Überleitung zu Neuronalen Netzen und Support Vector Machines

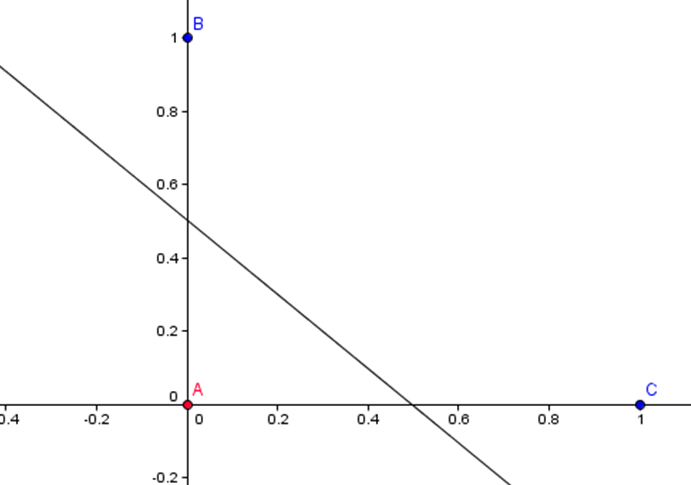
Es seien die Vektoren

x(1) = (0,0) y(1) = -1

x(2) = (1,0) y(2) = 1

x(3) = (0,1) y(3) = 1

gegeben. Aus der Skizze ist sofort zu entnehmen, dass sich die Klassen mit y= - 1 und y=1 einfach durch eine lineare Funktion trennen lassen:



Eine andere Idee als die lineare Diskriminanzanalyse hatte Rosenblatt mit seinem Ansatz zu neuronalen Netzen. Er fasst die beiden Variablen, die hinter den x Eingabewerten stecken als Eingaben in die input Schicht (input layer) eines neuronalen Netzes auf und die y-Werte als Ausgaben der Ausgabeschicht (output layer). Mit neuronalen Netzen soll die Gehirntätigkeit des Menschen nachgebildet werden. Neuronen (abgebildet durch Kreise) empfangen Reize, diese werden an die nächste Schicht mit Gewichtung weitergegeben (negative Gewichtung bewirkt eine Schwächung des Reizes). Ist die Summe der Reize, die bei der nächsten Schicht ankommen größer als ein Schwellwert, wird der Reiz weitergegeben (das Neuron feuert), sonst wird er ‚absorbiert‘. Feuert das einzige Ausgabeneuron im Beispiel, wird die Eingabe der einen Klasse (y= -1) bzw. der anderen Klasse zugeordnet. Ziel ist es, dass Neuronale Netz so lange zu trainieren –dh. die Gewichte und Schwellwerte zu ermitteln- bis es (möglichst) fehlerfrei die Trainingsmenge abbildet.

x1

1

w1

3

y

x2

2

w2

b

Input Layer Output Layer

Ist also die Summe der Reize, die beim Knoten im Output Layer größer als der Schwellwert

(threshhold) b, ‘feuert‘ das Neuron, entscheidet sich also für die Klasse y = 1; für y = -1 sonst.

Die Summe der Reize berechnet sich zu:

x1 \* w1 + x2 \* w2

Das ist gerade das Skalarprodukt der Vektoren x und w:

<

Ist dieses Produkt größer als b entscheidet sich das Netz zu feuern (also

Die Trennung zwischen den Klassen erfolgt also anhand der Trennlinie

< + b = 0

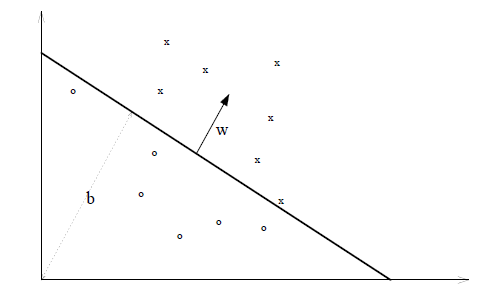
Das ist aber gerade die Kennzeichnung einer Gerade in der Ebene (bei höheren Dimensionen einer Hyperfläche). W steht senkrecht (orthogonal) auf der Geraden (Hyperebene) { x | <x,w> + b = 0 }.

Wenn w auf die Länge 1 gebracht wird (also w / ||w|| betrachtet wird), ist der Abstand der Geraden zum Nullpunkt gerade b.

Liefert das Netz also einen Wert mit positivem (negativem)Vorzeichen für diejenigen Eingabewerte, bei denen yi +1 (bzw. -1) ist, produziert das Netz die richtigen Ergebnisse; das bedeutet aber, wenn

ist, produziert das Netz die richtigen Ergebnisse; sonst muss nachjustiert werden.

Daraus ergibt sich der folgende Algorithmus: (eine Herleitung ist im Anhang zu finden)



{ x | <x , w > + b = 0 }

w0 := ; b := 0 ; k := 0

repeat

for i=1 to n

if

wk+1 := wk + yi \* x(i) bk+1 := bk + yi

k := k + 1

end if

end for

until no mistakes made within the *for* loop

(n bezeichnet dabei die Anzahl der Elemente der Trainingsmenge)

Rechnet man das im Beispiel durch, erhält man:

Erster Durchlauf (i=1) (-1) ( , also

(i=2) (+1) , also

(i=3) (+1) , also

Zweiter Durchlauf (i=1) (-1) ( , also

(i=2) (+1) 0 , also kein Handlungsbedarf

(i=3) (+1) , also kein Handlungsbedarf

usw.; erst der sechste Durchlauf beendet den Algorithmus mit und b9 = -1

(die anderen Iterationen befinden sich im Anhang)

(i=1) (-1) ( , also kein Handlungsbedarf

(i=2) 1 ( , also kein Handlungsbedarf

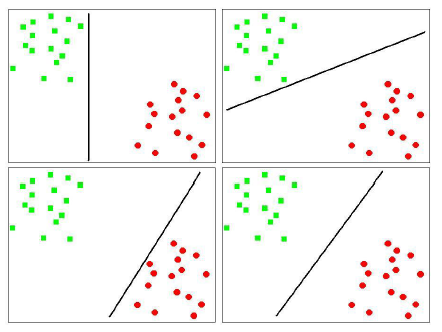
(i=3) 1 ( , also kein Handlungsbedarf

Die ISO Linie zur Trennung der beiden Klassen berechnet sich also durch:

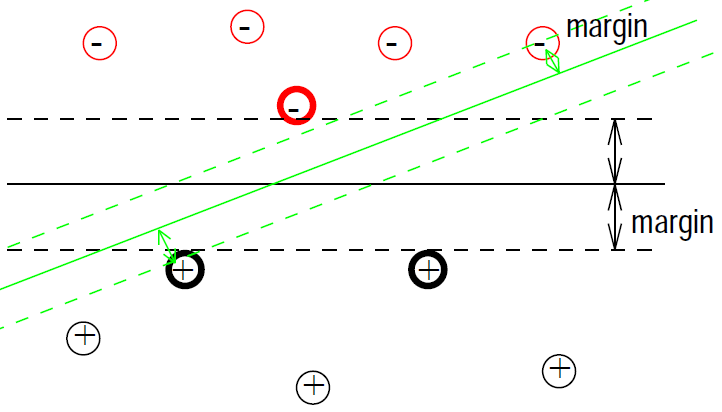
, dh. , also

Das ist genau die Trennlinie von der ersten Seite. Es lässt sich allerdings zeigen, dass der Algorithmus für solche zweischichtigen Netze nur dann konvergiert, wenn die Klassen linear ‚separierbar‘ sind. Mit dreischichtigen (oder mehrschichtigen) Netzen kann man dieses Manko beheben. Allerdings verliert man dabei die Einfachheit des obigen Algorithmus.

Auch die Überlegungen bei dem Support Vector Machine Ansatz startet zunächst bei linearen Trennungen. Welche der (jeder für sich perfekten) linearen Trennungen wird vermutlich die ‚stabilste‘ Trennung bewirken:



Den Zuschlag erhält die Lösung mit einem möglichst breiten Rand unten rechts. Bei der Berechnung des größten Randes spielen nur diejenigen Objekte eine Rolle. die dem Rand am nächsten liegen:

Die Anzahl der andern Werte spielen –anders als bei einigen anderen Verfahren- keine Rolle; das bewirkt eine große Robustheit.

Vor dem Maximieren des margins muss noch folgendes Problem beseitigt werden:

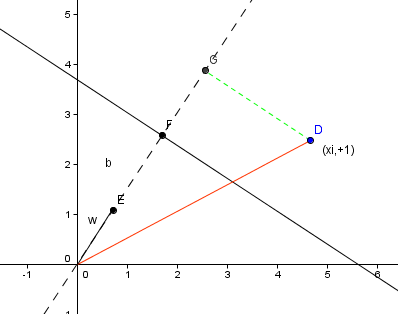
Für c≠0 gilt { x | <w,x> + b = 0 } = { x | < cw,x> + cb = 0 }

mit andren Worten: (cw,cb) beschreibt die gleiche Hyperebene wie (w,b); sie läßt sich also nicht eindeutig beschreiben. Deshalb wird (w,b) relativ zu den Trainingsdaten {xi,yi}i=1,n normiert:

(#)

Es wird zunächst davon ausgegangen, dass sich die Klassen perfekt trennen lassen. Der Abstand eines Punktes xi der Klasse yi ∈ {-1,+1} zu einer Hyperebene { x | <w,x> + b = 0} lässt sich berechnen zu:

||w|| steht für die Länge (Norm) des Vektors w.



z

w ist hier schon von der Länge 1 (also gleich w/||w||). Das Skalarprodukt <w,z> (z sei der Vektor, der

von 0 zu D zeigt) ist gleich der Länge der Strecke 0 bis G. b (auch hier schon b/||w|| ist die negativ genommene Strecke von 0 bis F (also des Abstandes der Hyperebene vom 0 Punkt). <w,z> + b ist also die Länge der Strecke F bis G, also des Abstands des Punktes D von der Hyperebene. Da yi hier +1 ist

misst yi (<w,z>+b) also hier den Abstand des Punktes D von der Hyperebene. Läge ein Punkt auf der anderen Seite der Hyperbene (wo die yi = -1 sind) misst dieser Ausdruck die Länge des Stückes von der senkrechten Projektion auf den Vektor w (dieser Punkt liegt ja zwischen 0 und F) nach F, also den Abstand von der Hyperebene. Die Multiplikation mit yi (=-1) korrigiert den negativen Wert von

(<w,z>+b).

In (#) wurde so skaliert, dass für Trainingspunkte (x1,+1) und (x2,-1) gilt:

also und

und damit

Den margin maximieren heißt also ||w|| minimieren; das ist gleichbedeutend mit der Minimierung von ||w||2

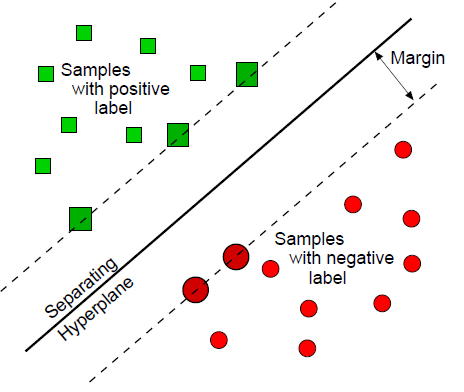
Damit die Hyperebene die Trainingsdaten auch wirklich trennt, werden die folgenden Nebenbedingungen eingeführt:

für i = 1,….,n . Es entsteht also ein Problem der Quadratischen Programmierung. Es werden Lagrange-Multiplikatoren αi ≥0 eingeführt und die Lagrange-Funktion minimiert für w und b und maximiert für die αi maximiert:

die partiellen Anleitungen von L nach w und b werden 0 gesetzt und man erhält:

und (+)

Nach einigen Umformungen erhält man das dazu duale Problem und löst nach den αi auf. Nur die Punkte, die der Hyperebene am nächsten liegen (die nennt man support vectors –„tragende Vektoren“) haben αi ≠ 0. Alle anderen Punkte haben keinen Einfluss.



(aus Florian Markowetz: Klassifikation mit SVM)

Mit den αi hat man auch den Normalenvektor w gefunden mit (siehe (+))

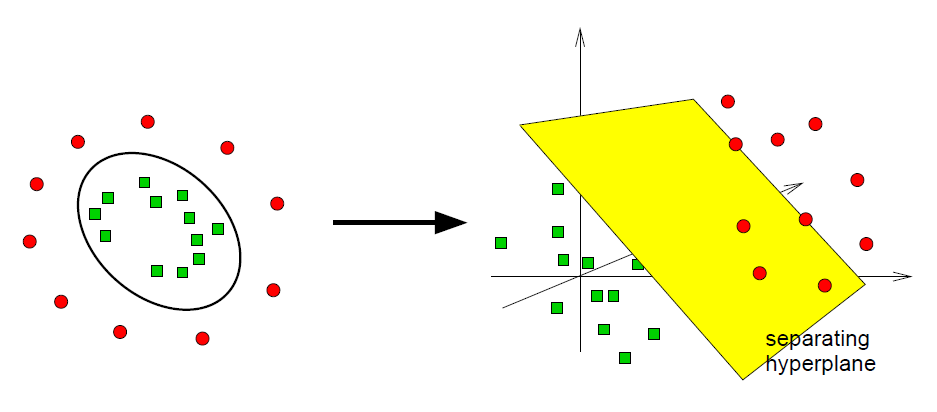
Damit hat man auch die Entscheidungsfunktion gefunden für die Klassifizierung von neuen Werten:

Das Vorzeichen der der Funktion <w,xneu> + b =

oder f(xneu) = sign (, wenn man die sign Funktion nutzt:

entscheidet über die Zuordnung zur Klasse.

Da beim Optimisierungsproblem und bei der Lösung die xi lediglich im Skalarprodukt vorkommen, macht man sich das für einen Verallgemeinerung zu Nutze, die auch auf nicht linear separierbare (im Ursprungsgsraum) Probleme angewendet werden kann:



Im Ursprungsraum ist das Problem nicht linear separierbar, aber im so genannten Feature Raum Η, in dem ein Skalarprodukt definiert ist. Die durch den Pfeil angedeutete Abbildung vom 2 in den dreidimensionalen Raum könnte z.B. wie folgt aussehen:

Zum Berechnen der Hyperebene muss man die Skalarprodukte der Form <Φ(p), Φ(q)> ausrechnen.

Das kann bei höher dimensionalen Räumen schwierig sein. Man behilft sich mit einer so genannten Kernel Funktion, die im Ursprungsraum definiert ist, sich aber im feature space wie ein Skalarprodukt verhält:

K(p.q) = <Φ(p), Φ(q)>

Im obigen Beispiel kann man das Skalarprodukt <Φ(p), Φ(q)> berechnen durch:

( p=(p1,p2) , q=(q1,q2) )

=

=

=

Man kann also das Skalarprodukt <Φ(p), Φ(q)> auszurechnen ohne die Funktion Φ zu berechnen oder überhaupt zu kennen. Es reicht, das Quadrat des üblichen Skalarproduktes im IR2 zu berechnen. Das Vorgehen wird in der Literatur auch ‚Kernel-Trick‘ genannt. Trick deshalb, weil man gar nicht wissen muss wie der feature space und die Abbildung dahin aussieht. Das hat etwas von einem Black box Verfahren an sich. Trotzdem ist die geometrische Interpretation der trennenden Hyperebene anschaulicher als etwa die bei neuronalen Netzen. Übliche Kernel Funktionen sind neben der eben schon kennen gelernten linearen

* linear K(p,q) = <p.q>
* polynomial vom Grad d K(p.q) = ( γ <p,q> + c0 ) d (c0 mitVoreinstellung 1)
* radial basis function K(p.q)= exp (-γ || p-q ||2 ) (auch Gauss Kernel genannt)

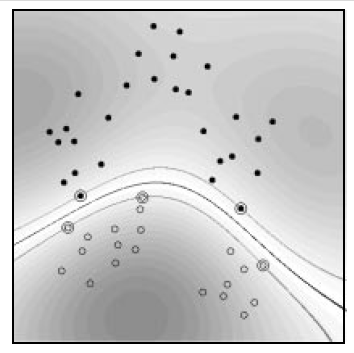
Frage: Wie sieht die Funktion Φ aus zum polynomialen Kernel vom Grad 2, γ=1 und c0 = 1 ?

[ also K(p.q) = ( <p,q> + 1 ) 2 ] Welche Dimension hat der feature space?

Lösung siehe Anhang

Vorteile von SVM:

* Klassifikation sehr schnell möglich (basierend auf wenigen Support Vektoren)
* hohe Generalisierungsfähigkeit (z.B. für nicht linear separierbare Probleme:



(aus : Schölkopf - Statistical Learning an Kernel Methods 20.2.2000)

Nachteile:

* neues Training für neue (verschiedene) Eingabedaten erforderlich
* Wahl des Kernels schwierig (muss empirisch gesucht werden)

**Herleitung des Algorithmus für das neuronale Netz:**

Der Schwellwert b wird als drittes Gewicht (w3) aufgefasst, das eine mit (+1) konstante Eingabe weiterleitet:

x1

1

w1

4

y

w2

x2

2

w3

3

**1**

Der Gesamtfehler der quadrierten Einzelfehler, den das neuronale Netz produziert (Bezeichnung Etotal), berechnet sich zu:

Etotal =  P bezeichne die Anzahl der Input Vektoren

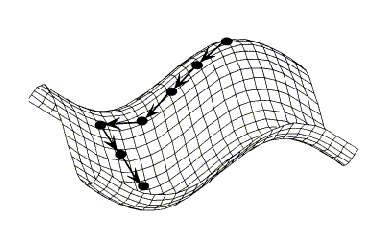
 yi ist das, was beim i-ten Input Vektor heraus kommen soll;

 bezeichnet das, was das Netz bei Eingabe des i-ten Inputs liefert

dann gilt mit der Kettenregel:

) da = für j=1,2 und für j=3:

Da bekanntlich der Gradient in die Richtung des stärksten Anstieges einer Funktion zeigt, folgt man hier dem negativen Gradienten in der Hoffnung die Funktionswerte (hier die Fehler) zu verkleinern:



Die Länge der Strecke, die man dem negativen Gradienten folgt heißt Lernrate. Hier werde die Lernrate auf 0.25 gesetzt. Bei kleiner Lernrate nähert man sich der Lösung eventuell zu langsam; bei zu großer ‚überspringt‘ man eventuell das ‚Tal‘, in dem das Minimum liegt. Dann korrigiert man den Gewichtsvektor **W** , indem man die Lernrate multipliziert mit dem negativen Gardienten dazu addiert:

für j=1,2 und für j=3:

Stimmen die Werte in den runden Klammern überein (d.h. da Netz liefert die richtigen Ergebnisse), erfolgt keine Veränderung der Gewichte. haben sie aber verschiedenes Vorzeichen, kann man unterscheiden:

Fall 1: =+1 und  = -1 dann ergibt der Ausdruck in der runden Klammer +2, d.h. für j=1,2

und für j=3

(d.h. gerade b = b + yi in der ursprünglichen Notation)

Fall 2: = -1 und  = +1 dann ergibt der Ausdruck in der runden Klammer -2, d.h. für j=1,2

und für j=3

(d.h. gerade b = b + yi in der ursprünglichen Notation)

Das ergibt gerade den vorgestellten Algorithmus.Anhang:

Dritter Durchlauf (i=1) (-1) ( , also

(i=2) (+1) ( , also

(i=3) (+1) ( , also kein Handlungsbedarf

Vierter Durchlauf (i=1) (-1) ( , also

(i=2) (+1) ( , also kein Handlungsbedarf

(i=3) (+1) ( , also

Fünfter Durchlauf (i=1) (-1) ( , also

(i=2) (+1) ( , also kein Handlungsbedarf

(i=3) (+1) ( , also kein Handlungsbedarf

Mit erhält man:

< und die Dimension des feature spaces ist 6.