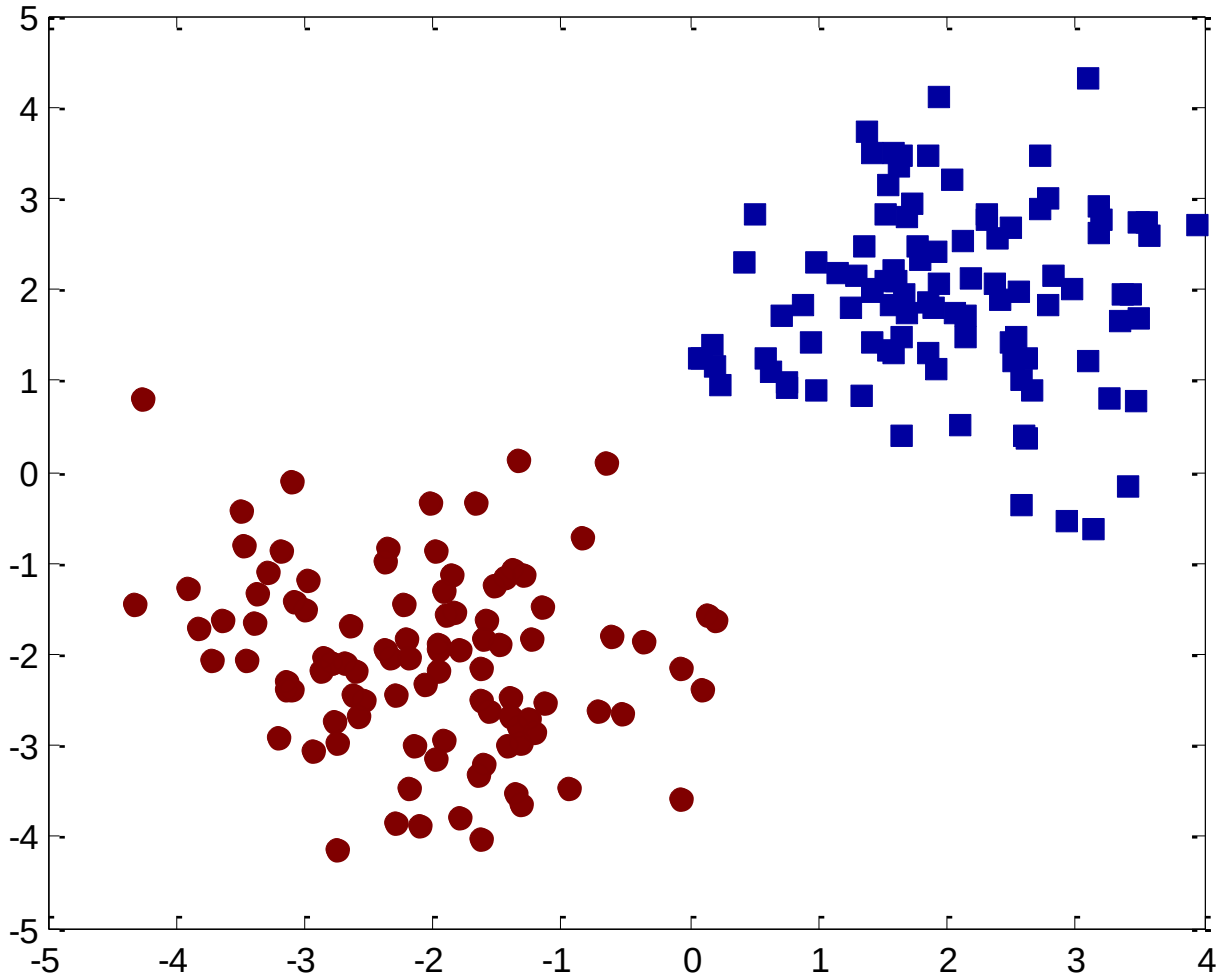
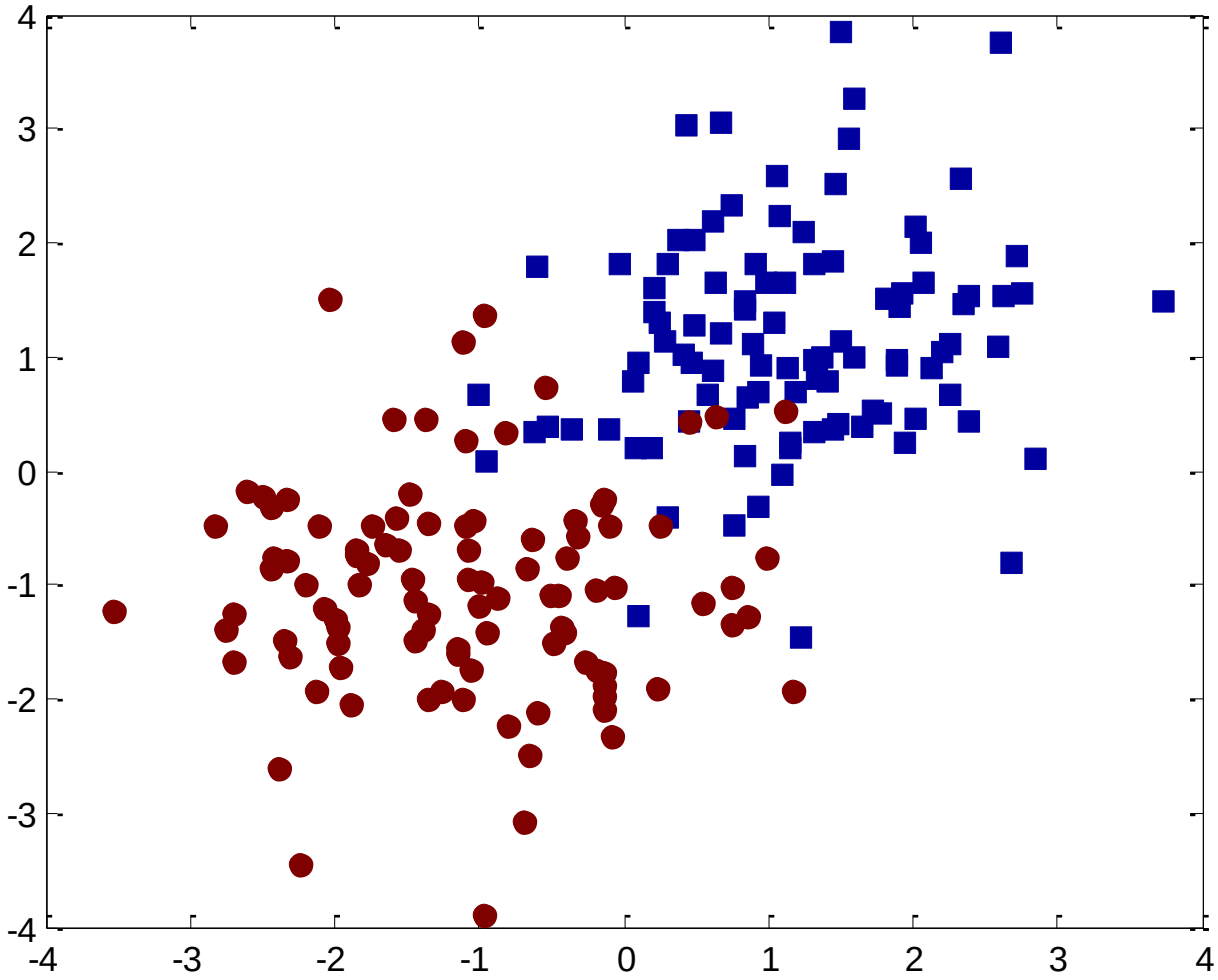
Podstawowe metody uczenia

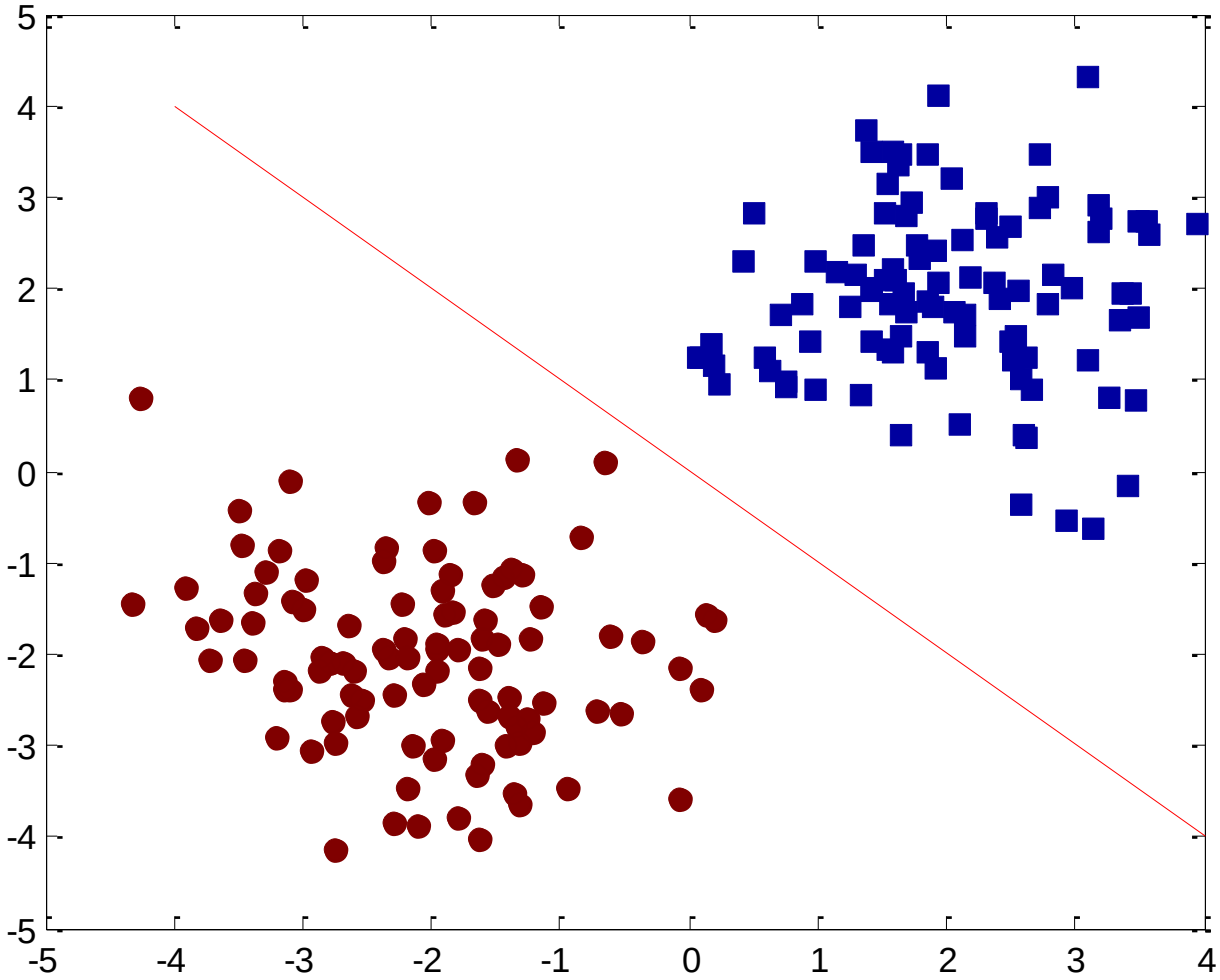
Problem separowalny



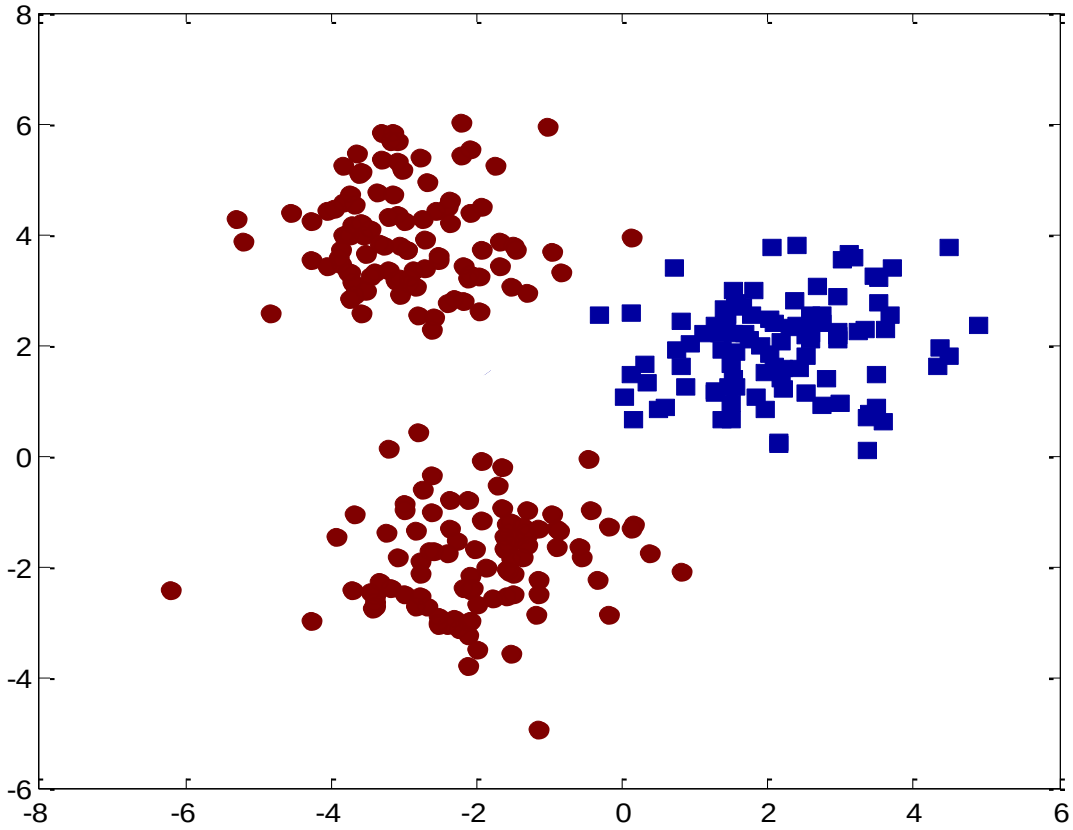
Problem nie separowalny



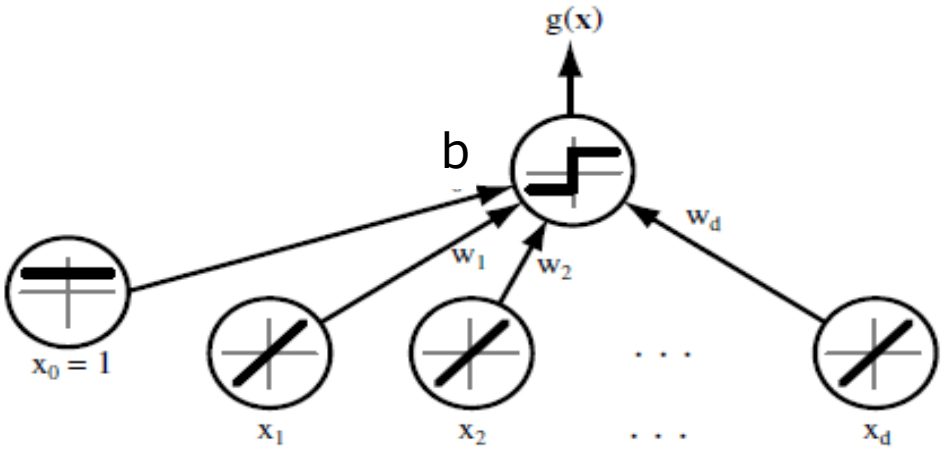
Problem liniowo separowalny



Problem liniowo nie separowalny



Klasyfikator liniowy



- $x_0 - x_n$ – wejścia (ilość analizowanych zmiennych)
- $b, w_1 - w_d$ – wagi neuronu/wagi klasyfikatora
- Funkcja aktywacji typu skok (-1, 1)
- $g(\mathbf{x})$ – wartość wyjściowa
- $\mathbf{x} = [x_0 - x_d]^T$

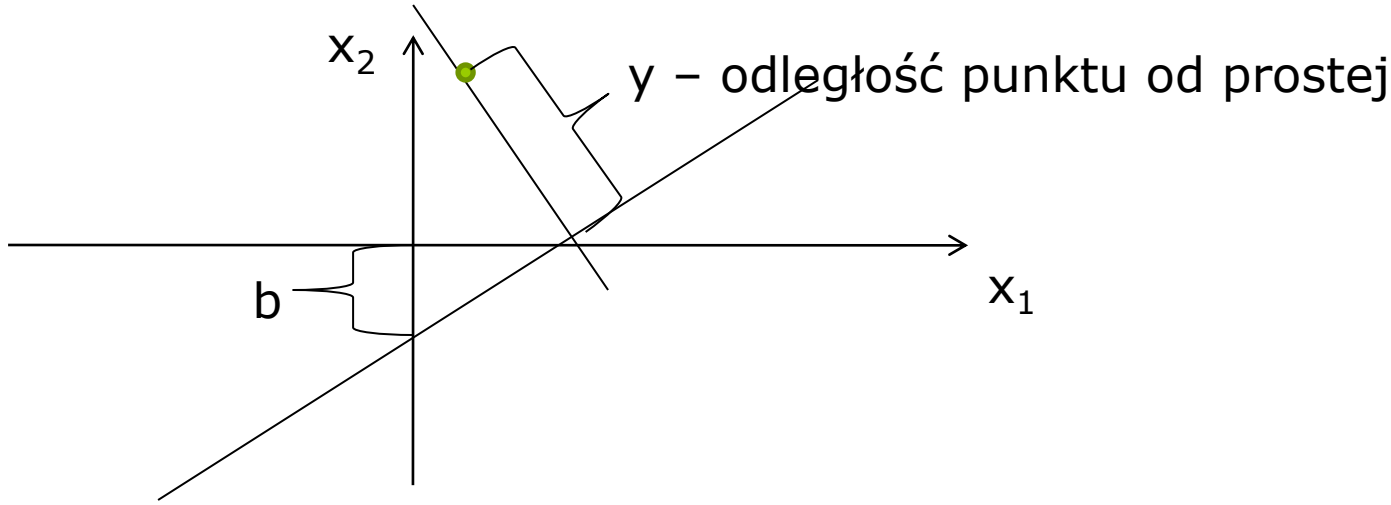
Podstawy algebry

- Równanie prostej

$$y = \mathbf{w}\mathbf{x} + b = \sum_{i=1}^{d} x_i w_i + b$$

- Przykładowa interpretacja

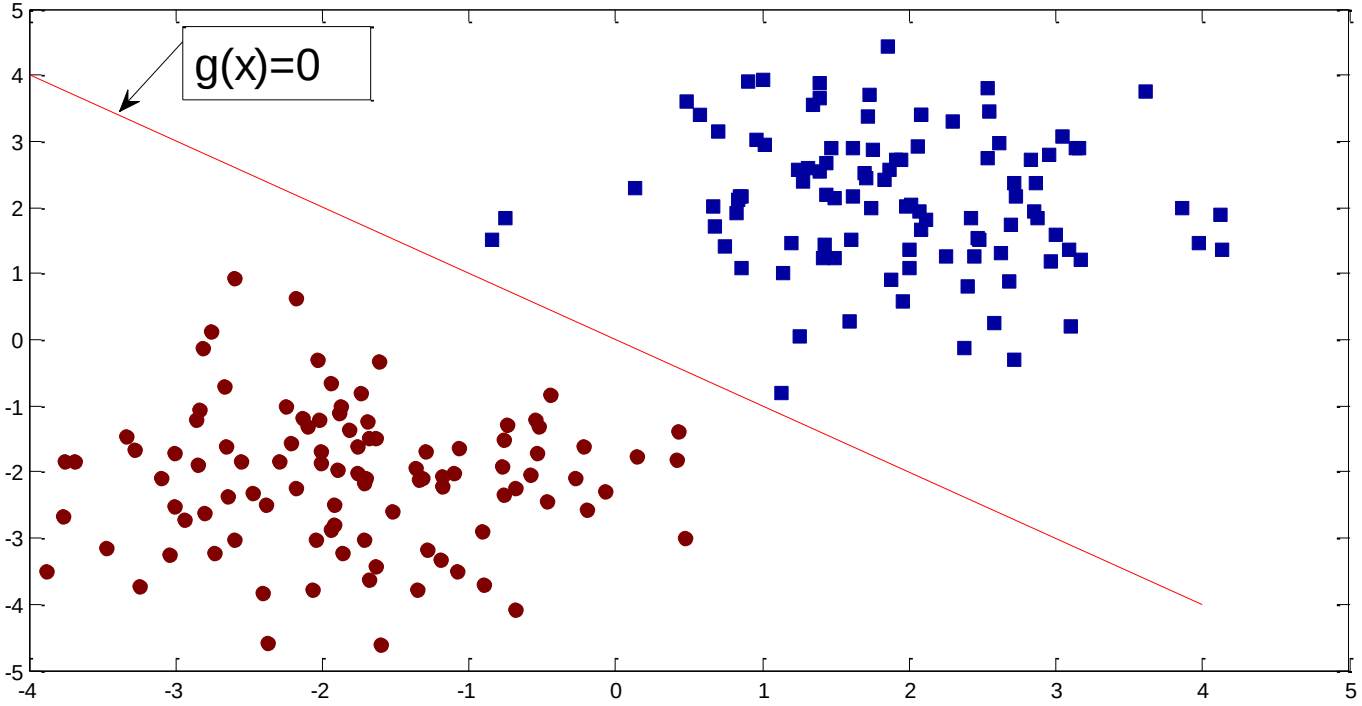
  $y = w_1x_1 + w_2x_2 + b$?



Zadanie – podział na dwie klasy

- g(x) = 0 – granica decyzji
- Szukamy takiej prostej **w** i w_0 która spełnia zależność:

$$\mathbf{w}^T\mathbf{x}^{(1)}+b = \mathbf{w}^T\mathbf{x}^{(2)}+b$$



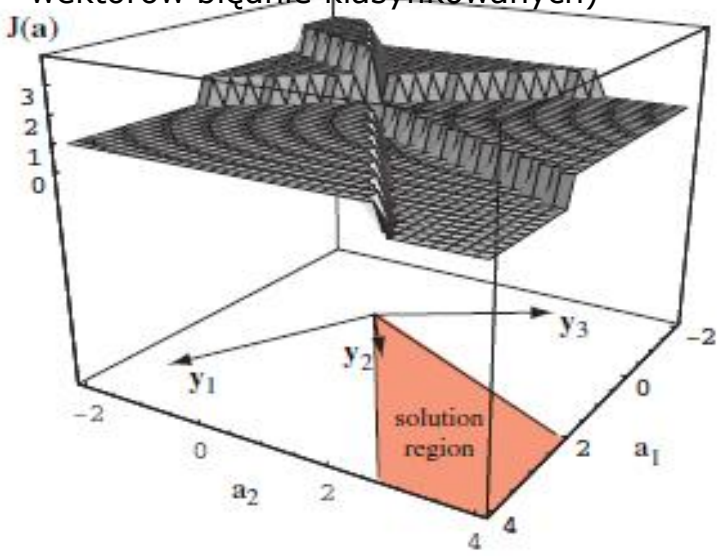
Uczenie sieci/neuronu = problem optymalizacyjny

- Jak znaleźć położenie prostej g(**x**) czyli wyznaczyć parametry wektora **w**?
  - Przez optymalizację
- Znalezienie optymalnej funkcji g(**x**) => problem NP zupełny
- Wykorzystanie metod suboptymalnych
  - Metody gradientowe, największego spadku itp
  - Metody optymalizacji stochastycznej (Monte Carlo, algorytmy genetyczne, ewolucyjne itp)
- Problem => definicja funkcji celu/kosztu którą będziemy optymalizować

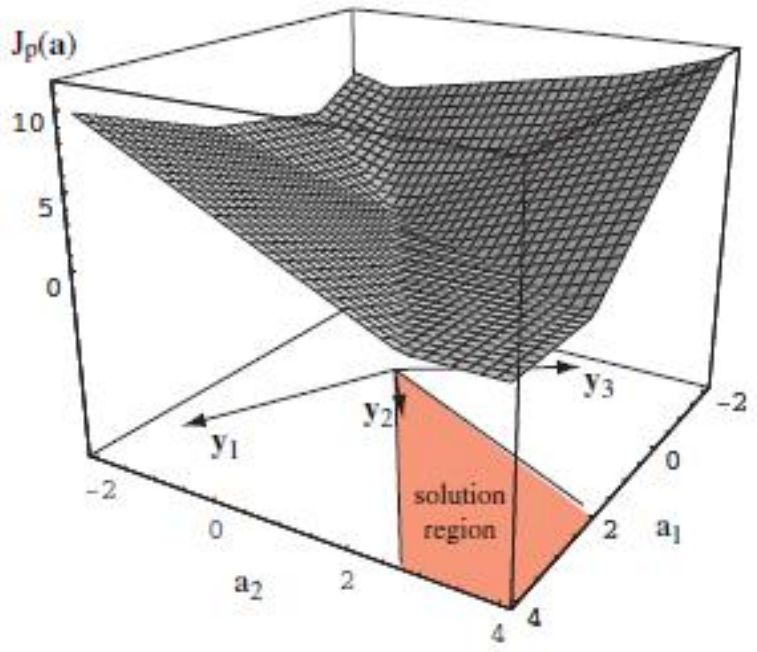
Kształt funkcji kosztu

Problem: płaska postać część funkcji kosztu w części separowalnej

Kształt schodkowej funkcji kosztu (liczba wektorów błędnie klasyfikowanych)



Kształt funkcji kosztu algorytmu Perceptronu



Algorytm Rosenblatta

Gwarantuje zbieżność gdy klasy (-1 i 1) są liniowo separowalne!

Funkcja celu $$J(\mathbf{w}) = \sum_{\mathbf{x}\in\Omega}(-\mathbf{w}^T\mathbf{x})$$

Gdzie $$\mathbf{w}^T\mathbf{x} = w_1x_1 + w_2x_2 + w_3x_3 . + .. + w_nx_n$$

- $\mathbf{w}^t\mathbf{x}$ – iloczyn skalarny wektorów:
- Ω-zbiór wektorów **x** które zostały błędnie sklasyfikowane

Algorytm:

t – numer iteracji, t=0
w – wektor wag
C(**x**) – funkcja zwracająca etykietę klasy dla wektora **x**
m – liczba wektorów uczących
w = generuj losowo
b = wygeneruj losowo
repeat
    t_0=t;
    g = $\mathbf{w}^T\mathbf{x}_i$ +b
    for i=1:m
        if (C($\mathbf{x}_i$)=1) and (g<0) => **w**=**w**+$\mathbf{x}_i$; t=t+1; b=b-g;
        if (C($\mathbf{x}_i$)=-1) and (g>0) => **w**=**w**-$\mathbf{x}_i$; t=t+1; b=b-g;
    end
until t_0=t

Algorytm Rosenblatta

Gwarantuje zbieżność gdy klasy (-1 i 1) są liniowo separowalne!

Funkcja celu $$J(\mathbf{w}) = \sum_{\mathbf{x}\in\Omega} (-\mathbf{w}^T \mathbf{x})$$

Gdzie

- $\mathbf{w}^t\mathbf{x}$ –
- Ω-zbió

$y_n x_n$

Algorytm:

Problem ze zbieżnością,
Działa tylko dla danych liniowo separowalnych

t – numer
w – wektor wag
C(**x**) – funkcja zwracająca etykietę klasy dla wektora **x**
m – liczba wektorów uczących
w = generuj losowo
b = wygeneruj losowo
repeat
    t_0=t;
    g = $\mathbf{w}^T\mathbf{x}_i$+b
    for i=1:m
        if (C($\mathbf{x}_i$)=1) and (g<0) => **w**=**w**+$\mathbf{x}_i$; t=t+1; b=b-g;
        if (C($\mathbf{x}_i$)=-1) and (g>0) => **w**=**w**-$\mathbf{x}_i$; t=t+1; b=b-g;
    end
until t_0=t

Poprawa zbieżności i stabilności

Funkcja kosztu perceptronu:

$$J(\mathbf{w}) = \sum_{\mathbf{x}\in\Omega} (-\mathbf{w}^T \mathbf{x})$$

Algorytm:

t – numer iteracji; t = 0

w – wektor wag

C(**x**) – funkcja zwracająca etykietę klasy dla wektora **x**

m – liczba wektorów uczących

$\eta(j)$ – współczynnik uczenia -> funkcja malejąca w czasie np. $\eta(t) = 1/t+1$

w = generuj losowo;

b = generuj losowo

for t=1:max_iter

g = $\mathbf{w}^T\mathbf{x}_i$ +b

for i=0:m

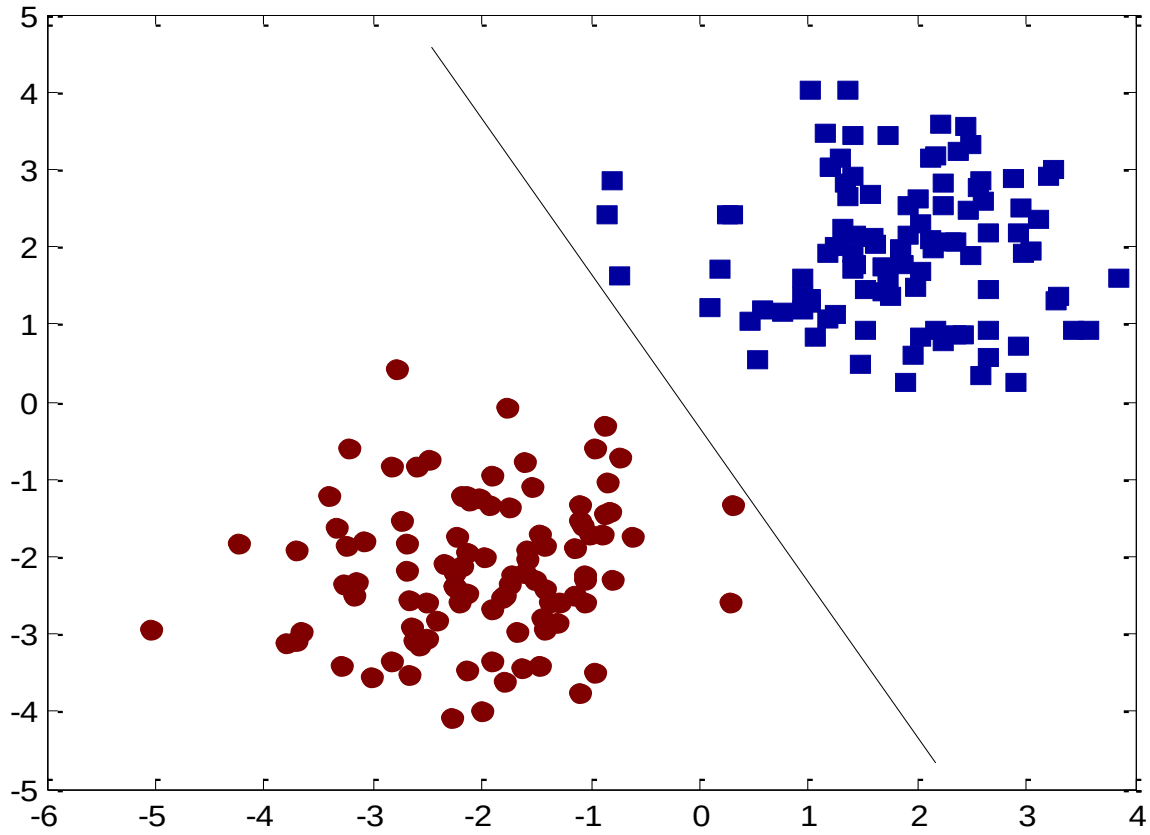
if (C($\mathbf{x}_i$)=1) and (g<0) => **w**=**w**+η(t)$\mathbf{x}_i$; b = b - η(t)g

if (C($\mathbf{x}_i$)=-1) and (g>0) => **w**=**w**-η(t)$\mathbf{x}_i$; b = b - η(t)g

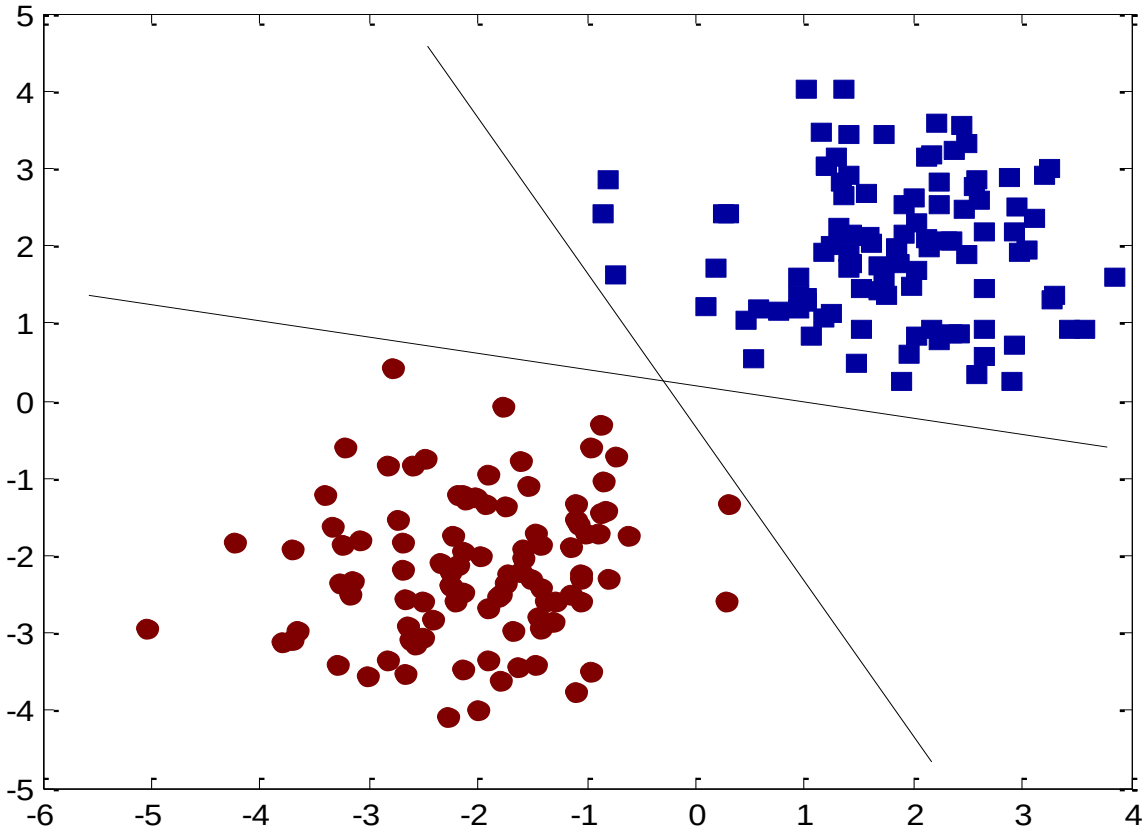
end

end

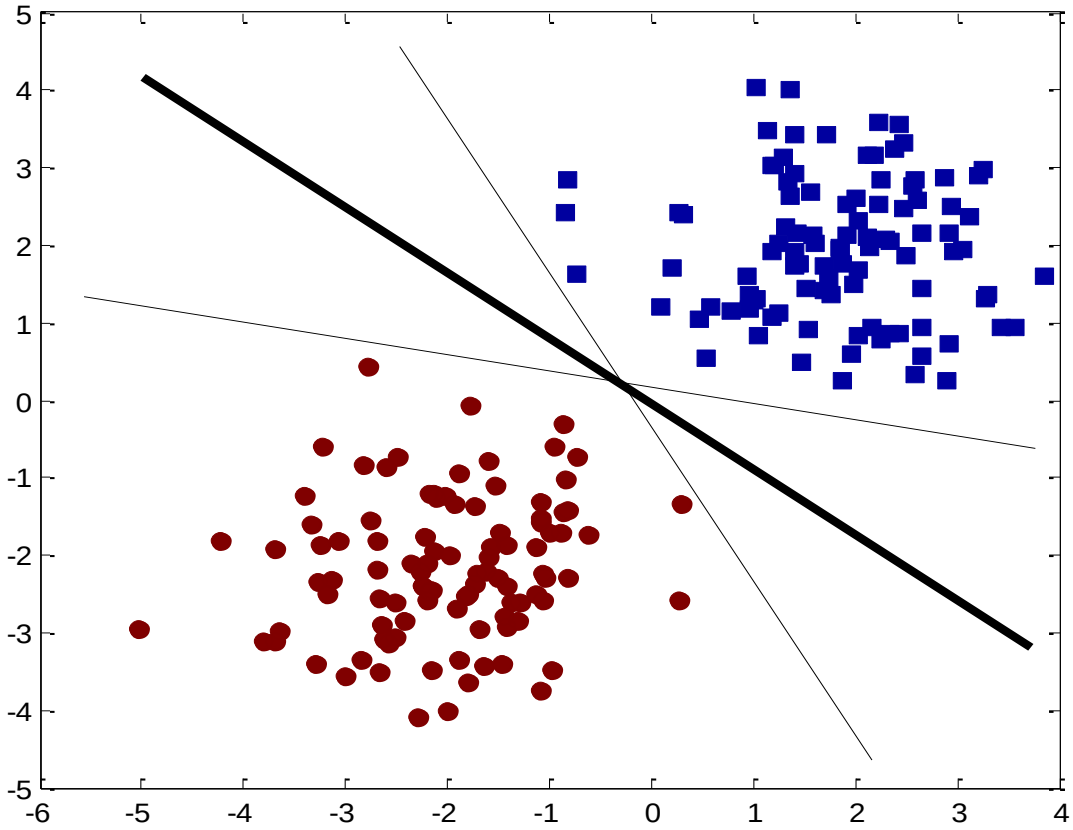
Jaka hiperpłaszczyzna jest lepsza?



Jaka hiperpłaszczyzna jest lepsza?



Jaka hiperpłaszczyzna jest lepsza?



Kolejna modyfikacja algorytm z marginesem

Procedura relaksacji:

nowa funkcja zamiast: $\mathbf{w}^t\mathbf{x} \le 0$ decyzyjna: $\mathbf{w}^t\mathbf{x} \le b$

Funkcja kosztu:

$$J(\mathbf{w}) = \sum_{\mathbf{x}\in\Omega} \frac{\left(\mathbf{w}^T\mathbf{x} - b\right)^2}{\|\mathbf{x}\|^2}$$

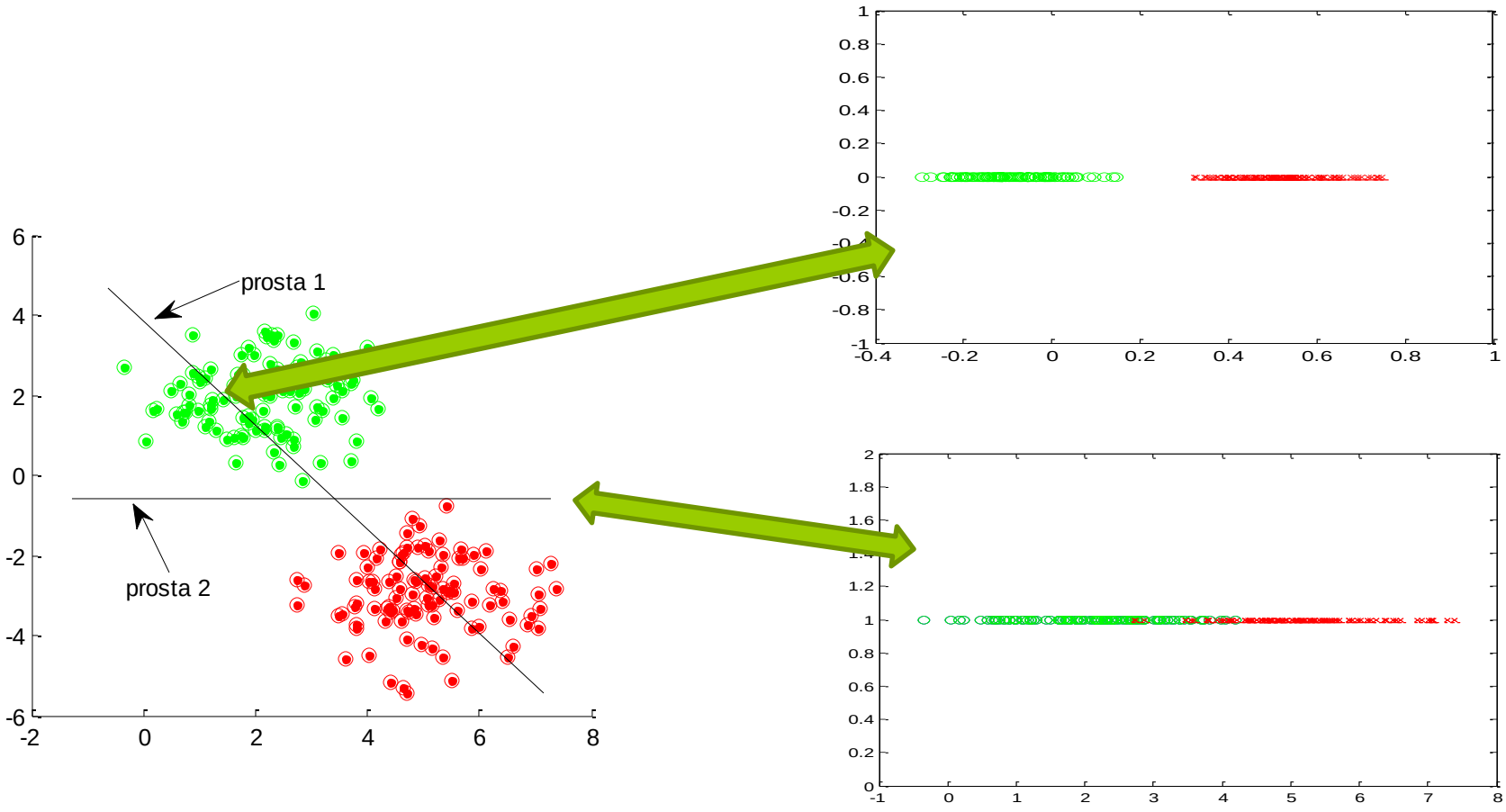
Stąd pochodna:

$$\nabla J(\mathbf{w}) = \sum_{\mathbf{x}\in\Omega} \frac{\mathbf{w}^T\mathbf{x} - b}{\|\mathbf{x}\|^2}\mathbf{x}$$

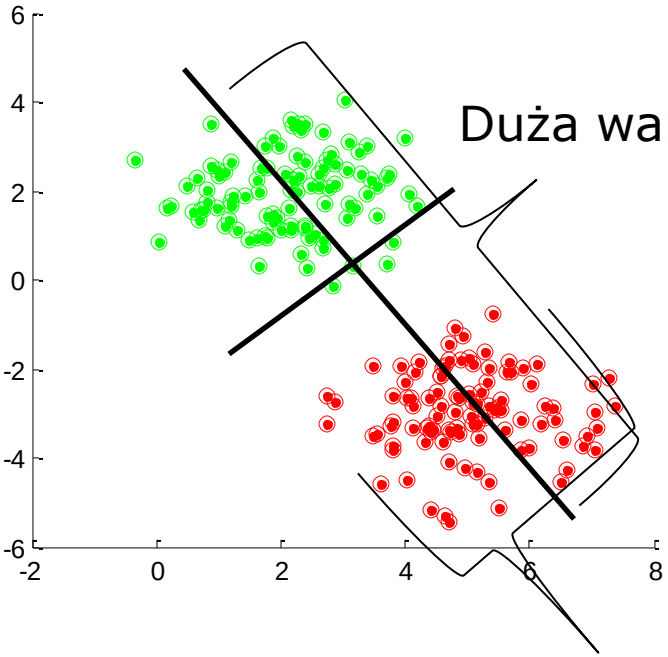
I zasada aktualizacji wartości wag:

$$w^{t+1} = w^t + \eta(t)\sum_{\mathbf{x}\in\Omega} \frac{b - \mathbf{w}^T\mathbf{x}}{\|\mathbf{x}\|^2}\mathbf{x}$$

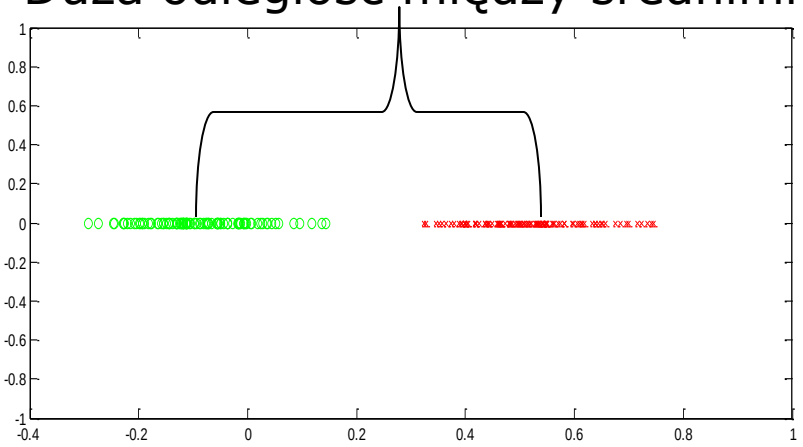
Klasyfikator Fishera

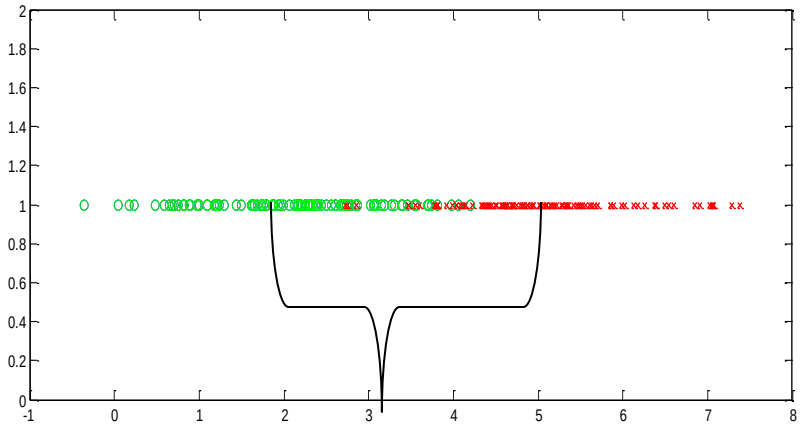


Klasyfikator Fishera



Duża odległość między średnimi





Mała odległość między średnimi

Klasyfikator Fishera

- Założenie – mamy problem dwuklasowy (klasę 0 i 1) o rozkładzie Gaussa, szukamy optymalnej hierpłaszczyzny separującej.
- Wyznaczamy średnie dla obydwu klas $\mu_{y=1}, \mu_{y=-1}$
- Wyznaczamy macierze kowariancji danych:

C

- Fisher funkcję kosztu zdefiniował jako stosunek wariancji pomiędzy klasami w stosunku do wariancji wewnątrz klas

$$S = \frac{\sigma^2_{mięięd\_klasowe}}{\sigma^2_{wewnąewn\underline{n}klasowe}} = \frac{\left(\mathbf{w}\mu_{y=1} - \mathbf{w}\mu_{y=-1}\right)^2}{\mathbf{w}^T\mathbf{C}\mathbf{w} + \mathbf{w}^T\mathbf{C}\mathbf{w}} = \frac{\left(\mathbf{w}\left(\mu_{y=1} - \mu_{y=-1}\right)\right)^2}{2\mathbf{w}^T\mathbf{C}\mathbf{w}}$$

- Gdzie $\vec{w}$– wektor normalny do hiperpłaszczyzny separującej klasy.

Klasyfikator Fishera

- Można pokazać że optymalny $\vec{w}$ można wyznaczyć wg. Zależności:

$$\mathbf{w} = \mathbf{C}^{-1}\left(\mu_{y=1} - \mu_{y=-1}\right)$$

- Wówczas proces decyzyjny:

$$\left|\mathbf{w}\mathbf{x}^T - \mathbf{w}\mu_{y=1}\right| > \left|\mathbf{w}\mathbf{x}^T - \mathbf{w}\mu_{y=-1}\right|$$

- Gdzie **x** – wektor który chcemy sklasyfikować

Lub wyznaczamy *b*

$$b = -\frac{\left(\mu_{y=1} + \mu_{y=-1}\right)\mathbf{w}}{2}$$

Wówczas

$$y = sign\left(\mathbf{w}\mathbf{x} + b\right)$$

Uczenie neuronu poprzez minimalizację błędu średniokwadratowego

- Rozwiązanie problemów nie separowalnych!

  Zapisując:

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,n} \\ x_{2,1} & x_{2,2} & x_{2,n} \\ x_{m,1} & x_{m,2} & x_{m,n} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_n \end{bmatrix} \Leftrightarrow \mathbf{Xw} = \mathbf{y}$$

  Gdzie **X** – jest macierzą reprezentującą cały zbiór uczący o *m* wektorach, każdy *n* elementowy

Zapisując błąd jako **e**=**Xw**-**y** wówczas funkcja kosztu przyjmuje postać:

$$J(\mathbf{x}) = \left(\mathbf{w}^T\mathbf{x} - \mathbf{y}\right) = \sum_{i=1}^{m}\left(\mathbf{w}^T\mathbf{x} - y\right)^2$$

Stąd pochodna:

$$\nabla J = \sum_{i=1}^{m} 2\left(\mathbf{w}^T\mathbf{x}_i - \mathbf{y}_i\right)\mathbf{x}_i = 2\mathbf{X}^T\left(\mathbf{Xw} - \mathbf{Y}\right)$$

I ostatecznie:

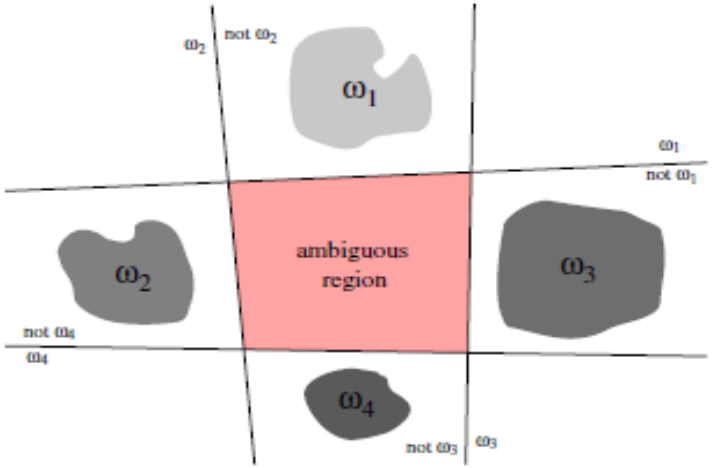
$$\mathbf{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{d}$$

To tyle w tym temacie

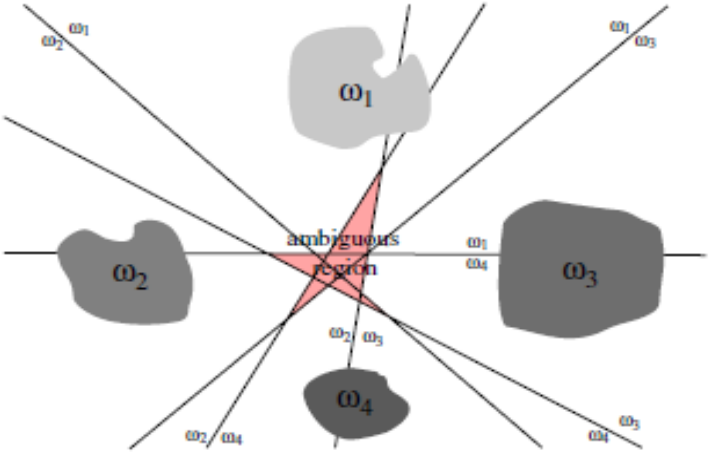
Krótko o problemie systemów wieloklasowych

Od klasyfikatora dwuklasowego do wieloklasowego

- Klasa – reszta



- Klasa - klasa



(*) Duda, Hart, Stork, Pattern Classification ->

Klasyfikacja wieloklasowa - porównanie

- Dwie koncepcje:
  - Klasa – klasa
    Problem skalowalności – mając *k* klas potrzebujemy $(k^2-k)/2$ klasyfikatorów, czyli liczba klasyfikatorów rośnie z kwadratem liczby klas ☹, ale jeśli rozmiar zbioru jest *m* i mamy równą ilość przypadków w danej klasie a złożoność uczenia algorytmu jest *kwadratowa* to uczenie jednego modelu ma złożoność $(2m/k)^2$ mamy więc złożoność całkowitą
    $(k^2-k)4m^2/(2k^2)=2(1-1/k)m^2$ ☺
  - Klasa – reszta
    Potrzebujemy *k* klasyfikatorów – liniowa zależność w funkcji liczby klasyfikatorów ☺
    ale tutaj całkowita złożoność jest:
    km^2 ☹