Maszyny wektorów nośnych

Lato 2025

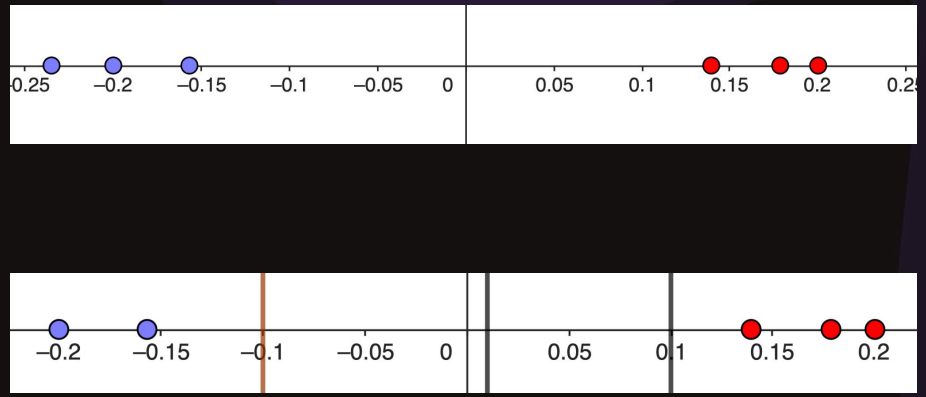
GAUSS

Wstęp - podstawowa idea

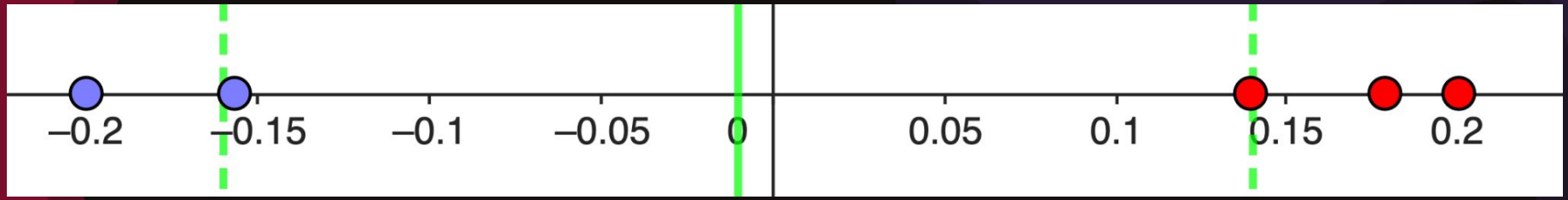
Załóżmy, że:

- Chcemy dokonać podziału dwóch klas w zależności od pewnej zmiennej
- Możemy dokonać tego podziału za pomocą hiperpłaszczyzny (w przykładzie mamy linie w celu uproszczenia wizualizacji).

Jak zrobić to najoptymalniej?



Odpowiedź:



Takie dopasowanie jest najbardziej optymalne dla naszych danych, ponieważ **maksymalizuje margines** (przerywane linie). Co w naszym przypadku gwarantuje najbardziej rzetelną klasyfikację. Zwróćmy uwagę na symetrię punktów granicznych względem linii ciągłej.

$$w^\top x + b = 0 \tag{1}$$

Zdefiniujmy hiperpłaszczyznę wzorem (1) gdzie w to wektor normalny do płaszczyzny, x to dowolny punkt w przestrzeni, b to wyraz wolny (bias), a y to etykieta klasy (w ramach konwencji -1 i 1).

$$y^{(i)}(w^\top x^{(i)} + b) \geq 1, \quad y \in \{-1, 1\} \tag{2}$$

Wtedy warunkiem przypisania elementu do odpowiedniej klasy jest równanie (2), co w konsekwencji daje odpowiednie wzory na hiperpłaszczyzny marginesów:

$$w^\top x + b = 1, \tag{3}$$

$$w^\top x + b = -1. \tag{4}$$

Przypominając sobie wzór na odległość punktu od płaszczyzny i wzór na odległość dwóch prostopadłych płaszczyzn o tym samym wektorze normalnym:

$$d(x, H) = \frac{|w^\top x + b|}{||w||}, \tag{5}$$

$$d(H_1, H_2) = \frac{|b_1 - b_2|}{||w||}, \tag{6}$$

możemy wyliczyć wzór na szerokość marginesu

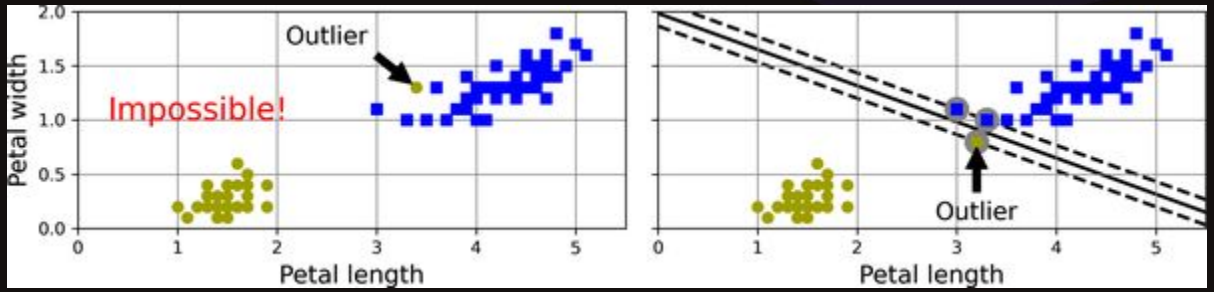
$$d = \frac{|1 - (-1)|}{||w||} = \frac{2}{||w||}. \tag{7}$$

Oznacza to, że naszym zadaniem w maksymalizowaniu szeokości marginesu jest

$$\min_{w,b} \frac{1}{2}||w||^2.$$

Oczywisty problem

Co się dzieje, kiedy w danych pojawi się błąd, albo podział nie będzie idealnie liniowy?



Pojawiają się dwie opcje:

- Dopuszczenie pewnej niedokładności modelu - soft margin,
- Zmniejszenie marginesu - hard margin (po prawej).

Hard margin

- Zakłada, że dane SĄ liniowo separowalne,
- Maksymalizuje margines bez dopuszczenia błędnych klasyfikacji,
- Jest rzadko używany w praktyce, ponieważ dane mogą zawierać szum, nakładać się na siebie

Soft margin

-Pozwala na pewne błędne klasyfikacje, aby dostosować się do rzeczywistych danych.
- Dodaje parametr regularyzacji C, który kontroluje kompromis między szerokością marginesu a liczbą błędnie sklasyfikowanych punktów:

- Duże C → model stara się minimalizować błędy (mniejszy margines, większe ryzyko przeuczenia).

- Małe C → model dopuszcza więcej błędów, ale maksymalizuje margines (lepsza generalizacja).

Kiedy mowa o mękkim marginesie w grę wchodzi dodatkowa zmienna tzw. "slackowa" oznaczana grecką literą "xi". A nasza funkcja celu przyjmuje następującą formę:

$$f(x) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i \tag{8}$$

Gdzie "xi" definiuje się następująco:

1. Jeśli punkt x_i jest poprawnie sklasyfikowany i leży po odpowiedniej stronie hiperpłaszczyzny (nie przekracza marginesu), to

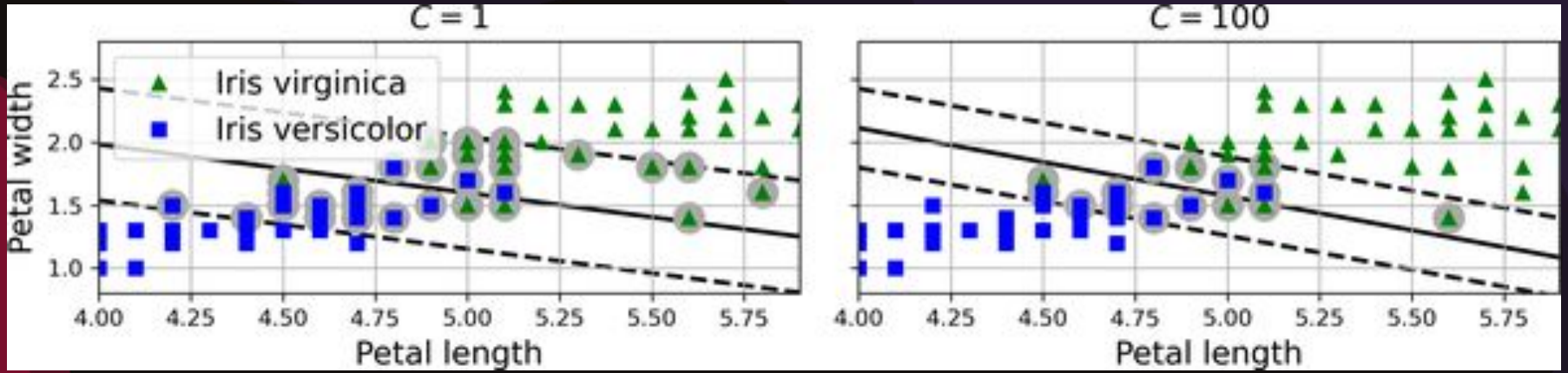
   $$y_i(w^\top x_i + b) \geq 1 - \xi_i$$

   i wtedy $\xi_i = 0$

2. Jeśli punkt x_i jest błędnie sklasyfikowany, (jest po złej stronie hiperpłszczyzny lub narusza margines) to przy równaniu

   $$y_i(w^\top x_i + b) \leq 1 - \xi_i$$

   $\xi_i > 0$ - im większy błąd klasyfikacji, tym większa zmienna.

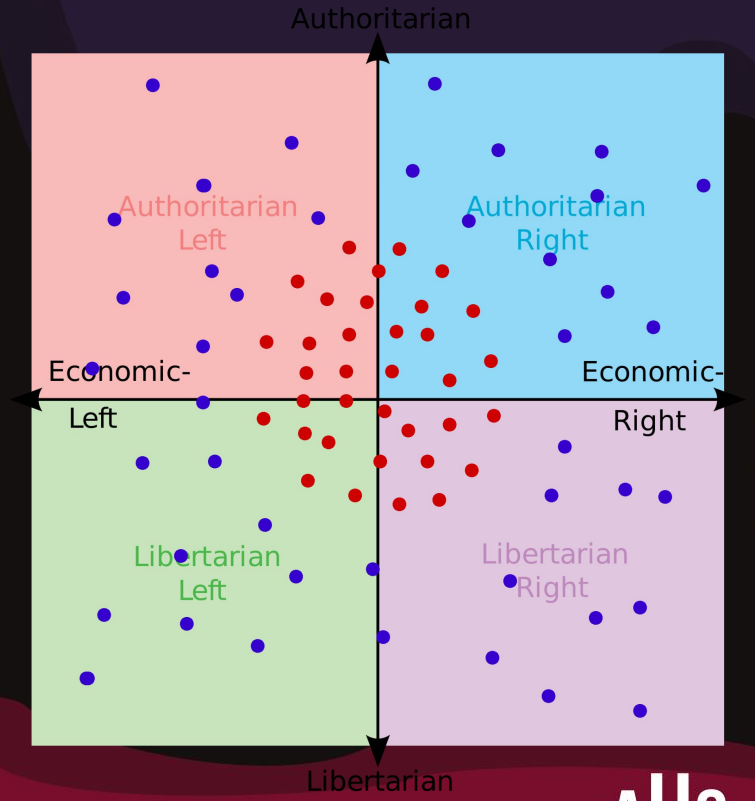


Różne wartości parametru C na przykładzie danych z datasetu iris z pakietu scikit-learn.

Co jeśli dane są zupełnie “nieliniowe”?

Załóżmy, że mamy do dyspozycji kompas polityczny - dwie zmienne (poglądy ekonomiczne i poglądy społeczne) oraz za zadanie odróżnić osoby na ogół zadowolone z panującego ustroju (na czerwono) oraz osoby oczekujące gruntownych przemian (na niebiesko).

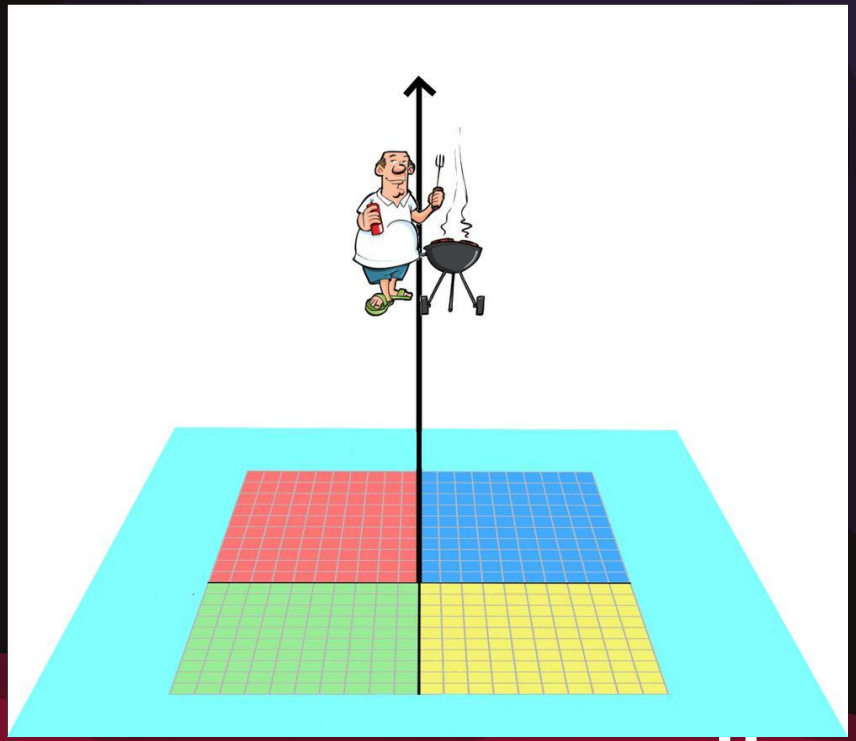
UWAGA: Zarówno kompas polityczny jak i prezentacja nie przekazują rzetelnej wiedzy z zakresu polityki i należy je traktować jako ciekawostkę (przynajmniej w tym zakresie).

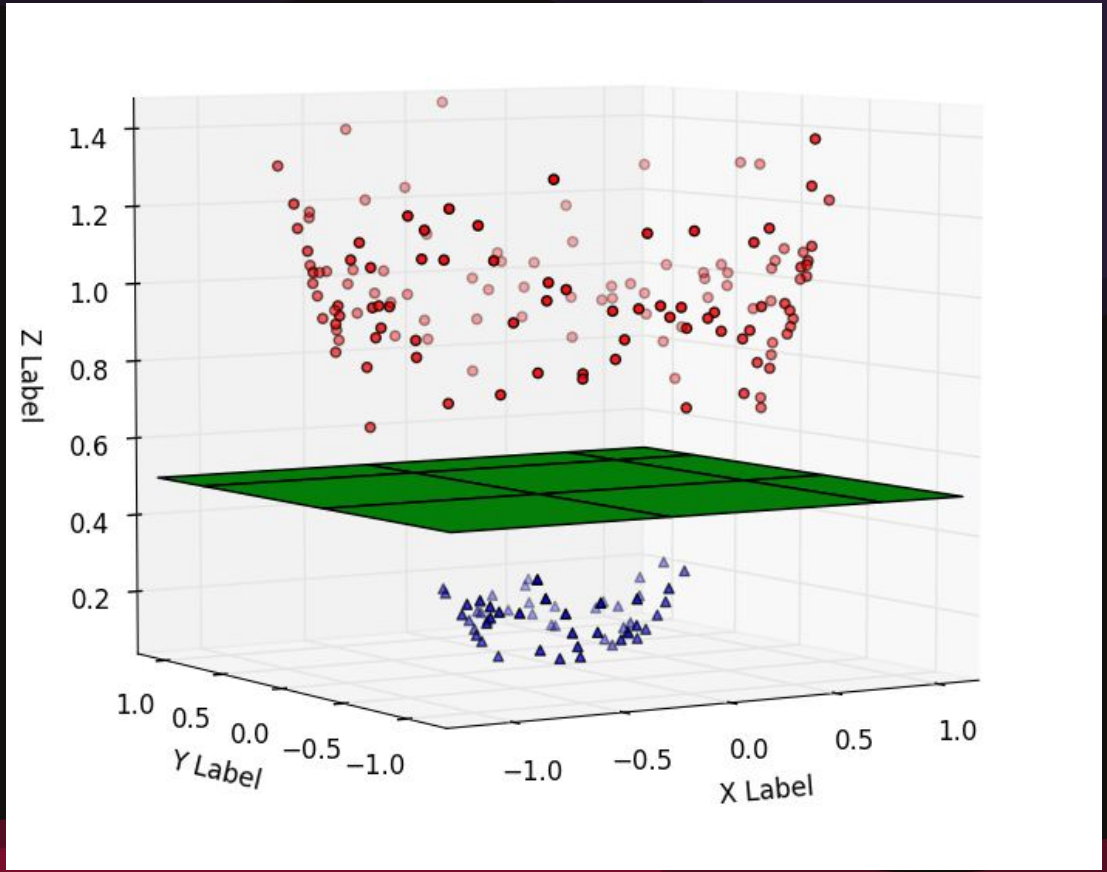


Oś grillowa, czyli inżynieria cech

Jednocześnie, prostym i trudnym do wymyślenia rozwiązaniem jest dodanie 3. osi, która będzie jakąś formą odwzorowania naszych wartości x i y na zbiór liczb rzeczywistych. Zobaczmy jak wygląda podobny do poprzedniego przykład po zastosowaniu

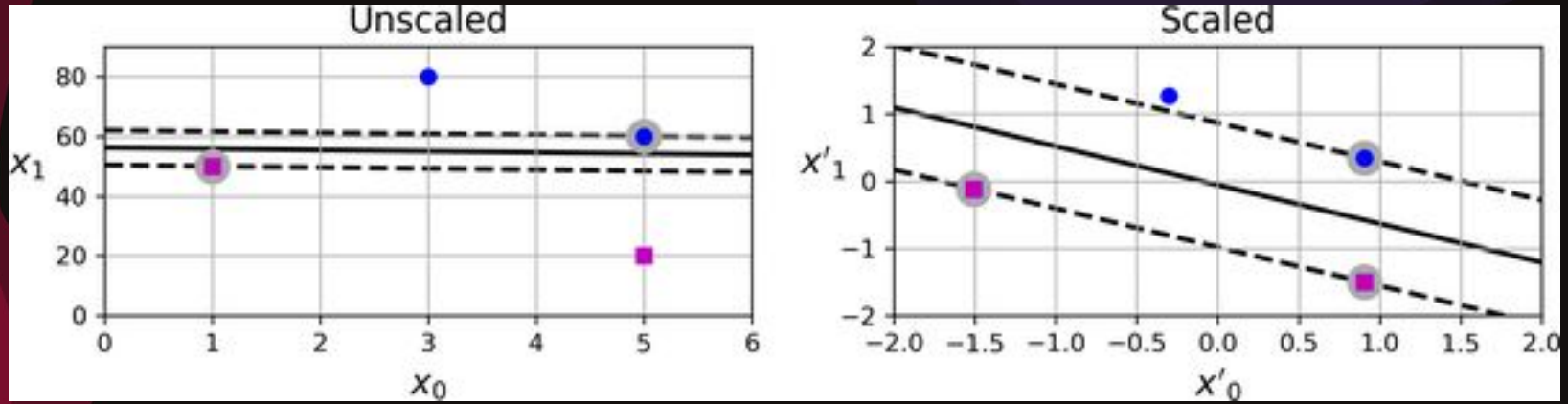
$z=x^2 +y^2$.




1.4
1.2
1.0
0.8
0.6
0.4
0.2
Z Label
1.0
0.5
0.0
−0.5
−1.0
Y Label
−1.0
−0.5
0.0
0.5
1.0
X Label

GAUSS

Przestroga - s(z)kalujcie swoje dane!



Działanie algorytmu jest wrażliwe na nieprzeskalowane dane, zbyt duże wartości mogą zdominować zbiór.

Kernel trick - czyli inżynieria cech bez inżynierii cech

Tzw. “sztuczka z jądrem” polega na zmianie sposobu w jaki liczymy iloczyn skalarny zamiast faktycznie zmieniać strukturę danych. Dzięki temu można np. policzyć iloczyn skalarny taki jak dla przestrzeni sześciowymiarowej mając do dyspozycji tylko przestrzeń dwuwymiarową. Algorytm prezentuje się następująco:

1. Za pomocą wybranego rodzaju jądra liczymy “podobieństwo” każdej próbki do każdej w zbiorze, tworzymy w ten sposób nową macierz próbek K.
2. Do wytrenowania algorytmu wykorzystujemy macierz K, a nie początkową.
3. Model rozwiązuje problem optymalizacyjny.

Dokładny opis procesu za chwilę.

Najważniejsze rodzaje jąder:

- jądro liniowe (domyślne)

  $K(x, x') = x^\top x'$ musiałem tu coś napisać bo z jakiegoś powodu LaTeX nie wyrównuje równania do lev

- jądro wielomianowe

  $K(x, x') = (x^\top x' + c)^d$, gdzie c to przesunięcie, a d to stopień wielomianu,

- jądro gausowskie/RBF

  $K(x, x') = exp(-\gamma||x - x'||^2)$, gdzie $\gamma = \frac{1}{2\sigma^2}$, a σ to odchylenie standardowe dla krzywej gaussa,

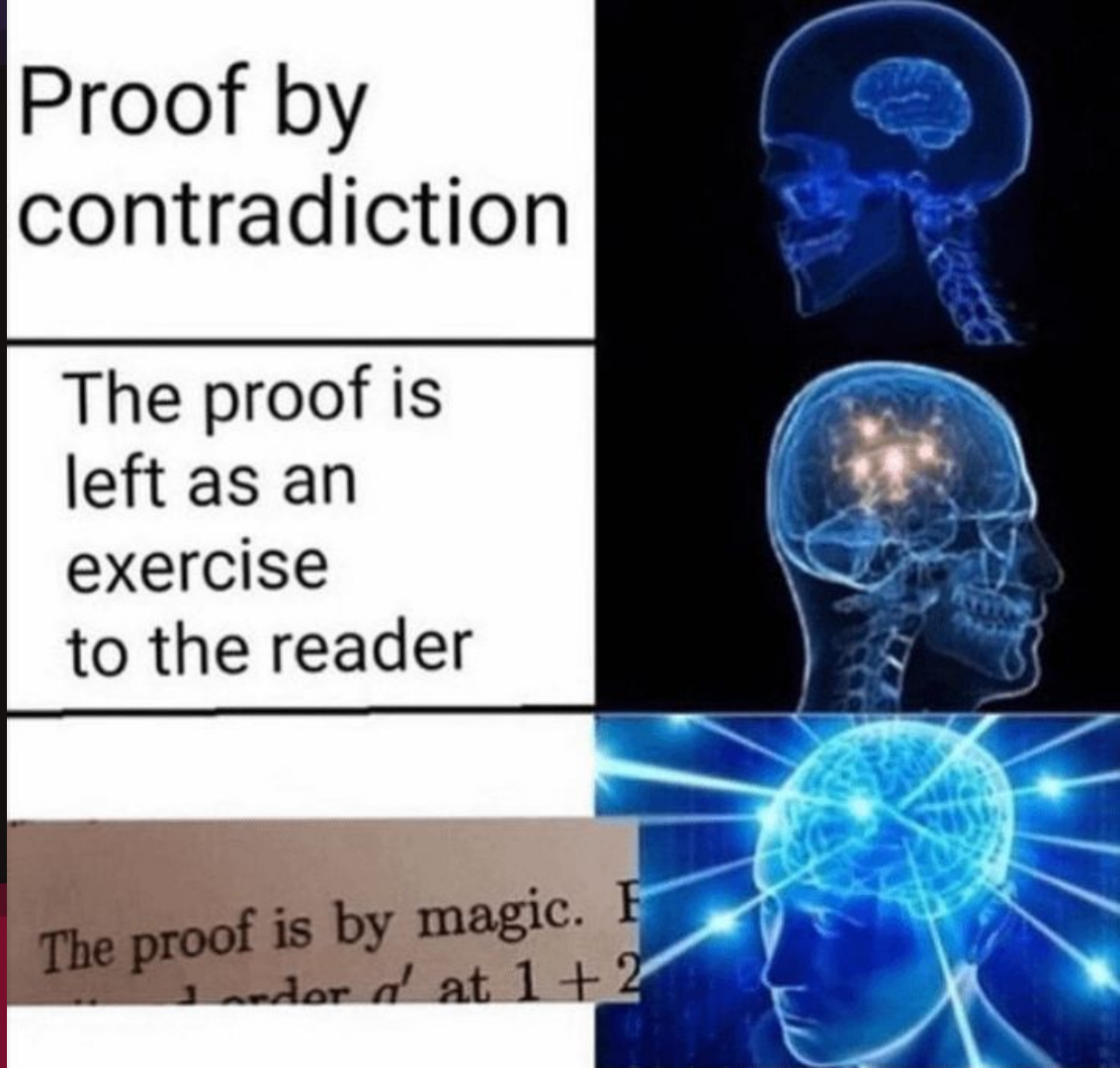
- jądro sigmoidalne

  $K(x, x') = tanh(\alpha x^\top x' + c)$, gdzie α to parametr "wzmocnienia" , a c to przesunięcie.

Dlaczego te wzory działają?

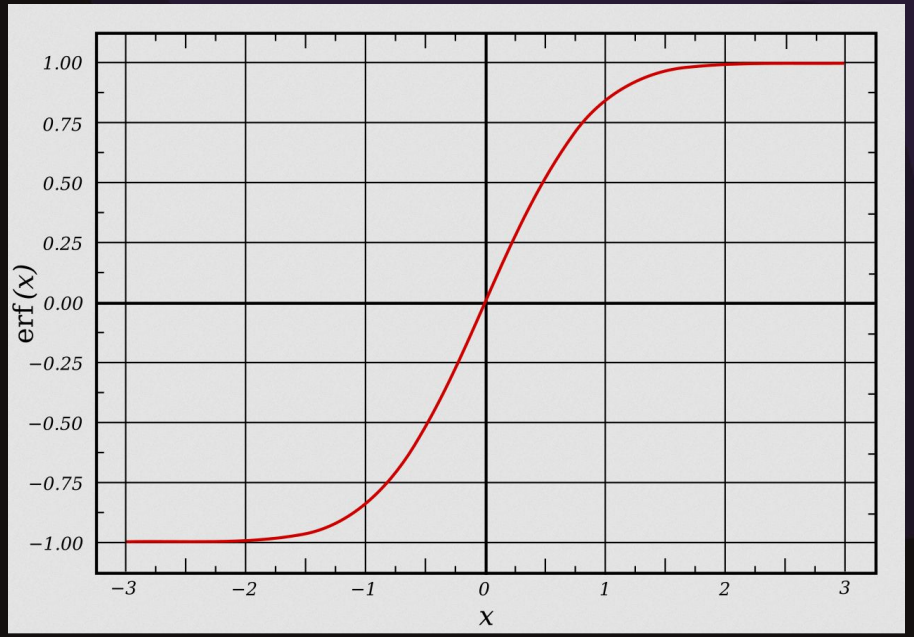
Pomimo tego, że funkcje podobieństwa nie są w sposób dosłowny iloczynami skalarnymi, symulują iloczyn skalarny dla wektorów poddanych pewnemu przekształceniu który podnosi ich wymiar tworząc przestrzeń ukrytą (feature space).

Dla zainteresowanych - Twierdzenie Mercera, przestrzenie Hilberta, psychoterapia.

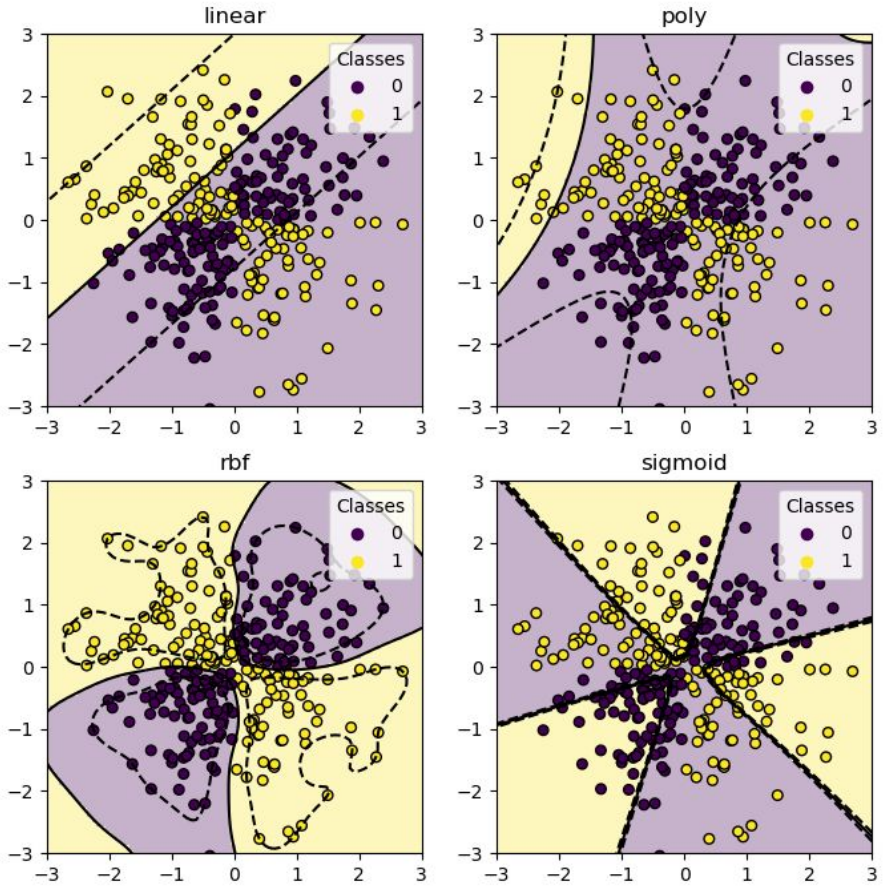


Intuicje stojące za poszczególnymi jądrami

- Liniowe - prosty pomiar odległości i strony po której znajduje się punkt względem hiperpłaszczyzny,
- Wielomianowe - symuluje iloczyn skalarny dla hiperparametrów będących wyrazami wielomianu o stopniu wybranym przy tunowaniu modelu,
- Sigmoidalne - symuluje aktywację neuronu: 1 dla wartości podobnych, -1 dla różnych (ujemny iloczyn skalarny) i wartości bliskie zeru dla wartości ortogonalnych (niezależnych).
- Gaussa - wykorzystuje wzór na krzywą Gaussa jako funkcję podobieństwa, każdy punkt tworzy miniaturową krzywą w swojej okolicy, która składa się na granicę decyzyjną.



Funkcja sigmoidalna (logistyczna)


linear
poly
rbf
sigmoid
Classes
0
1
−3
−2
−1
0
1
2
3

Załóżmy, że mamy macierz cech $M_{n \times d}$, gdzie n to liczba próbek, a d to liczba cech. Pierwszym krokiem będzie obliczenie wcześniej wspomnianej nowej macierzy jądra K, gdzie kolumny są wyliczonymi (przy użyciu d cech) podobieństwami danego punktu do wszystkich punktów datasetu. Nowa macierz ma wymiary $n \times n$, złożoność obliczeniowa tego procesu to $O(d \cdot n^2)$

Dla nowej macierzy tworzy się nowe zagadnienie optymalizacyjne, aby uniknąć konieczności liczenia zwykłego iloczynu skalarnego, ponieważ to wymagałoby faktycznego przekształcenia danych, co byłoby zdecydowanie zbyt kosztowne. Wzór na nową funkcję celu prezentuje się następująco:

$$min_{\alpha} \frac{1}{2} \alpha_i \alpha_j y_i y_j k(x_i, x_j) - \sum_{i=1}^{n} \alpha_i$$

gdzie:

- α_i to zmienne dualne, które są parametrami modelu,
- y_i to etykiety klasy dla próbek,
- $k(x_i, x_j)$ to funkcja jądra,
- $\sum_{i=1}^{n}$ to suma zmiennych dualnych.

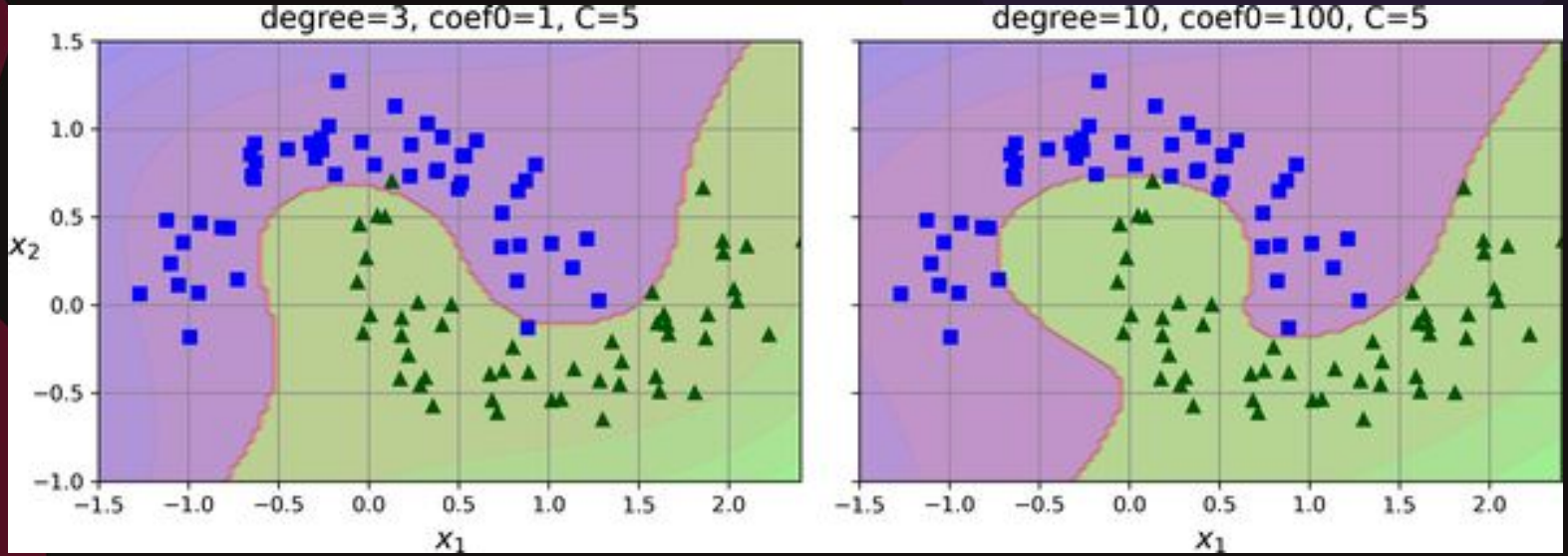
Złożoność obliczeniowa problemu optymalizacyjnego to $O(n^3)$, zatem ogólna złożoność trenowania modelu z użyciem jądra to $O(d \cdot n^2 + n^3)$, gdzie złożoność dla jądra liniowego mieści się między $O(nd)$ i $O(n^2 d)$.

Przewidywania odbywają się według wzoru

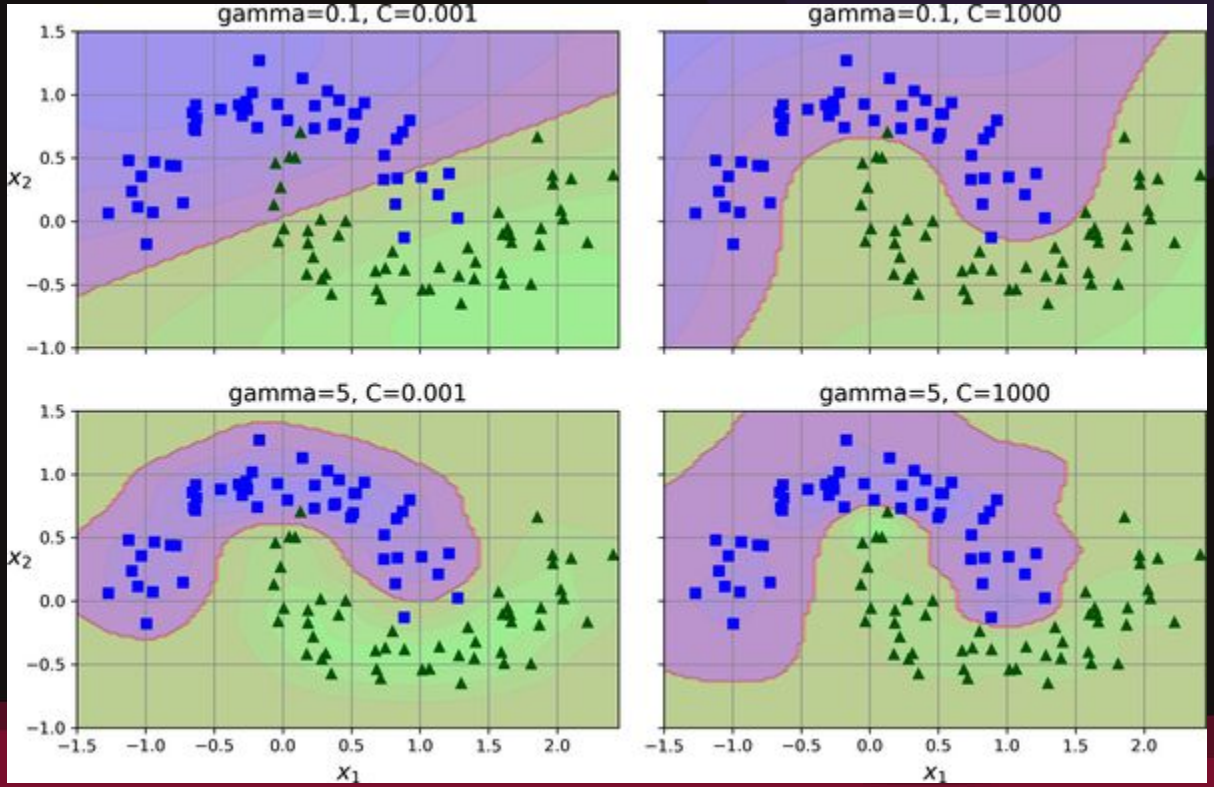
$$f(x) = \sum_{i \in SV} \alpha_i y_i K(x, x_i) + b$$

w zależności od $sgn(f(x))$ przypisujemy x do klasy 1 lub -1, jeśli funkcja jest równa zero (ekstremalnie rzadkie) punkt leży na granicy decyzyjnej (odpowiednik hiperpłaszczyzny) i jest uznany za punkt niepewny.

Różne wartości parametrów dla jądra wielomianowego



Różne wartości parametrów dla jądra RBF



Klasyfikacja wieloklasowa

Z założenia SVM jest modelem obsługującym jedynie klasyfikację binarną, aczkolwiek nie znaczy to, że nie możemy go zastosować do klasyfikacji wieloklasowej. Do wyboru mamy dwa wyjścia:

- One versus all - Trenujemy oddzielne modele rozpoznający, czy dany punkt należy do odpowiedniej klasy, wygrywa najwyższy wynik spośród uzyskanych spośród modeli.
- One vs one - Trenujemy modele dla każdej kombinacji dwóch cech, wygrywa ten, który został przyporządkowany najwięcej razy.

OvA

Zalety:

- Prosty i szybki.

Wady:

- Nierównowaga klas (np. klasyfikator dla rzadkiej klasy vs. reszta).
- Możliwe konflikty (kilka klasyfikatorów daje wysokie wartości).

OvO

Zalety:

- Mniejsze podzbiory treningowe → szybsze trenowanie pojedynczych modeli.
- Lepsze przy zrównoważonych klasach.

Wady:

- Dużo klasyfikatorów — dla 10 klas → 45 modeli.

W przypadku regresji z użyciem SVM zmieniają się nieco założenia:

Szukamy funkcji postaci $f(x) = w^\top x + b$

Chcemy, by większość punktów spełniała równanie

$$|y_i - f(x_i)| \leq \epsilon$$

Punkty, które przekraczają ten margines, wprowadzają straty kontrolowane przez zmienne slackowe;

ξ_i, jeśli punkt leży powyżej marginesu
ξ_i^*, jeśli punkt leży poniżej.

Funkcja celu prezentuje się następująco

$$min_{w,b,\xi,\xi^*} \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

z warunkami

$$\begin{cases} y_i - \mathbf{w}^\top \mathbf{x}_i - b \leq \varepsilon + \xi_i \\ \mathbf{w}^\top \mathbf{x}_i + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases}$$

Regresja z użyciem SVM

Dziękuję za uwagę!

Źródła:

https://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html

Géron, A. (2022). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems (3rd ed.). O'Reilly Media.

Wilmott, P. (2019). Machine learning: An applied mathematics introduction. Wilmott Press.