk a pontot, ha pedig kisebb a másodikba.

Szeretnénk két egymással párhuzamos hipersíkot, melyre igaz a következő:

$w \cdot x - b = 1$ és $w \cdot x - b = -1$, úgy hogy köztük nincsenek pontok a tanulóhalmazban

A két hipersík távolsága $2 / ||w||$

Ebben az esetben a két a első illetve második halmazból vett pontot **támogató (support)** **vektor**oknak nevezzük



Support Vector Machine

Ha $w \cdot x - b \geq 1$ akkor az első osztályba kerül x

Ha pedig $w \cdot x - b \leq -1$ akkor a másodikba (értelmszerűen a tanulópontoknál ez feltétel)

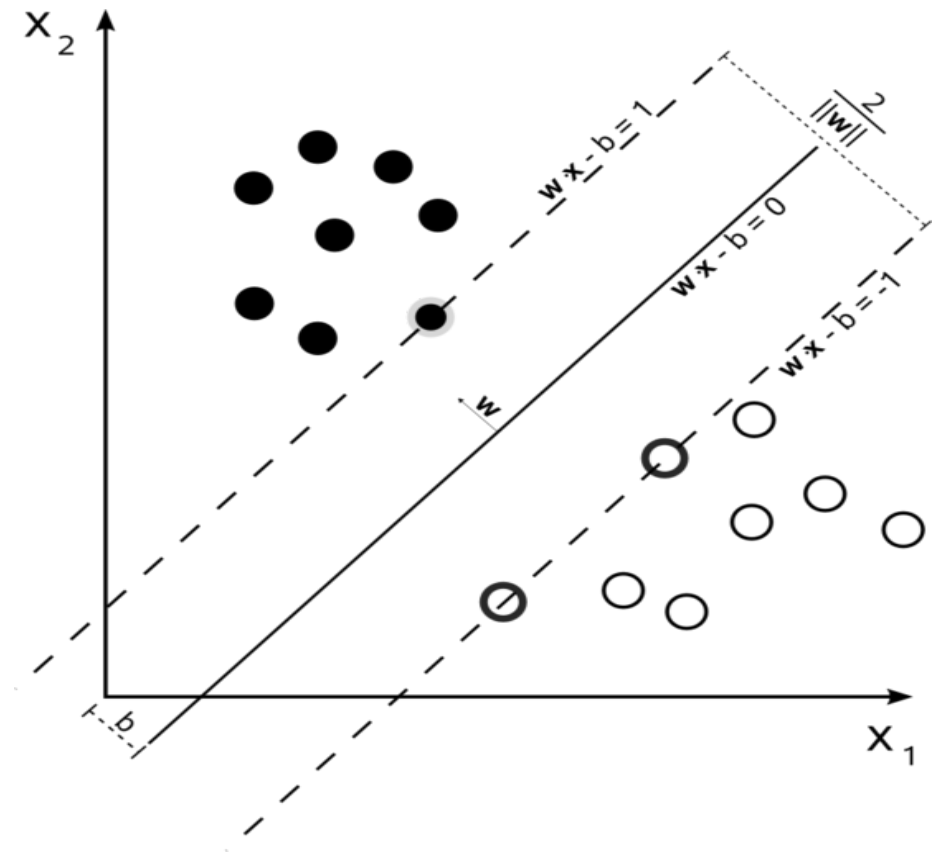
Abban az esetben ha lineárisan szeparálható az adat!

A feladat $\|w\|$ minimalizálása, hiszen ekkor a legnagyobb a két határhipersík távolsága

$$y_i(x_i \cdot w + b) - 1 \geq 0$$

Kvadratikus programozási feladat

$$\alpha_i \geq 0 \sum_{i=1}^n \alpha_i y_i = 0$$



Support Vector Machine

$$y_i(x_i * w + b) - 1 \geq 0$$

A tanulóhalmaz n pontból áll, keressük azt a K pontot (Support vectort) melyre igaz, hogy van olyan rajtuk átmenő párhuzamos hipersíkpár, melyek távolsága maximális

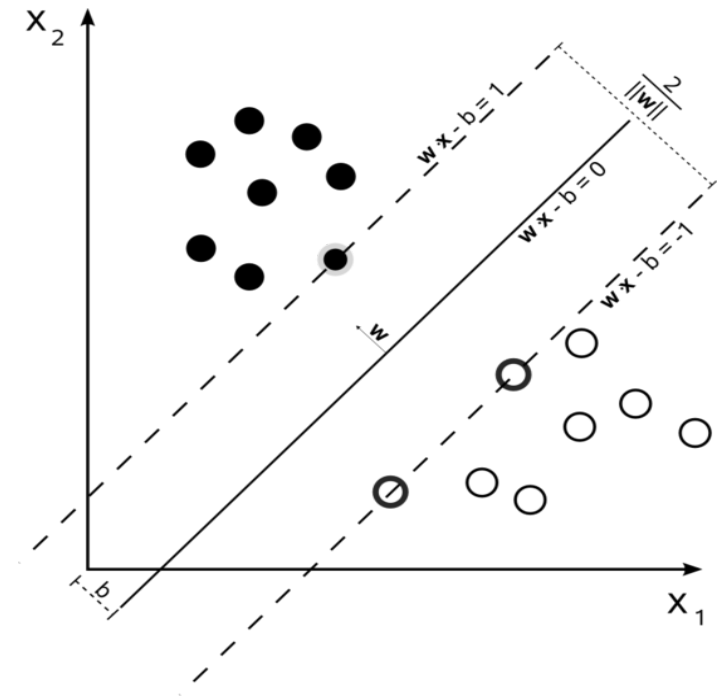
$$\alpha_i \geq 0 \quad \sum_{i=1}^n \alpha_i y_i = 0$$

Langrange szorzók segítségével a parciális deriváltak után ($K(x,y)$ a mag (kernel) függvény):

$$\max \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

Az így keletkezett kimeneti függvény:

$$g(x) = wx + b = \sum_{i \in \text{Support Vectors}} \alpha_i K(x_i, x) + b$$



Support Vector Machine

$$y_i(x_i * w + b) - 1 \geq 0$$

A tanulóhalmaz n pontból áll, keressük azt a K pontot (Support vectort) melyre igaz, hogy van olyan rajtuk átmenő párhuzamos hipersík-pár, melyek távolsága maximális

$$\alpha_i \geq 0 \quad \sum_{i=1}^n \alpha_i y_i = 0$$

Langrange szorzók segítségével deriváltak után ($K(x, y)$ függvény):

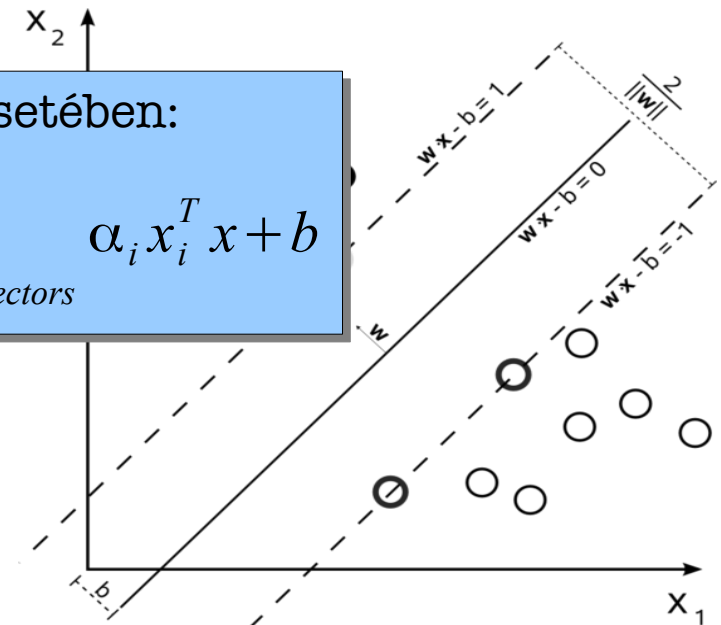
$$\max \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

Líneáris kernel esetében:

$$g(x) = wx + b = \sum_{i \in \text{SupportVectors}} \alpha_i x_i^T x + b$$

Az így keletkezett kimeneti függvény:

$$g(x) = wx + b = \sum_{i \in \text{SupportVectors}} \alpha_i K(x_i, x) + b$$



Support Vector Machine

Soft margin SVM , hibát is megengedünk a marginokhoz képest:

$$y_i(x_i * w + b) - 1 \geq -\xi_i$$

Líneárisan nem szeparálható adatok esetében is alkalmazható az előbbi számítás.

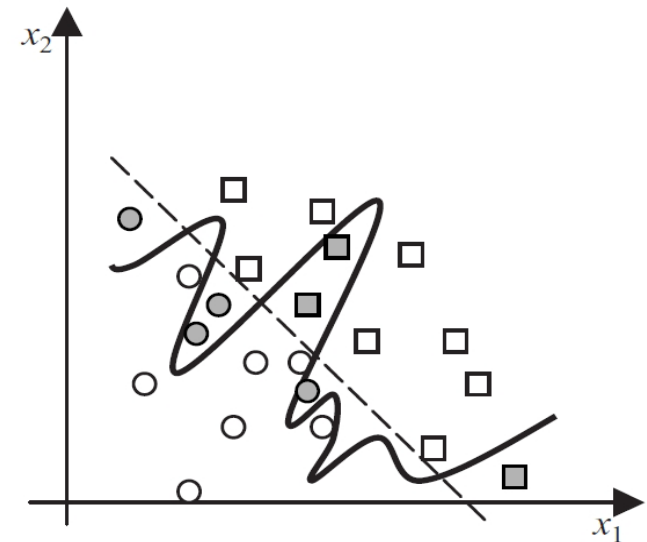
Transzformáljuk át a pontjainkat a egy magasabb dimenzióba s a transzformált térben határozzuk meg a szeparáló hipersíkjunkat:

Líneáris kernel: $K(u,v) = u * v$

Polinomiális kernel: $K(u,v) = u * v + 1$

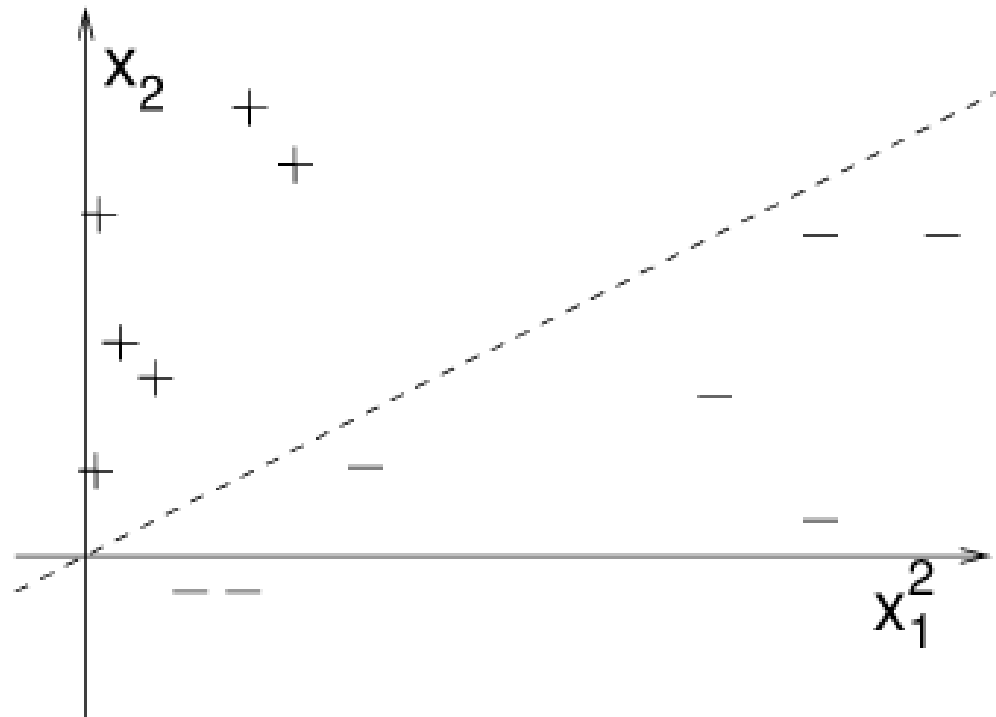
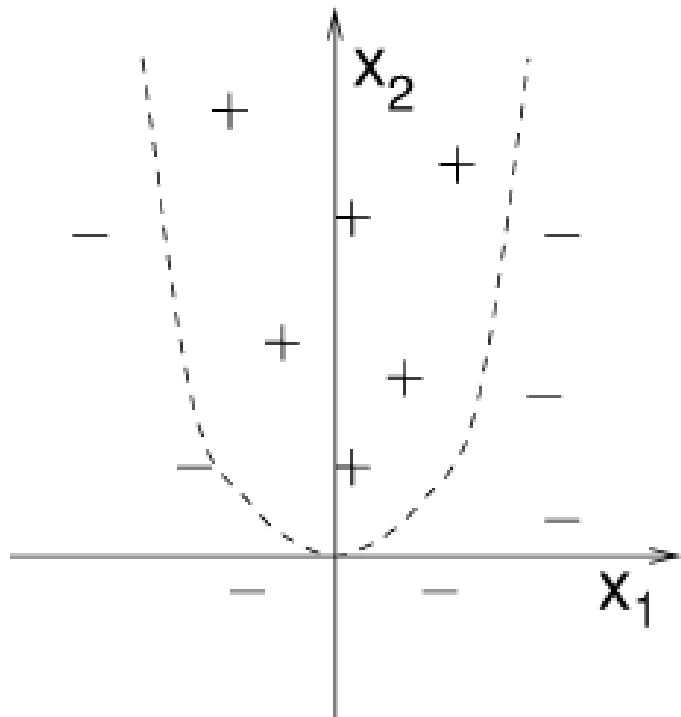
Radial Basis Function: $K(u,v) = \exp(-\lambda ||u-v||)$

A kernel függvénynek nem szükséges véges dimenziójúnak lennie!



Egy példa: az átttranszformált térben lehetséges lineárisan szeparálni (a kép egy vetület)

$$x = (x_1, x_2) \text{ a transzformált térben } x' = (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{(2)}x_1x_2, 1)$$



LibSVM

Open source, jól optimalizált s sokrétű:

<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

- ritka mátrixokra is működik
- érthető s könnyen fejleszthető
- gyors (főleg egy kis oprimalizálással)
- folyamatosan fejlesztik (CUDA ext.)
- C/Ruby/C#/Matlab stb.
(Windows alatt Matlab vagy Cygwin ajánlott)

Formátum és használat megegyezik a LibLinear-al:

Előre számolt kerneleket is használhatunk!

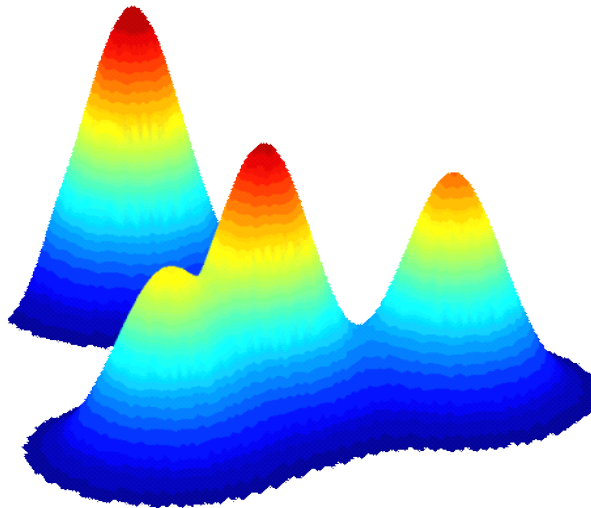
Amennyiben nem csak osztálykimenetet szeretnénk kapni, cross-validációval dönti el a vágó paramétert. Ez hat külön tanítást jelent, helyette egy kis módosítással a sorrendet megkaphatjuk (AUC,MAP!)

Kernelek használata

Amennyiben bármely lineárisan szeparálható véges dimenziójú térbe transzformáljuk a meglévő tanulóhalmazunkat, használhatóvá válnak az eddig is ismert tanító algoritmusok

Az SVM előnye, hogy az optimalizálás során már figyelembe veszi a kernel függvényt.

A jó kernelek sokszor nagyon magas vagy végtelen dimenziójúak, így nem használhatóak vagy csak korlátokkal a meglévő algoritmusok (kép: RBF kernel végtelen dimenziós egységgömb felületére képezi a pontokat)



Kernelek használata

Általános trükk, hogy a kerneleket megelőzendő, egy ál-kernelt számolunk:

Amennyiben a tanulóhalmaz jól szeparálható, feltételezzük, hogy egy ismeretlen elem tanuló halmaz pontjaitól vett távolsága erős reprezentáns:

$$K'(X, T_i \in \{T_0, T_1, \dots, T_N\}) = 1 - \text{dist}_p(X, T_i)$$

Minkowski (L1, L2) és χ^2 kernelek:

$$\text{dist}_{L^p}(x, t_i) = \left(\sum_{j=1}^d |x_j - t_{ij}|^p \right)^{\frac{1}{p}}$$

$$\text{dist}_{\chi^2}(x, t_i) = \frac{\sum_{j=1}^d (x_j - t_{ij})^2}{\frac{1}{2}(x_j + t_{ij})}$$

Weka command-line

MS Windows

Keressük meg a `weka.jar` (általában: `C:\Program Files\Weka-3-6`)

- jobb klikk a `My Computer`-en majd válasszuk ki `Properties`
- az `Advanced` fülben -> `Environment Variables`

Nevezzük el a környezeti változónkat:

`CLASSPATH`

s adjuk hozzá a `weka.jar` fájlunk elérését

`C:\Program Files\Weka-3-6\weka.jar`

(ellenőrzés: `%CLASSPATH%`)

Linux

Tegyük fel hogy a `weka.jar` a `/home/johndoe/weka-3-6/`-ban található

```
export CLASSPATH=$CLASSPATH:/home/johndoe/weka-3-6/weka.jar
```

Usage:

Példa: `java weka.classifiers.trees.J48 -t data/iris.arff`

Classifiers: `java weka.classifiers.*` (training -t)

Filters: `java weka.filters.supervised.attribute.*` (input -i)

Classifier parameters

- o output model
- T test file
- x cross validation
- d model output (human readable)
- l loads a previously saved model
- p show predictions
- i more detailed performance evaluation

Gnuplot használata

indítás
gnuplot

set terminal png
set output 'proba.png'

plot „perc2” using 2:3

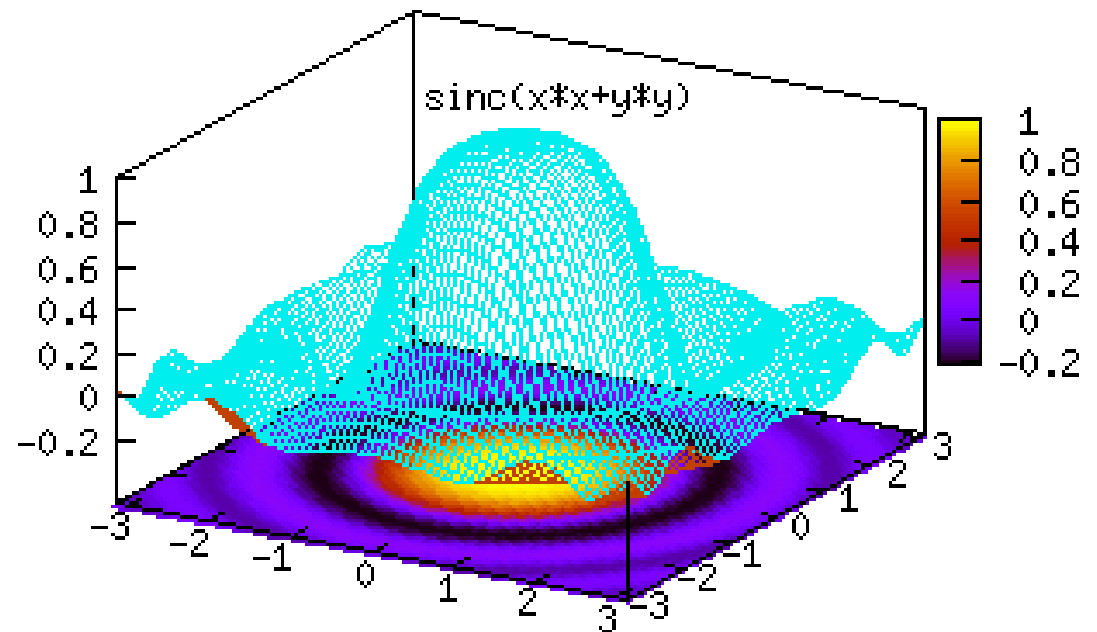
vonalak használata
pontok helyett

par hasznos:

with lines

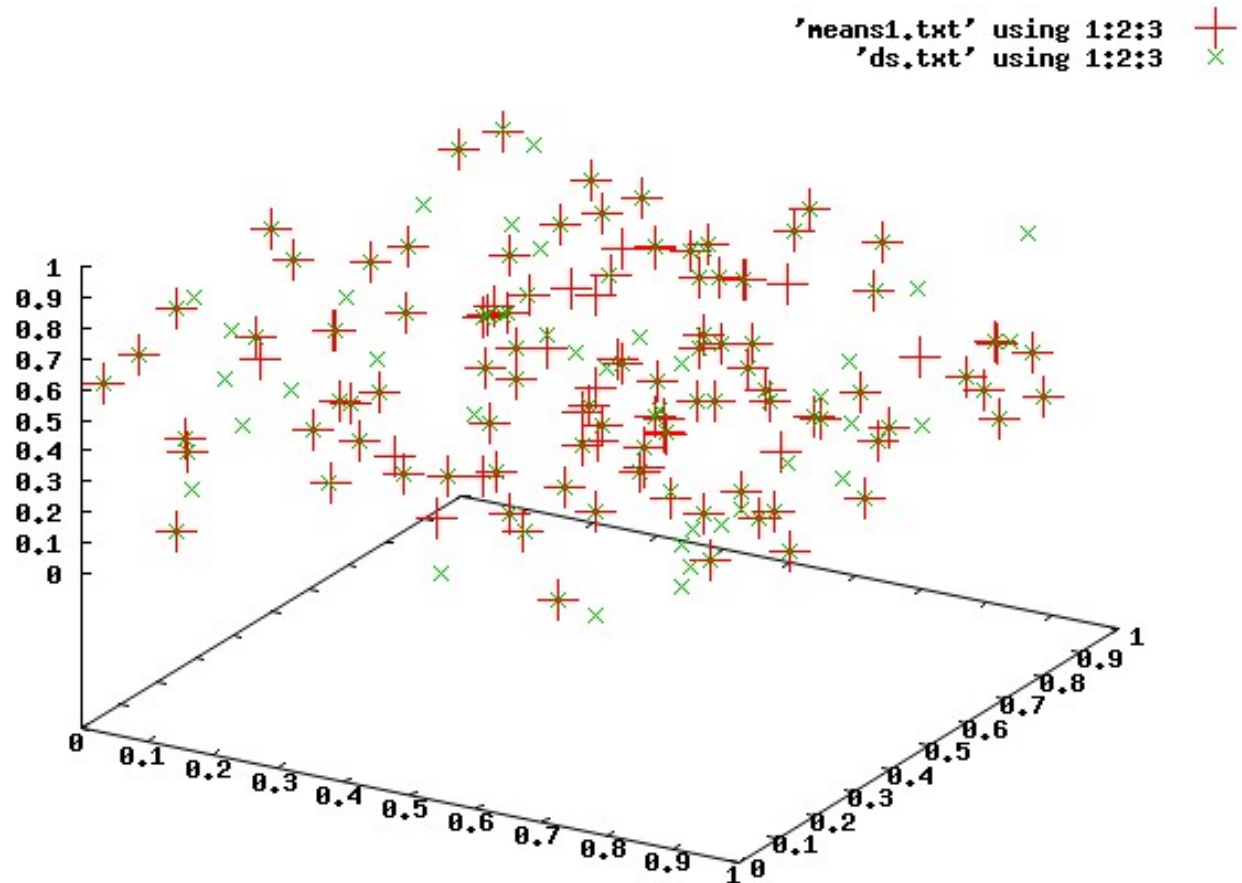
y tengelyt logaritmikusra
rakni:
set log y (unset)

visszaállítás
unset log y



Gnuplot

`abs(x)`
`cos(x)`
`sin(x)`
`set xlabel`
`set ylabel`
`set title`
`set xrange`
`set pm3d`
`splot`
`using 2:4`



Órai feladat 2: Gnuplot

Ábrázoljuk a perc2,perc3,perc4 adathalmazokat!

Ábrázoljuk a szeparáló hipersíkkunkat! Normalizáció?

Egyenes egyenlete -> $y=?$

