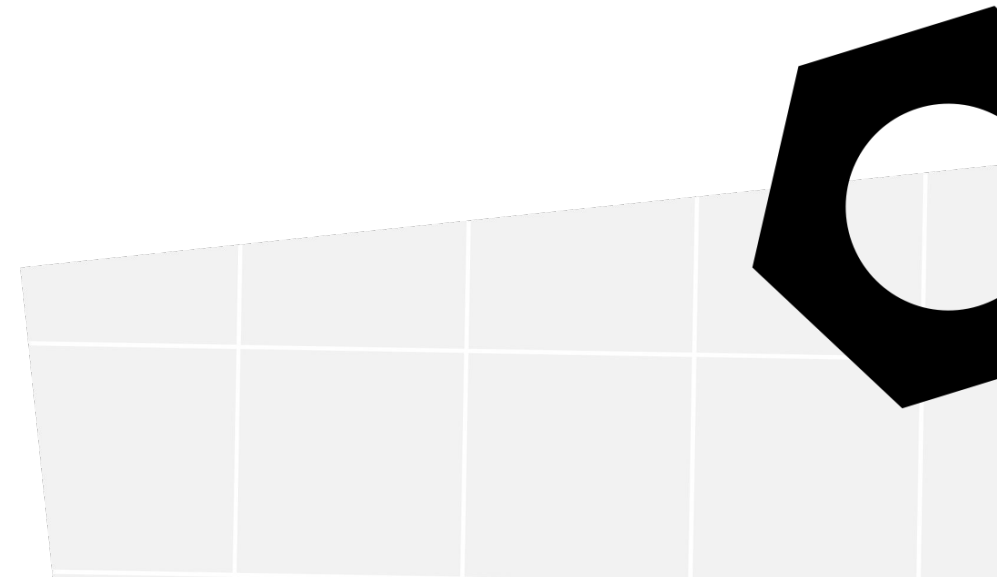




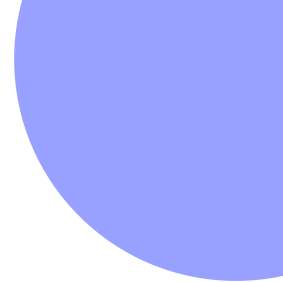
# **Machine Learning w praktyce: problemy klasyfikacji w uczeniu nadzorowanym**

**Kurs Data Science**

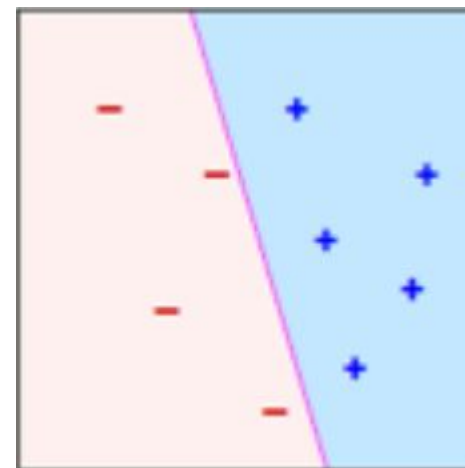
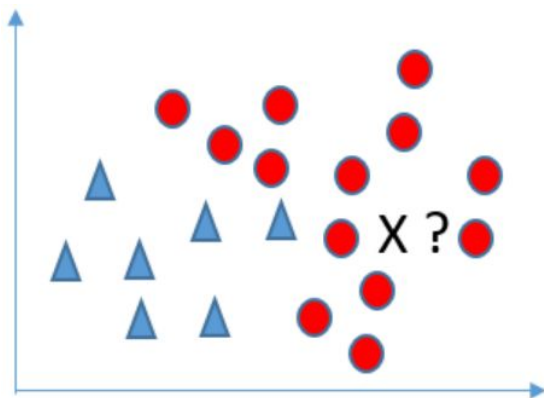




# Uczenie nadzorowane

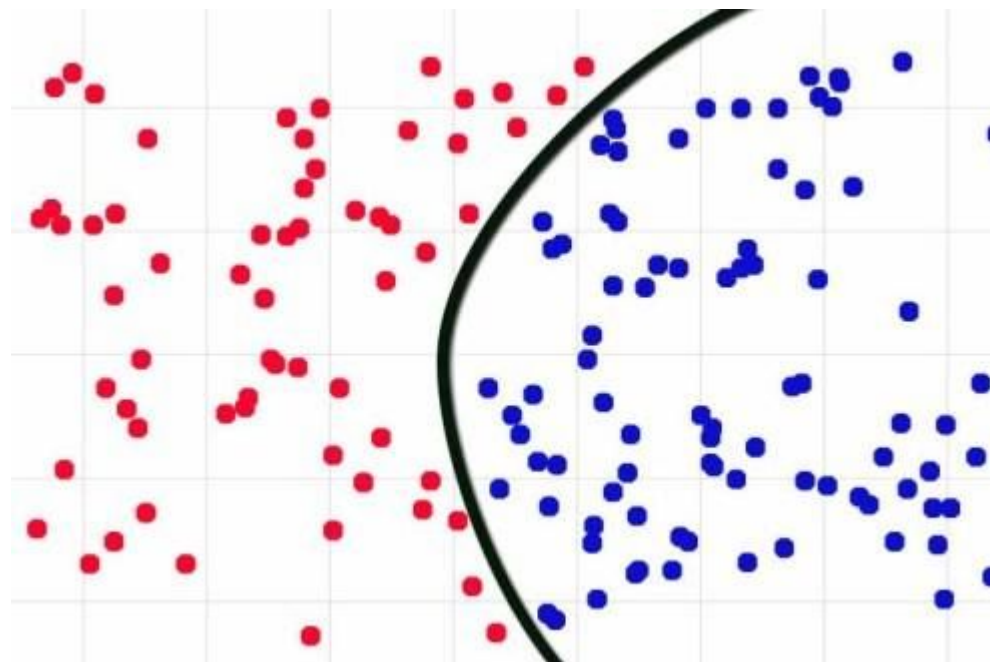


Przykładowe problemy: klasyfikacja i regresja



# Klasyfikacja

- predykcja wartości dyskretnych/kategorycznych
- przypisywanie przykładów do określonej klasy (etykiety)
- przykłady:
  - klasyfikacja maili jako spam
  - określanie gatunku zwierzęcia



# Dane – podział



# Podział danych

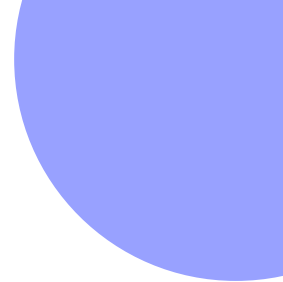


Zanim dostarczymy do modelu nasze dane, musimy zapewnić ich odpowiedni podział na:

- zbiór treningowy (uczący) – na którym nasz model będzie się uczył, muszą to być dane kompletne (wraz z klasą), stanowią około 70–95% zbioru danych
- zbiór testowy (walidujący) – na którym będziemy ewaluować (sprawdzać jakość działania) wyuczonego modelu. Do modelu dostarczamy wtedy same atrybuty i czekamy na jego decyzję (przydział do klasy), a następnie porównujemy z wynikiem prawdziwym.



# Metody podziału danych



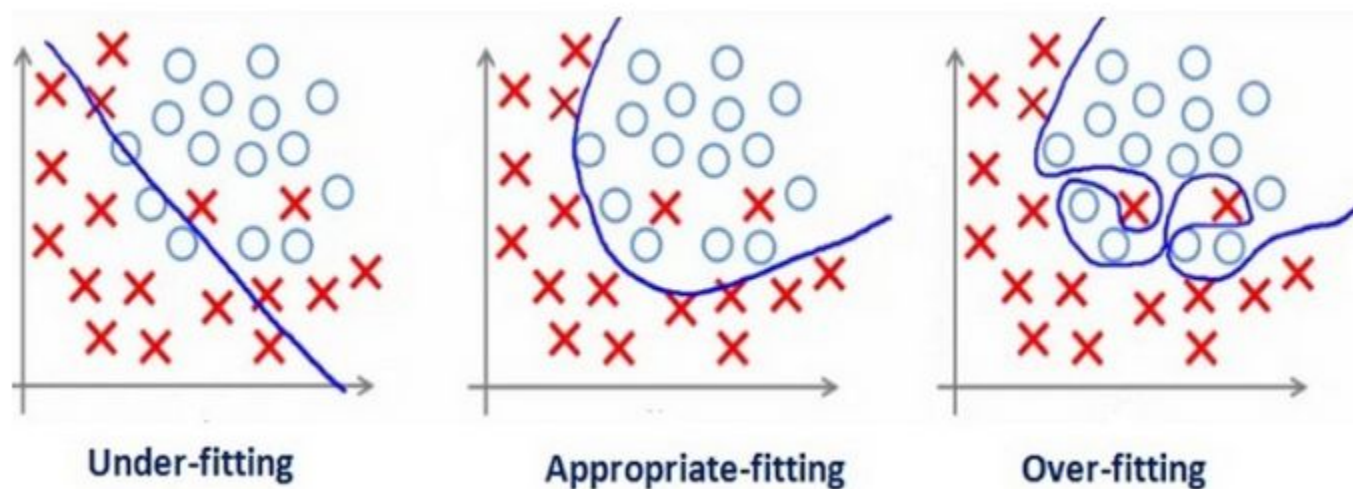
Zwykły, statyczny podział danych, jest mocno nieefektywny i często prowadzi do przeuczenia (overfittingu) modelu – zwłaszcza przy małych zbiorach. Dlatego często prowadzi się tak zwaną walidację krzyżową:

- k-foldową, gdzie dzielimy zbiór na k kawałków, gdzie jeden zawsze służy do walidacji, a pozostałe do uczenia. Proces uczenia powtarzamy k razy
- stratyfikowaną k-foldową, bardzo podobną do poprzedniej, z tym, że w każdej części stara się zachować równomierny rozkład klas



# Niedouczenie i przeuczenie danych

Z drugiej strony, należy również uważać, czy model nie jest niedouczony, to znaczy za bardzo generalizuje dane





# Biblioteka Sklearn (sci-kit learn)



- proste i efektywne narzędzie do analizy danych
- dostępne online, do użycia w różnych kontekstach
- zbudowane w oparciu o biblioteki numpy, scipy i matplotlib
- oprogramowanie open source, do użytku komercyjnego włącznie
- zawiera metody stratyfikacji, walidacji, ewaluacji, wszystkie modele będące treścią tych zajęć
- wspaniała dokumentacja!



# Instalacja Sklearn

W konsoli komenda:

*sudo pip3 install sklearn*

Bądź poprzez Pycharm

```
test@DESKTOP-2DKEUGV: ~  
test@DESKTOP-2DKEUGV:~$ sudo pip3 install sklearn  
[sudo] password for test:  
Collecting sklearn  
  Downloading sklearn-0.0.tar.gz (1.1 kB)  
Collecting scikit-learn  
  Downloading scikit_learn-0.23.2-cp38-cp38-manylinux1_x86_64.whl (6.8 MB)  
    | 6.8 MB 4.7 MB/s  
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.8/dist-packages (from scikit-learn->sklearn) (1.19.1)  
Collecting threadpoolctl>=2.0.0  
  Downloading threadpoolctl-2.1.0-py3-none-any.whl (12 kB)  
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.8/dist-packages (from scikit-learn->sklearn) (1.5.2)  
Collecting joblib>=0.11  
  Downloading joblib-0.17.0-py3-none-any.whl (301 kB)  
    | 301 kB 5.1 MB/s  
Building wheels for collected packages: sklearn  
  Building wheel for sklearn (setup.py) ... done  
  Created wheel for sklearn: filename=sklearn-0.0-py2.py3-none-any.whl size=1315 sha256=b82a27bc994a970769514c76d696663158be9b86968c89d789fbb46072367532  
  Stored in directory: /root/.cache/pip/wheels/22/0b/40/fd3f795caaa1fb4c6cb738bc1f56100be1e57da95849bfc897  
Successfully built sklearn  
Installing collected packages: threadpoolctl, joblib, scikit-learn, sklearn  
Successfully installed joblib-0.17.0 scikit-learn-0.23.2 sklearn-0.0 threadpoolctl-2.1.0  
test@DESKTOP-2DKEUGV:~$
```

# Metryki klasyfikacji



# Różnica między Ground truth a predykcją

- sprawdzenie, w ilu przypadkach wynik modelu nie zgadza się z wartością prawdziwą zbioru testowego
- raczej nieużywane ze względu na małą wartość informacyjną

```
GT:    [0 0 0 0 1 0 0 0 0 0 0 1 0 0 0]
Pred:  [0 0 0 0 1 0 0 0 0 0 0 1 0 0 0]
Dumb:  [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

# Macierz pomyłek (Confusion Matrix)

- odzwierciedlenie rodzajów błędów
- podział na TP, TN, FP, FN
- TP – True Positive – przykład poprawnie zakwalifikowany jako prawdziwy
- TN – True Negative – przykład poprawnie zakwalifikowany jako nieprawdziwy (klasa 0)
- FP – False Positive – przykład niepoprawnie zakwalifikowany jako prawdziwy (wynik 1, gdy poprawnie 0)
- FN – False Negative – przykład niepoprawnie zakwalifikowany jako nieprawdziwy (wynik 0, gdy poprawnie 1)

		ACTUAL VALUES	
		NEGATIVE	POSITIVE
PREDICTED VALUES	NEGATIVE	TRUE NEGATIVES	FALSE NEGATIVES
	POSITIVE	FALSE POSITIVES	TRUE POSITIVES

# Macierz pomyłek – przykład

- Załóżmy że mamy 13 zdjęć, 8 przedstawia kota (klasa 1), 5 przedstawia psa (klasa 0)
- Są one ułożone w następującej sekwencji: [1,1,1,1,1,1,1,1,0,0,0,0,0]
- Klasyfikator zwraca nam następujący wynik: [0,0,0,1,1,1,1,1,0,0,0,1,1]
- Na tej podstawie możemy stworzyć macierz pomyłek

		Actual class	
		Cat	Dog
Predicted class	Cat	5	2
	Dog	3	3

		Actual class	
		P	N
Predicted class	P	TP	FP
	N	FN	TN

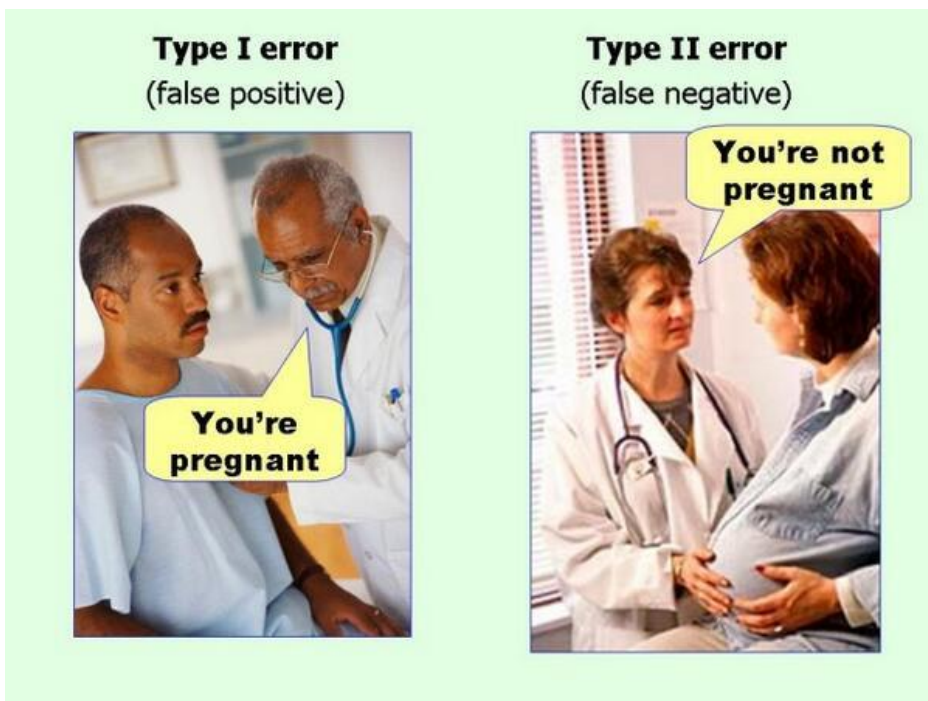
# Błąd pierwszego i drugiego rodzaju

- błąd I rodzaju – odrzucamy hipotezę zerową, gdy jest ona prawdziwa
- błąd II rodzaju – przyjmujemy nieprawdziwą hipotezę zerową

Hipoteza zerowa	Decyzja	
	przyjąć $H_0$	odrzuć $H_0$
Hipoteza zerowa prawdziwa	decyzja prawidłowa	błąd I rodzaju
Hipoteza zerowa fałszywa	błąd II rodzaju	decyzja prawidłowa

# Błąd pierwszego i drugiego rodzaju

- błąd I rodzaju – odrzucamy hipotezę zerową gdy jest ona prawdziwa
- błąd II rodzaju – przyjmujemy nieprawdziwą hipotezę zerową



Hipoteza zerowa	Decyzja	
	przyjąć $H_0$	odrzuć $H_0$
Hipoteza zerowa prawdziwa	decyzja prawidłowa	błąd I rodzaju
Hipoteza zerowa fałszywa	błąd II rodzaju	decyzja prawidłowa



# Accuracy (dokładność)

- procent poprawnie zaklasyfikowanych przykładów
- 1 oznacza, że wszystkie przykłady zostały zaklasyfikowane poprawnie
- źle radzi sobie przy niezbalansowanych danych! Na przykład, gdy mamy za zadanie sprawdzać, czy produkty z fabryki nie są wadliwe, a wadliwy jest tylko jeden produkt na sto – to gdy przyjmiemy, że model ma cały czas zwracać wartość POPRAWNE, to będzie miał on dokładność 99%, mimo ewidentnie złego działania

$$\text{accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$





# Recall (czułość)



- określa jaką część dodatnich wyników wykrył klasyfikator
- 1 oznacza, że wykryto wszystkie pozytywne przykłady
- 0 – nie wykryto żadnego
- bardzo ważne w przypadku np. badań medycznych
- powiązana ze swoistością, czyli stosunkiem wyników prawdziwie ujemnych do sumy prawdziwie ujemnych i fałszywie dodatnich

$$\text{recall} = \frac{TP}{TP + FN}$$




# Precision (precyzja)

- sprawdza pewność klasyfikatora dla przykładów pozytywnych
- im mniejsza precyzja tym więcej False Positives
- istotna na przykład przy wykrywaniu naruszeń prawa


$$\text{precision} = \text{PPV} = \frac{TP}{TP + FP}$$

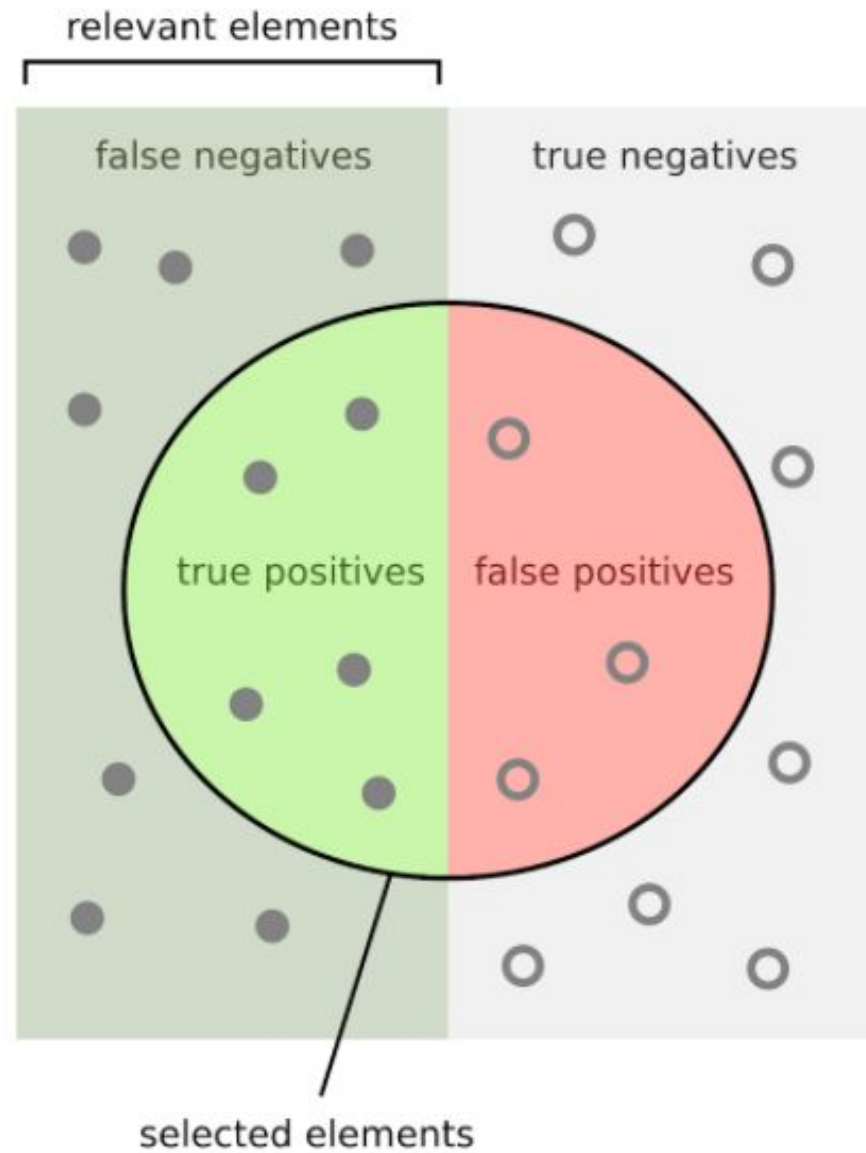
# Zobrazowanie

How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$


How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$






# F1 score

- łączy precyzję i czułość – średnia harmoniczna
- ogólnie – im wyższy F1 Score tym lepszy model

$$F = \frac{2 \cdot \text{precision} \cdot \text{recall}}{(\text{precision} + \text{recall})}$$



# Inne metryki



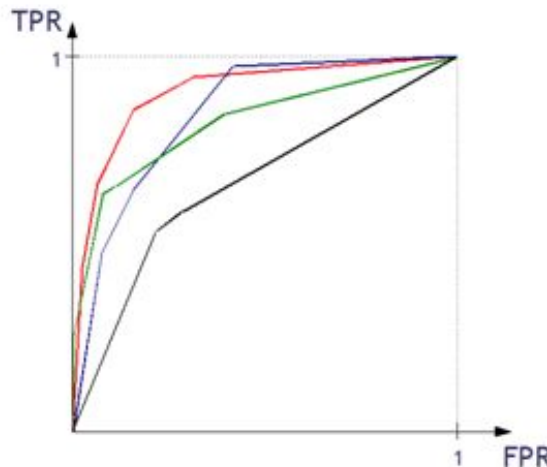
- TPR to stosunek poprawnie zakwalifikowanych przykładów pozytywnych do wszystkich przykładów pozytywnych
- FPR to stosunek niepoprawnie zakwalifikowanych przykładów pozytywnych do wszystkich przykładów pozytywnych do wszystkich przykładów negatywnych

$$\text{TPR} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{FPR} = \frac{\text{FP}}{\text{N}} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

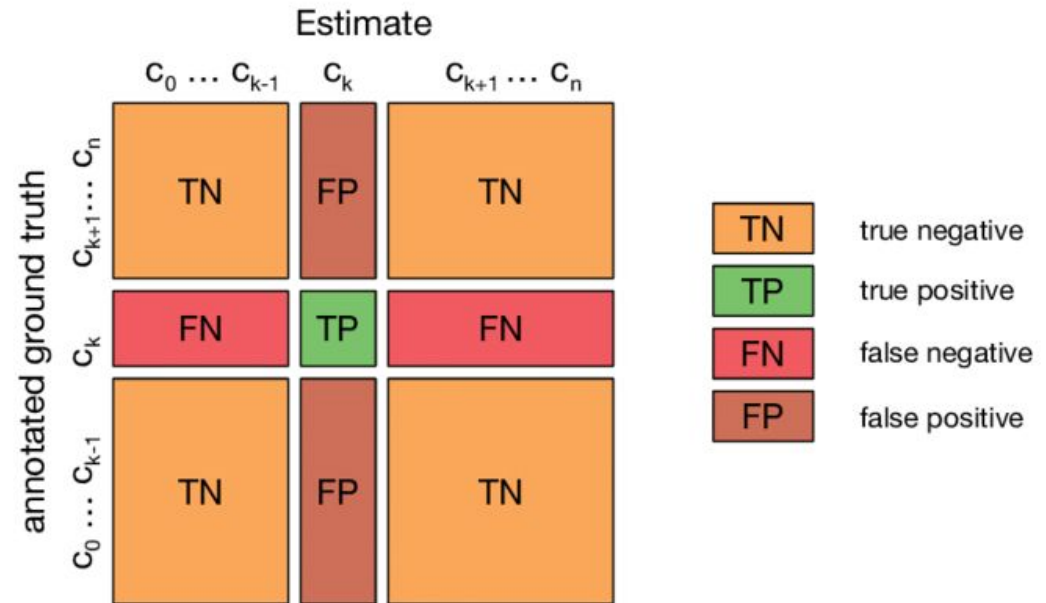
# Krzywa ROC – Receiver Operating Characteristic

- Krzywa ROC to jeden ze sposobów wizualizacji jakości klasyfikacji, pokazujący zależności wskaźników TPR (True Positive Rate) oraz FPR (False Positive Rate).
- wykonujemy ją zmieniając próg modelu, tzn. kiedy ma uznać, że dana próbka należy do klasy 1 czy 0
- im bardziej wypukły wykres modelu, tym jest on lepszy (w przykładzie najlepszy model jest czerwony), to znaczy im pole pod powierzchnią wykresu (AUC – area under curve) jest większe



# Co w przypadku klasyfikacji wieloklasowej?

- obliczanie osobno dla każdej klasy
- średnia:
  - przez zsumowanie TP, FP, TN, FN
  - przez uśrednienie wyników miar dla każdej klasy (ważone lub nie)





# Czas na praktykę!

Będziemy korzystać z danych, które znajdziesz w zeszycie z metryk.

Możesz go pobrać z sekcji “Dodatkowe materiały do bloku” [tutaj](#).

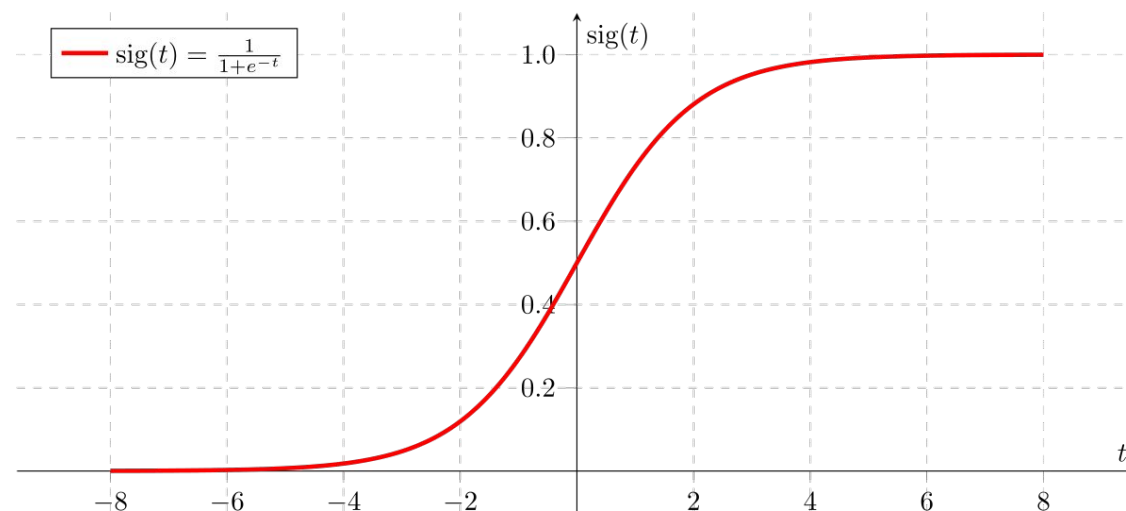


# Regresja logistyczna



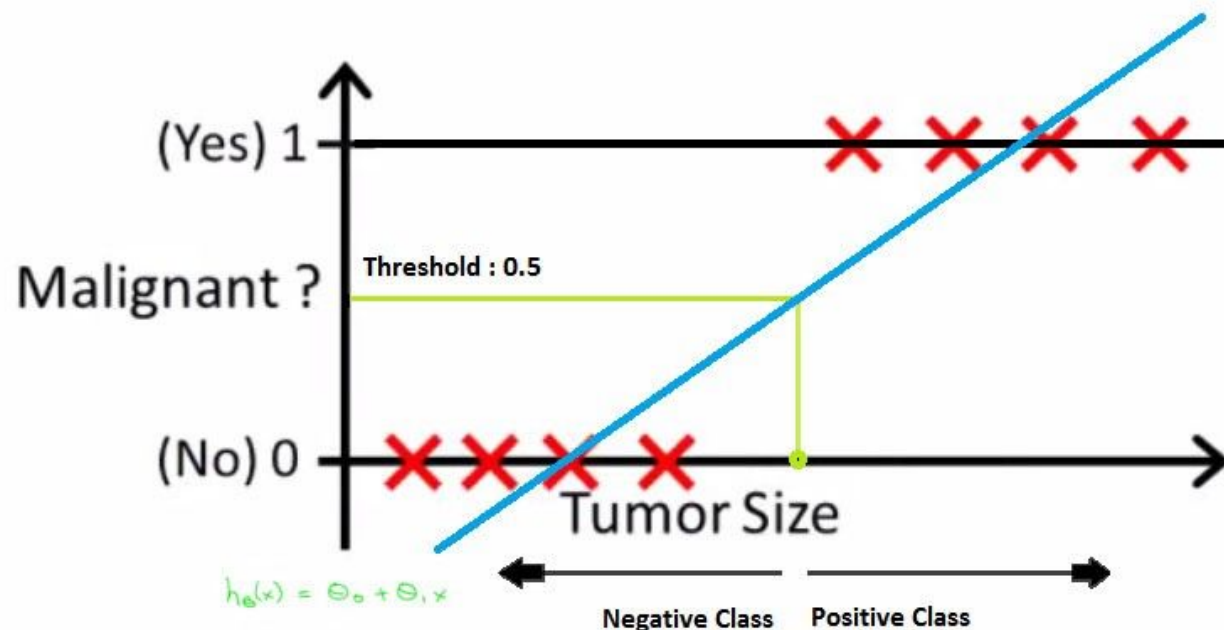
# Regresja logistyczna

- wbrew nazwie służy do klasyfikacji
- metoda statystyczna do predykcji klas binarnych
- binarna klasyfikacja - tylko dwie możliwe klasy
- wykorzystuje funkcję sigmoidalną



# Dlaczego nie regresja liniowa?

Rozpatrzmy prosty przykład dla 8 próbek danych o klasie 1 i 0, zwykła regresja liniowa dość łatwo wskaże granicę pomiędzy klasami.

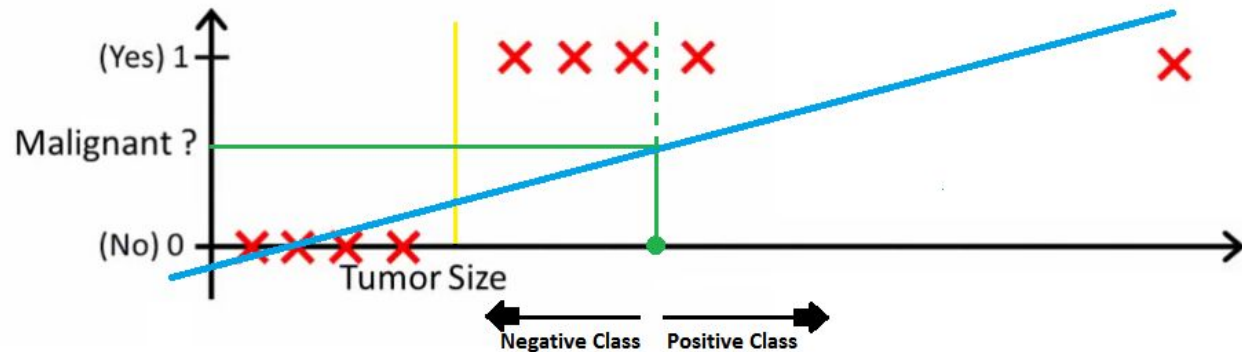


# Dlaczego nie regresja liniowa?

Rozpatrzmy prosty przykład dla 8 próbek danych o klasie 1 i 0, zwykła regresja liniowa dość łatwo wskaże granicę pomiędzy klasami.

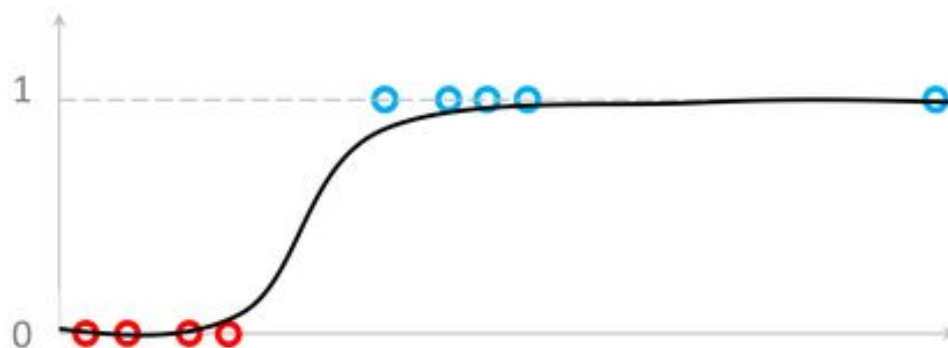
Jednak jeśli dodamy kolejną wartość, dość mocno odstającą, wtedy regresja liniowa, wbrew prawdzie, zaliczy większość przykładów z klasy 1 do klasy 0, ze względu na to, że funkcja liniowa nie będzie mogła dobrać odpowiednich współczynników.

W tym przypadku sprawdzi nam się regresja logistyczna z funkcją sigmoidalną.



# Funkcja sigmoidalna

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

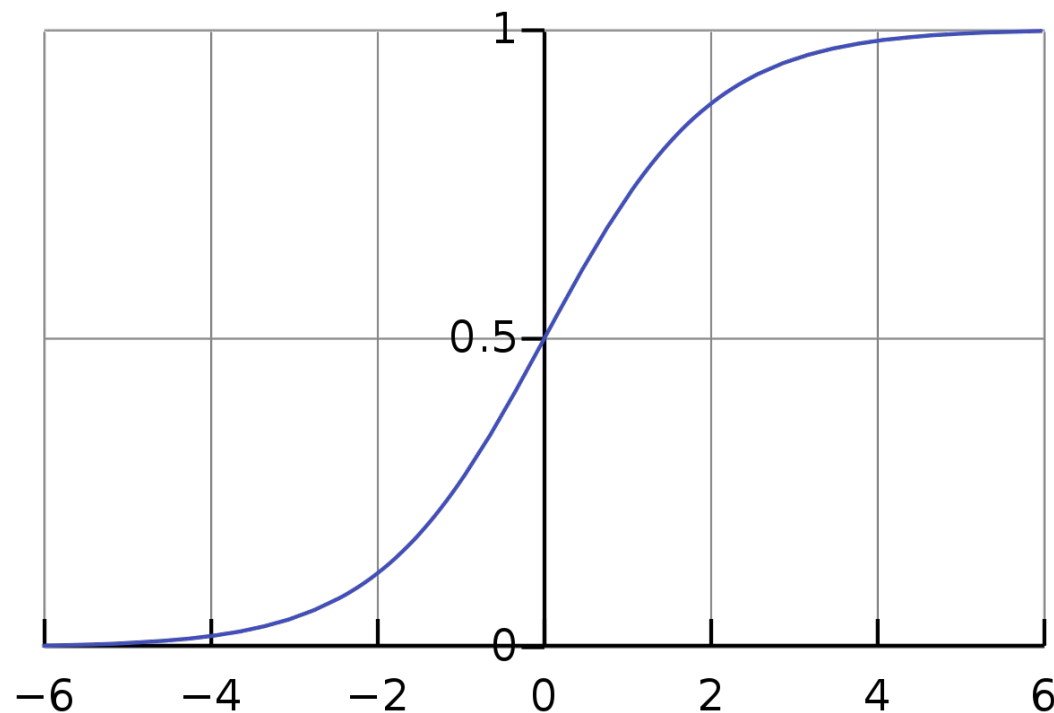


# Regresja logistyczna

$$h_{\theta}(x) = g(\theta^T x)$$

$$z = \theta^T x$$

$$g(z) = \frac{1}{1 + e^{-z}}$$



$$h_{\theta}(x) = P(y = 1|x; \theta) = 1 - P(y = 0|x; \theta)$$



# Regresja logistyczna – funkcja kosztu



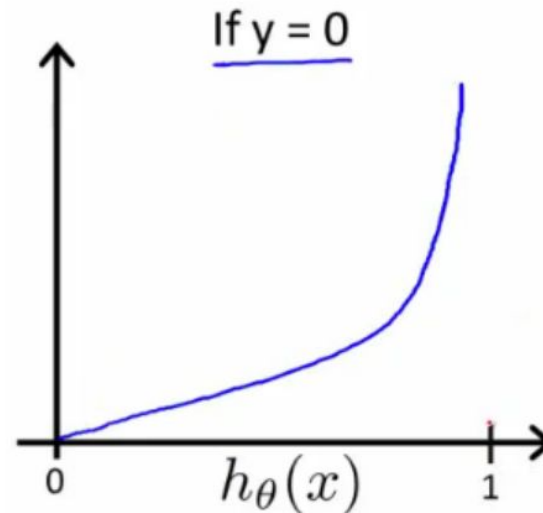
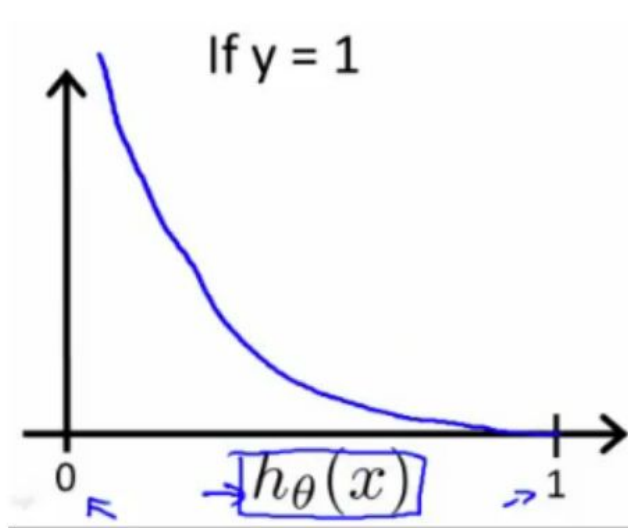
$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))]$$

# Regresja logistyczna - funkcja kosztu

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_{\theta}(x), y) = -\log(h_{\theta}(x)) \quad \text{if } y = 1$$

$$\text{Cost}(h_{\theta}(x), y) = -\log(1 - h_{\theta}(x)) \quad \text{if } y = 0$$



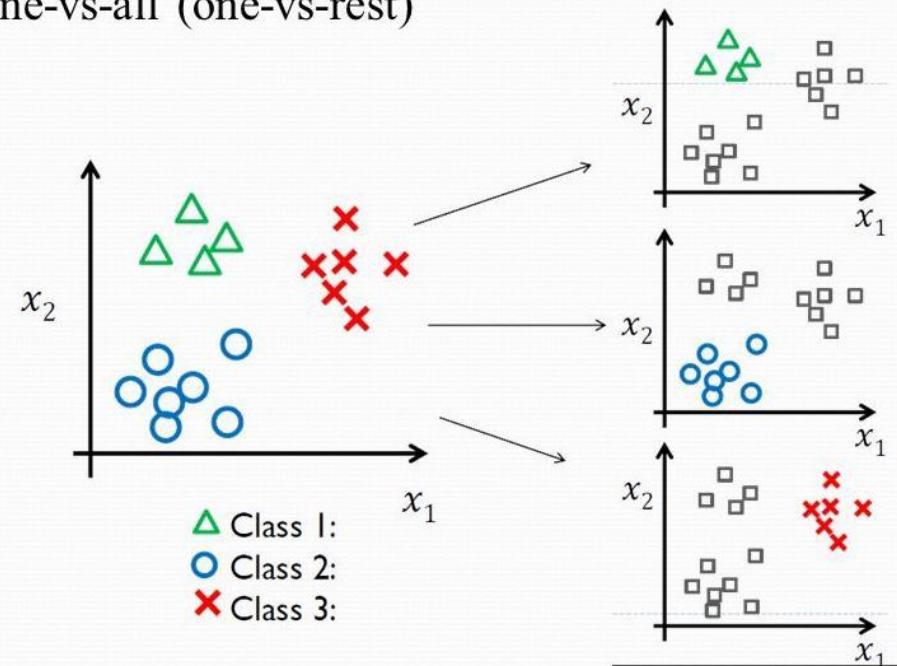


# Regresja logistyczna – klasyfikacja wieloetykietowa

Metoda one vs rest:

- stosowane w klasyfikatorach binarnych
- tworzymy oddzielny model dla każdej klasy i trenujemy: 1 dla tej klasy, 0 dla pozostałych klas
- wbudowane w scikit

One-vs-all (one-vs-rest)





# Regularyzacja



Podczas trenowania modelu modyfikowane są jego wagi — wraz z dalszą specjalizacją modelu wagi zwiększają się, dopasowując się do przykładów występujących w zbiorze uczącym. Ponieważ wysokie wartości wag oznaczają zwykle przetrenowanie modelu (wynikają z nadmiernego dopasowania), preferowane są modele prostsze, o niższych wartościach wag.

Wśród metod regularyzacji wag należy wyróżnić regularyzacje L1 oraz L2. Polegają one na modyfikacji postaci funkcji kosztu przez dodanie członu regularyzacji, co wymusza zmniejszanie wag w trakcie trenowania modelu ( $L = \text{Loss} + \lambda \text{Reg}$ , gdzie  $\lambda$  jest hiperparametrem — określa jak bardzo penalizowane są wysokie wagi).

# Regularyzacja L1 i L2

W regularyzacji **L1 (Lasso)** człon regularyzacji to suma wartości bezwzględnych wszystkich wag ( $\text{Reg} = \|w\|_1$ ). Ponieważ rozważana jest wartość bezwzględna wag, metoda regularyzacji L1 może prowadzić do zmniejszenia ich do zera, dzięki czemu możliwa jest selekcja cech (poprzez przypisanie nieistotnym cechom wejściowym wag zerowych) oraz kompresja modelu, często jednak może ona prowadzić do zjawiska underfittingu.

W przypadku regularyzacji **L2 (Ridge, nazywanej czasem z ang. weight decay)** człon ten ma postać  $\text{Reg} = \|w\|_2^2$ , co powoduje zmniejszanie wartości wag, lecz nie do zera (uzasadnienie). Metoda ta nie przeprowadza zatem selekcji cech i jest preferowana w większości sytuacji, gdyż prowadzi do uwzględnienia wszystkich danych wejściowych.



# Czas na praktykę!

Będziemy korzystać z danych, które znajdziesz w zeszycie z regresji logistycznej.

Możesz go pobrać z sekcji  
"Dodatkowe materiały do bloku" [tutaj](#).

# Maszyny Wektorów Nośnych (SVM)





# Support Vectors Machines

Klasyfikator liniowy – choć nie do końca – za pomocą transformacji wymiarów radzi sobie również z danymi liniowo nieseparowalnymi).

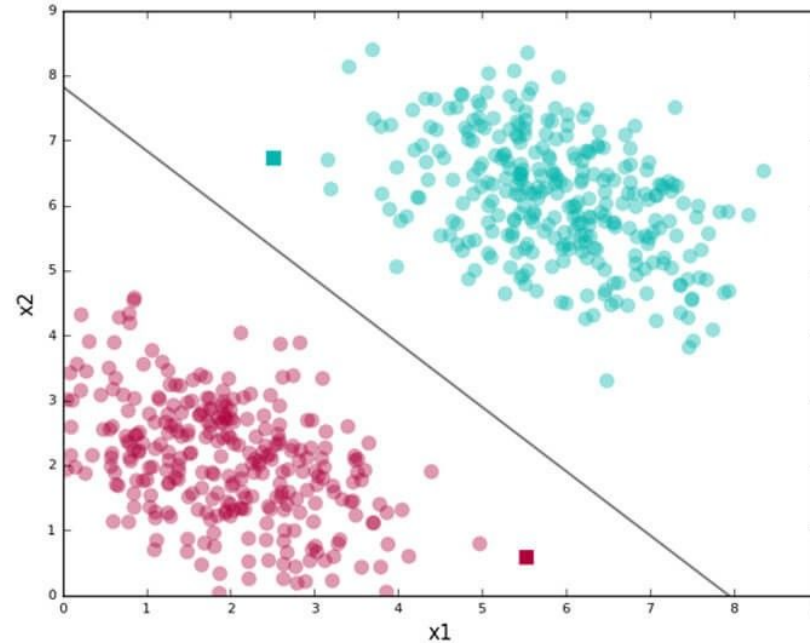
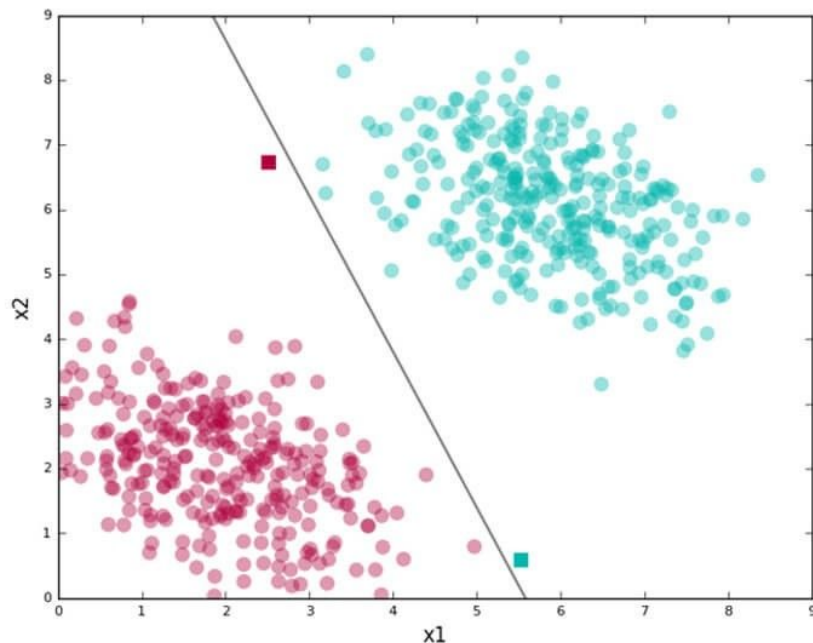
SVM maksymalizuje margines pomiędzy danymi opierając się na wektorach wsparcia.

Wektor wsparcia to przykład danych, który jest blisko granicy marginesu.

Ze względu na branie pod uwagę jedynie wektorów wsparcia SVM daje bardzo szybkie działanie przy zachowaniu wysokiej skuteczności.

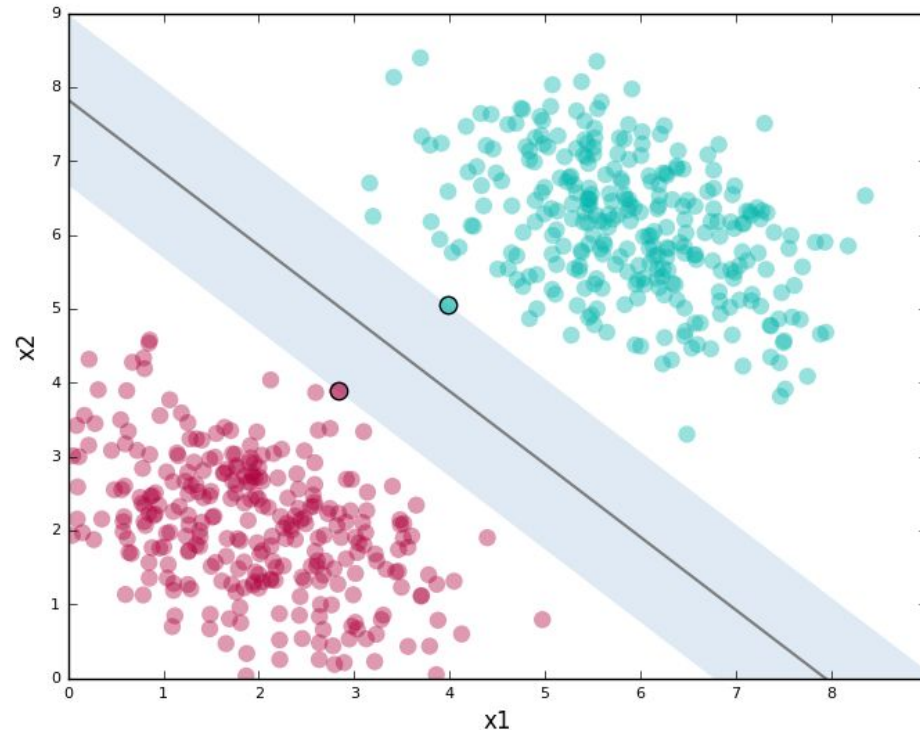
# SVM – wektory wsparcia

Klasyfikator SVM bierze pod uwagę tylko próbki położone blisko granicy klas – tak zwane wektory wsparcia i na ich podstawie rozdziela cały zbiór



# SVM – wektory wsparcia, marginesy

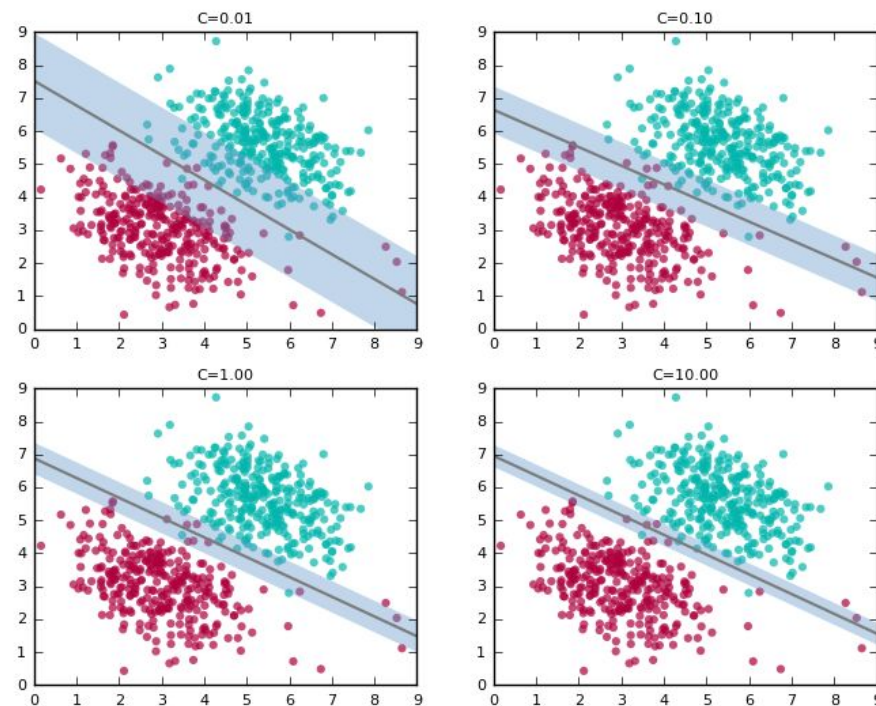
Pozwalają na rozdzielenie klas w zbiorze danych, dążąc do zachowania jak największego marginesu pomiędzy klasami.





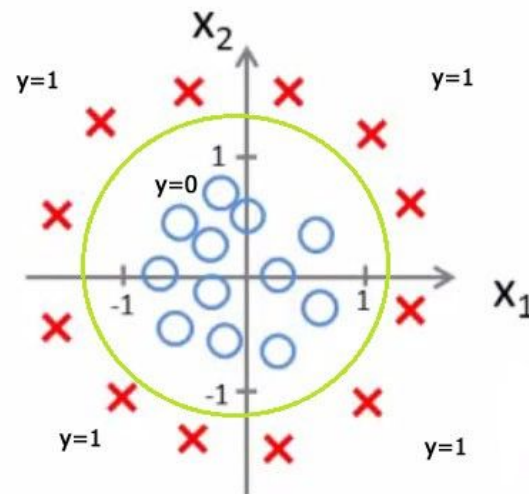
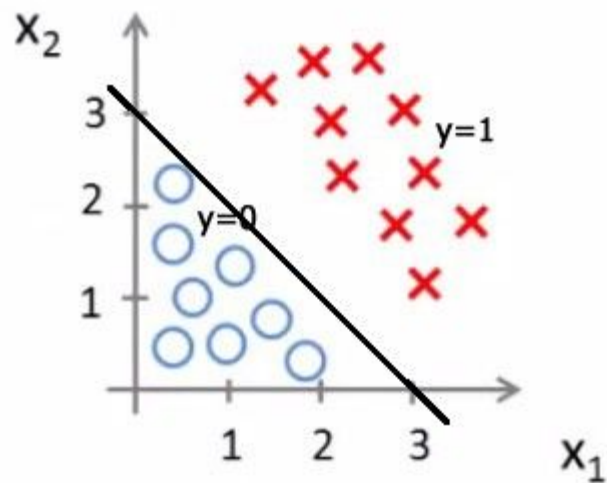
# SVM – możliwość błędów

Przy rzeczywistych zbiorach danych poszczególne próbki często zachodzą na siebie. Za pomocą parametru  $C$  możemy określić, czy model ma dążyć do zachowania mniejszego marginesu i mniejszej liczby błędów czy pozwolić mu na pewne błędy, rozszerzając margines i lepiej generalizując model dla nowych danych.



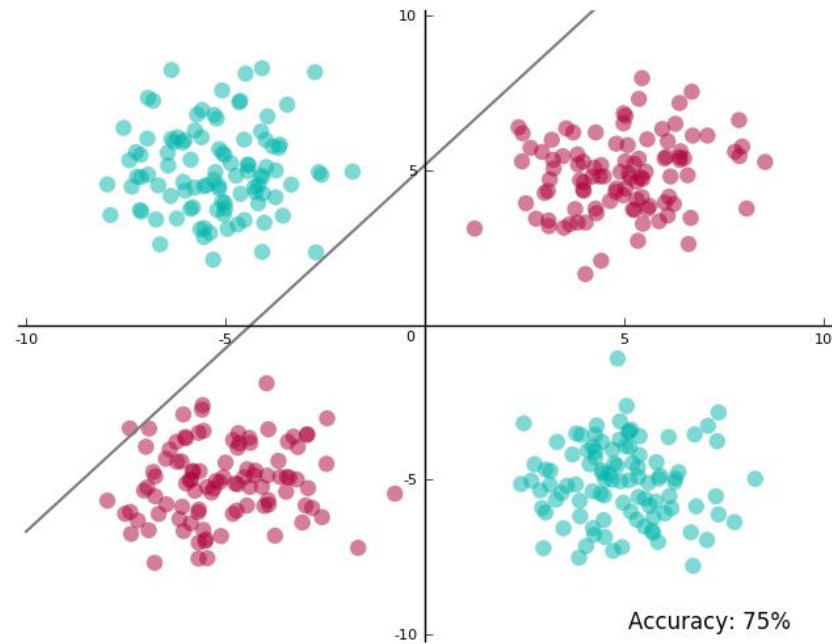
# Problemy liniowo i nieliniowo separowalne

Niestety większość problemów związanych z danymi jest nieliniowo separowalna, to znaczy nie da się ich rozwiązać za pomocą klasyfikatorów liniowych. Można jednak zastosować transformację wymiarów, o czym będzie później.



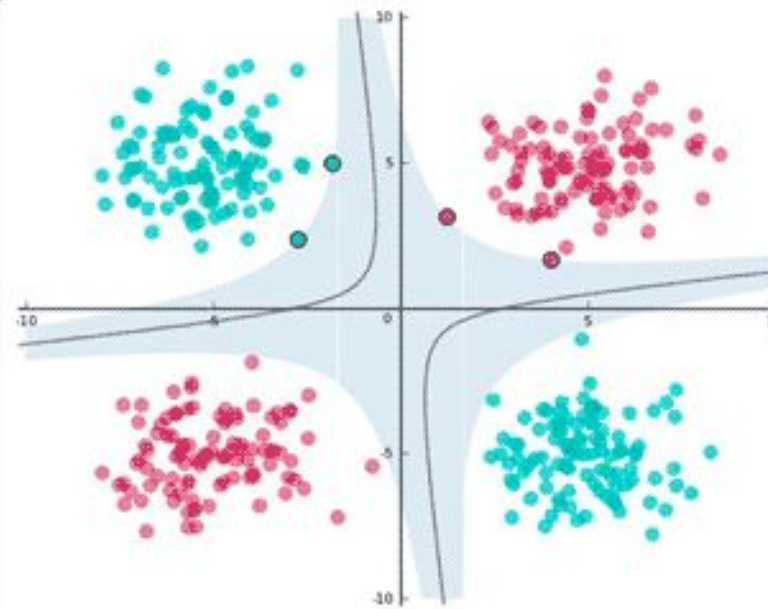
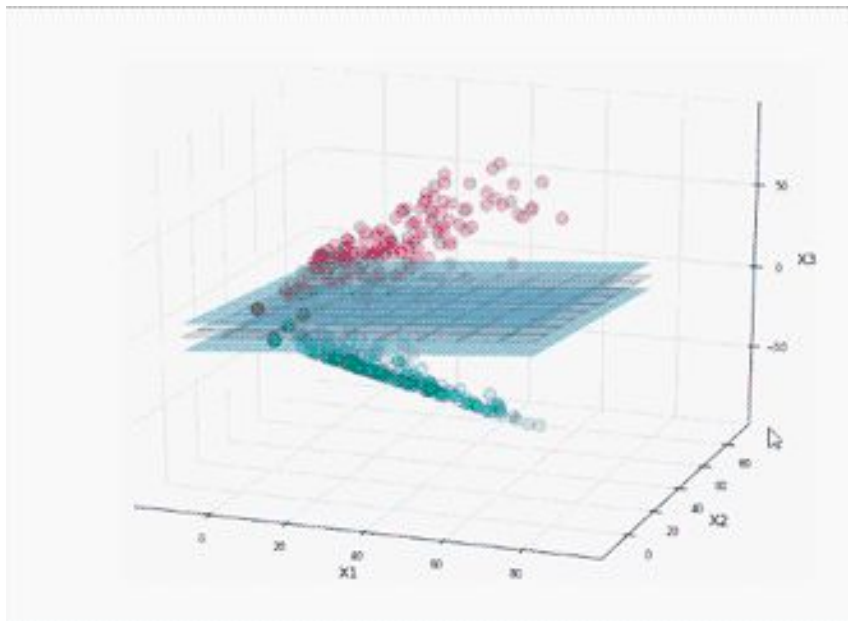
# SVM – dane liniowo nieseparowalne

Co w przypadku danych rozłożonych w ten sposób? Jak sobie z nimi radzić za pomocą klasyfikatora SVM?

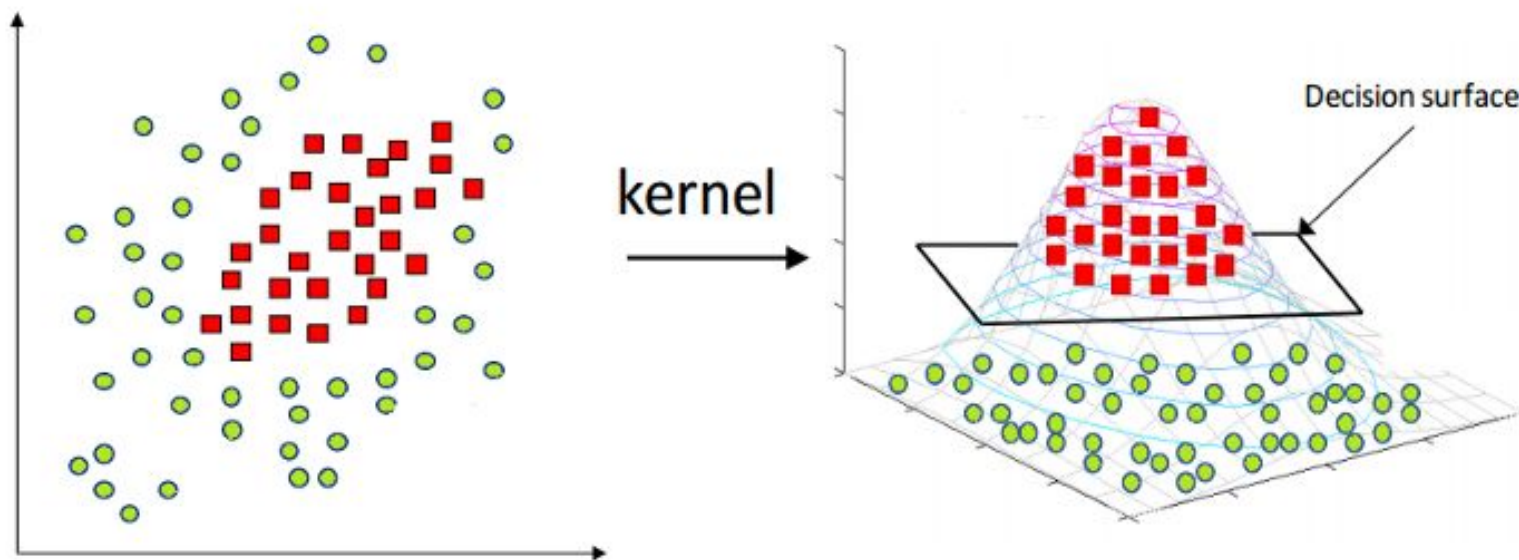


# SVM – transformacja przestrzeni

Za pomocą jądra przekształcenia zmieniamy przestrzeń dodając do niej dodatkowy wymiar, w którym dane są już separowalne.



# SVM – transformacja przestrzeni





# Transformacja przestrzeni – jądro



Jądro (kernel) transformacji, to sposób (wzór), w jaki algorytm transformuje naszą przestrzeń. W bibliotece sklearn jest wbudowanych kilka najpopularniejszych kerneli transformacji.

Wielomianowy:

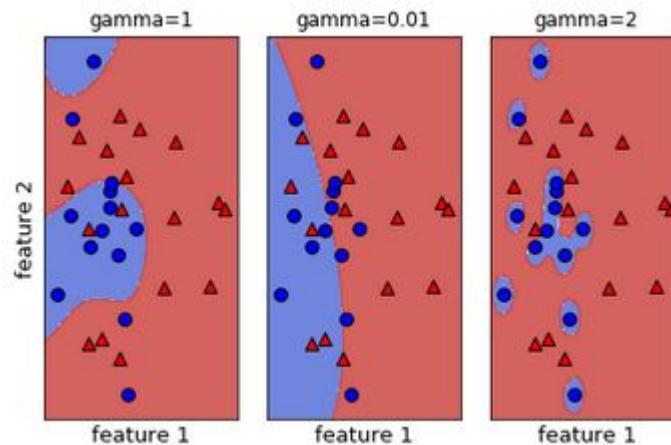
$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^d$$

RBF (Gaussowski):

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$$

# Transformacja przestrzeni – głębokość

Głębokość transformacji (ilość dodatkowych cech, które się wytworzą) można regulować za pomocą **parametru gamma**. Należy uważać, aby nie przeuczyć modelu!





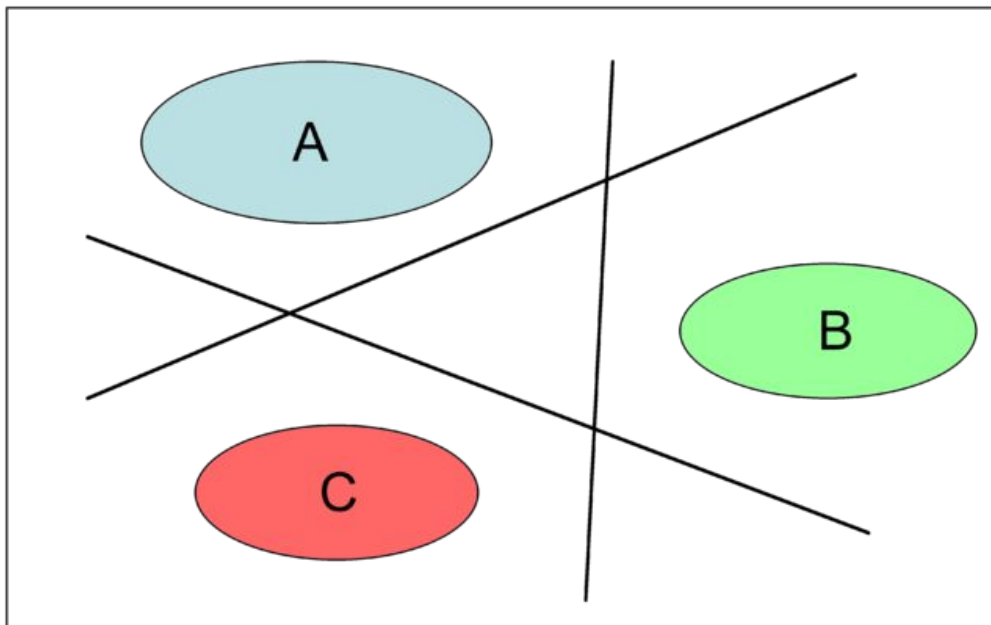
# Porównanie kerneli:

- czas uczenia klasyfikatora SVM: liniowy < wielomianowy < rbf
- zdolność do przystosowania się do różnych danych: liniowy < wielomianowy < rbf
- ryzyko przeuczenia: liniowy < wielomianowy < rbf
- ryzyko niedouczenia: rbf < wielomianowy < liniowy
- liczba hiperparametrów: liniowy < rbf < wielomianowy



# SVM – wiele klas

W odróżnieniu od regresji logistycznej, SVM w przypadku klasyfikacji wieloklasowej stosuje technikę One vs One. Każda klasa jest porównywana z każdą klasą pojedynczo, tzn. A z B, A z C, B z C.






# Regresja logistyczna vs SVM – porównanie



Regresja	SVM
klasyfikacja liniowa	klasyfikacja linowa
Wolna przy transformacji wymiarów	Szybka przy transformacji wymiarów
wyjściem są prawdopodobieństwa danej klasy	nie ma łatwego dostępu do prawdopodobieństw (tak lub nie)
wszystkie punkty są używane do trenowania	tylko wybrane punkty – wektory wsparcia są używane do trenowania
Regularyzacja L1 lub L2	Regularyzacja L2



# Metoda k-najbliższych sąsiadów

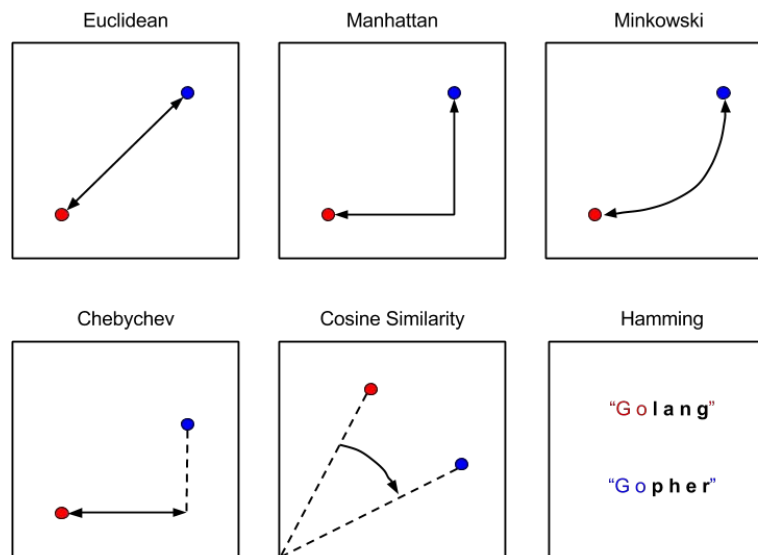


# Algorytm k – najbliższych sąsiadów

Jeden z prostszych algorytmów uczenia maszynowego, klasyfikuje na podstawie położenia najbliższych sąsiadów za pomocą różnych miar odległości (Euklidesowa, Manhattan, Czebyszewa) i sposobów głosowania (większościowe, ważone, ważone do kwadratu).

Jako parametry przyjmuje liczbę sąsiadów, których ma brać pod uwagę, miarę odległości i sposób głosowania.

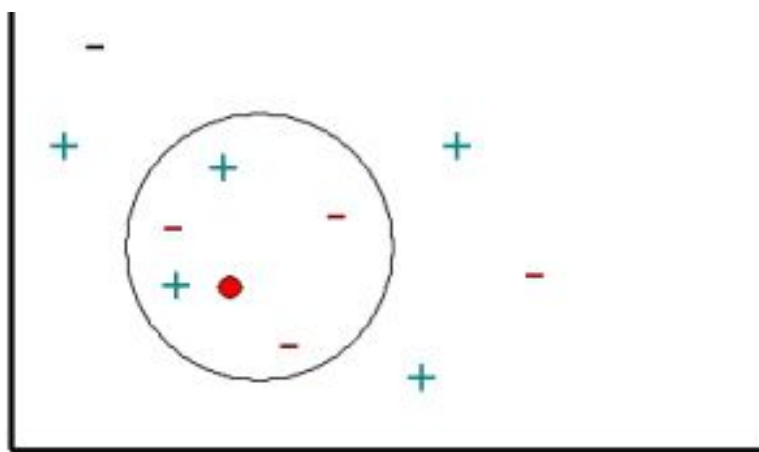
Algorytm leniwy (nie potrzebuje nauki).



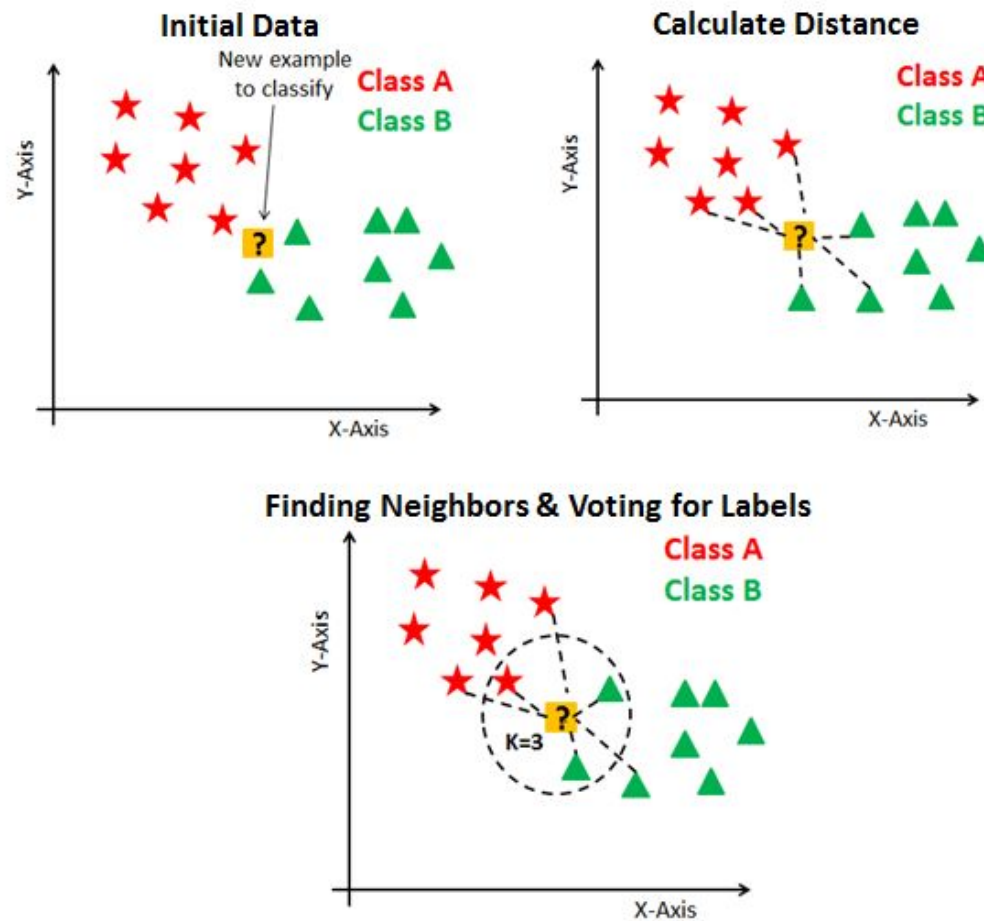
# Algorytm k – działanie

Przykładowo, dla nowego punktu, który pojawia się w zbiorze, jeśli weźmiemy pod uwagę tylko jednego sąsiada i pomiar euklidesowy oraz głosowanie większościowe, wynik będzie dodatni, jeżeli dwóch, nieokreślony, a dla 5 sąsiadów będzie ujemny.

Powinno się ustawiać nieparzystą liczbę sąsiadów, szczególnie przy głosowaniu większościowym!



# Algorytm k - działanie





# Czas na praktykę!

Będziemy korzystać z danych, które znajdziesz w zeszycie z knn i SVM.

Możesz go pobrać z sekcji  
“Dodatkowe materiały do bloku” [tutaj](#).

# Naiwny klasyfikator Bayesowski







# Twierdzenie (reguła) Bayesa



$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

gdzie A i B są zdarzeniami,  $P(B) > 0$ , oraz:

$P(A \mid B)$  – prawdopodobieństwo (warunkowe) zajścia zdarzenia A pod warunkiem zajścia zdarzenia B,

$P(A)$ ,  $P(B)$  – prawdopodobieństwa zajścia/zaobserwowania niezależnych zdarzeń A i B.



# Przykład: gra liczbowa

Założmy, że mamy generator liczb, który wybiera liczby całkowite od 1 do 100, które spełniają określoną regułę. Naszym zadaniem jest znaleźć tą regułę. Po czterech generacjach mamy wylosowane następujące liczby: 16, 2, 8, 64.

Jaka jest reguła?



# Przykład: gra liczbowa

Założmy, że mamy generator liczb, który wybiera liczby całkowite od 1 do 100, które spełniają określoną regułę. Naszym zadaniem jest znaleźć tą regułę. Po czterech generacjach mamy wylosowane następujące liczby: 16, 2, 8, 64.

Jaka jest reguła?

**Potęgi liczby 2.**



# Przykład: gra liczbowa

Założmy, że mamy generator liczb, który wybiera liczby całkowite od 1 do 100, które spełniają określoną regułę. Naszym zadaniem jest znaleźć tę regułę. Po czterech generacjach mamy wylosowane następujące liczby: 16, 2, 8, 64.

Jaka jest reguła?

**Potęgi liczby 2.**

Ale dlaczego nie liczby parzyste? Dlaczego nie wszystkie liczby poza 37?

**Żeby odpowiedzieć na to pytanie musimy poznać jeszcze dwa pojęcia**

# Likelihood

Wyjaśnia dlaczego widząc zbiór (16, 2, 8, 64) mówimy że są to potęgi liczby 2, a nie liczby parzyste. Określa on jak prawdopodobne jest wystąpienie danego przykładu mówiącego o tym, że dana hipoteza jest prawdziwa (uwiarygadnia hipotezę w świetle napływających danych).

$$p(D|h) = \left[ \frac{1}{\text{size}(h)} \right]^N$$

gdzie D – zbiór dotychczasowych obserwacji, h – hipoteza, size(h) – wielkość zbioru hipotezy, N – liczba dotychczasowych obserwacji

# Likelihood

$$p(D|h) = \left[ \frac{1}{\text{size}(h)} \right]^N$$

W rozważanym przykładzie:  $D = \{16, 2, 8, 64\}$ ,  $\text{size}(\text{potęgi liczby } 2) = 7$ ,  $\text{size}(\text{liczby parzyste}) = 50$ ,  $N = 4$

$$p(D|\text{potęgi liczby } 2) = (1/7)^4$$

$$p(D|\text{liczby parzyste}) = (1/50)^4$$



# Prior



Określa jak prawdopodobna jest nasza reguła (jeszcze przed wystąpieniem jakichkolwiek próbek). Wykorzystuje się tutaj wiedzę ekspercką albo zaetykietowany zbiór danych.

Hipoteza: liczby parzyste będą miały znacznie wyższe prawdopodobieństwo prior niż liczby parzyste z pominięciem 37.

Jeśli w zbiorze danych mamy 80 kulek czerwonych i 20 zielonych, to prawdopodobieństwo prior wylosowania kulki czerwonej będzie wynosiło 80%.



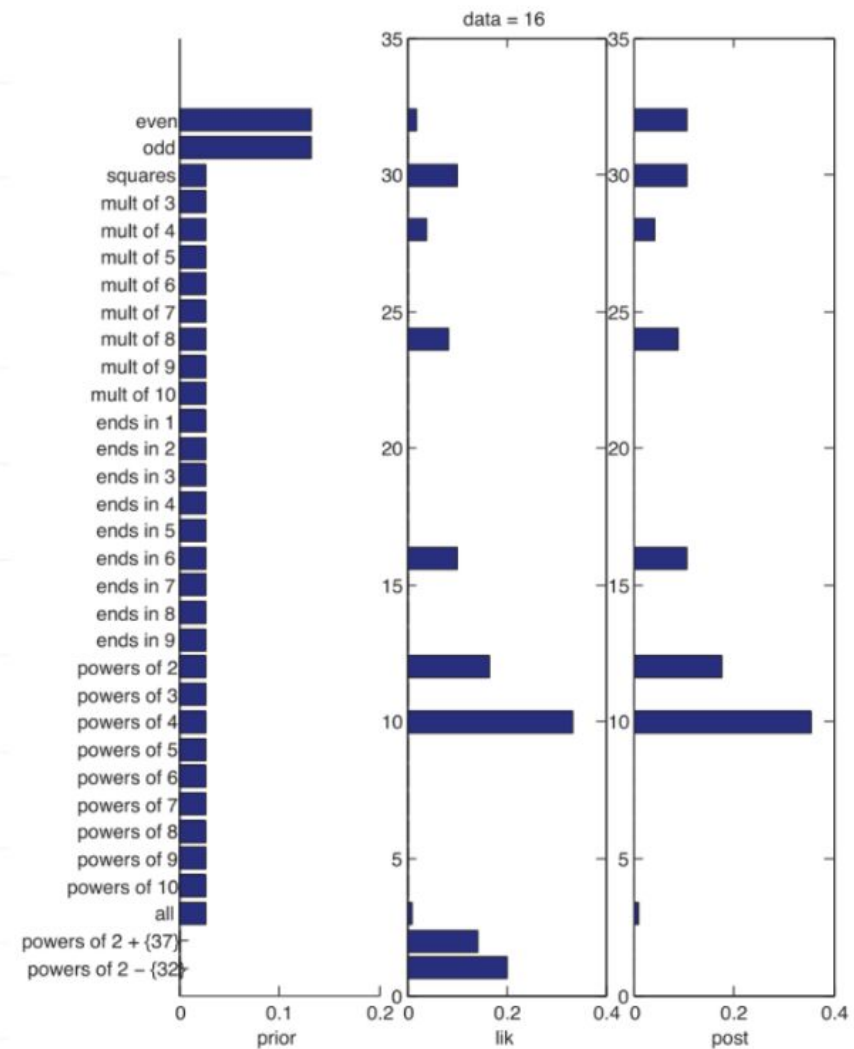
# Posterior

Najprościej jak się da: likelihood pomnożony przez prior.



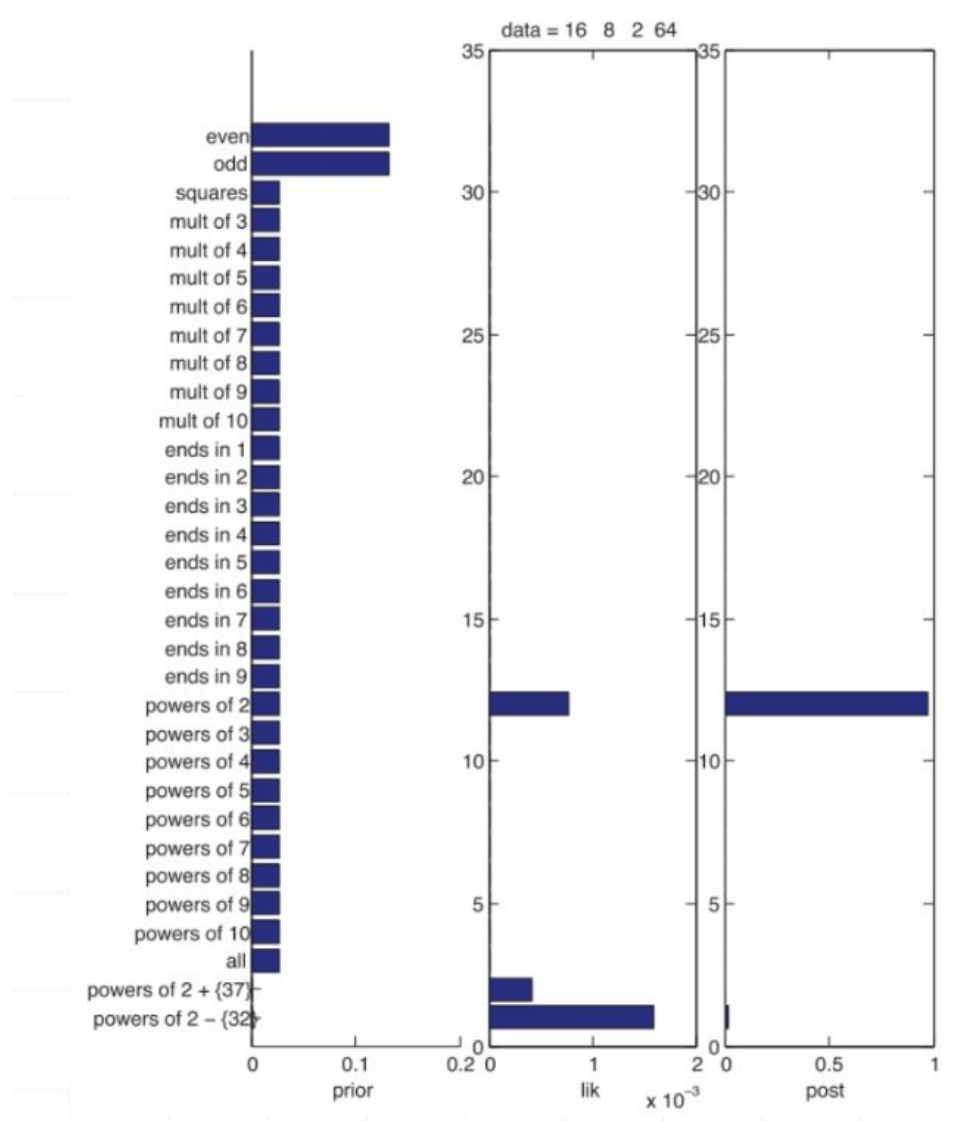
# Nasza gra liczbowa:

Rozkład powyższych  
prawdopodobieństw:



# Nasza gra liczbowa:

A gdyby zmienić dane?



# Podsumowując:

## Likelihood

How probable is the evidence  
given that our hypothesis is true?

## Prior

How probable was our hypothesis  
before observing the evidence?

$$P(H | e) = \frac{P(e | H) P(H)}{P(e)}$$

## Posterior

How probable is our hypothesis  
given the observed evidence?  
(Not directly computable)

## Marginal

How probable is the new evidence  
under all possible hypotheses?  
 $P(e) = \sum P(e | H_i) P(H_i)$




# Klasyfikator naiwnego Bayesa



Klasyfikator naiwnego Bayesa to model probabilistyczny, oparty w całości na twierdzeniu Bayesa.

Algorytm naiwnego Bayesa jest często używany w analizie sentymentu, filtrowaniu spamu, systemach rekomendacyjnych. Jest szybki i łatwy do implementacji. **Jego główną wadą jest założenie o niezależności danych (dlatego mówimy o naiwności klasyfikatora)**. W większości przypadków w realnym życiu atrybuty są między sobą zależne, przez co klasyfikator traci na dokładności.



# Klasyfikator naiwnego Bayesa

## GAUSSIAN NAIVE BAYES CLASSIFIER

"Gaussian" because this is a normal distribution

This is our prior belief

$$P(\text{class} | \text{data}) = \frac{P(\text{data} | \text{class}) \times P(\text{class})}{P(\text{data})}$$

We don't calculate this in naive bayes classifiers

ChrisAlbon



# Klasyfikator naiwnego Bayesa

Traktujemy zmienne jako niezależne, możemy więc zapisać regułę w ten sposób:

$$P(y|x_1, \dots, x_n) = \frac{P(x_1|y)P(x_2|y)\dots P(x_n|y)P(y)}{P(x_1)P(x_2)\dots P(x_n)}$$



# Typy klasyfikatorów



Różnego rodzaju klasyfikatory Bayesowskie, przyjmują różnego rodzaju prawdopodobieństwo prior występowania atrybutów. I tak:

- Klasyfikator Gaussowski przyjmuje normalny rozkład atrybutów, jako prior. Dobrze nadaje się do ciągłych danych (wzrost, waga).
- Klasyfikator Bernoulliego i Multinomialny przyjmuje inne rozkłady atrybutów, obliczane na podstawie ich występowania w zbiorze uczącym. Lepiej nadaje się do danych dyskretnych i na przykład do przetwarzania tekstu.

# Przykład: czy pogoda jest dobra do gry w golfa?

Klasyfikator na podstawie istniejących przykładów oblicza prawdopodobieństwo do gry w golfa dla każdego z atrybutów, to znaczy w ilu procent przypadków gra była możliwa przy deszczu, cieplej temperaturze, czy wilgotności. Potem dla nowych przypadków po prostu podstawia dane do prawdopodobieństw i oblicza, czy gra w golfa jest możliwa.

[http://www.cs.put.poznan.pl/jpotoniec/?page\\_id=305](http://www.cs.put.poznan.pl/jpotoniec/?page_id=305)

	OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY GOLF
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes
5	Sunny	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Rainy	Mild	High	False	No
8	Rainy	Cool	Normal	False	Yes
9	Sunny	Mild	Normal	False	Yes
10	Rainy	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Sunny	Mild	High	True	No





# Problem – ciągłość danych

Algorytmy uczenia maszynowego (przede wszystkim klasyfikatory) nie lubią danych ciągłych. Znacznie lepiej pracuje im się, gdy dane są oddzielone od siebie i przypisane do określonych kategorii. Temu właśnie służy dyskretyzacja. Możemy wyróżnić dyskretyzację:

- równego podziału
- równej częstotliwości
- inne, bardziej skomplikowane

# Problem – ciągłość danych

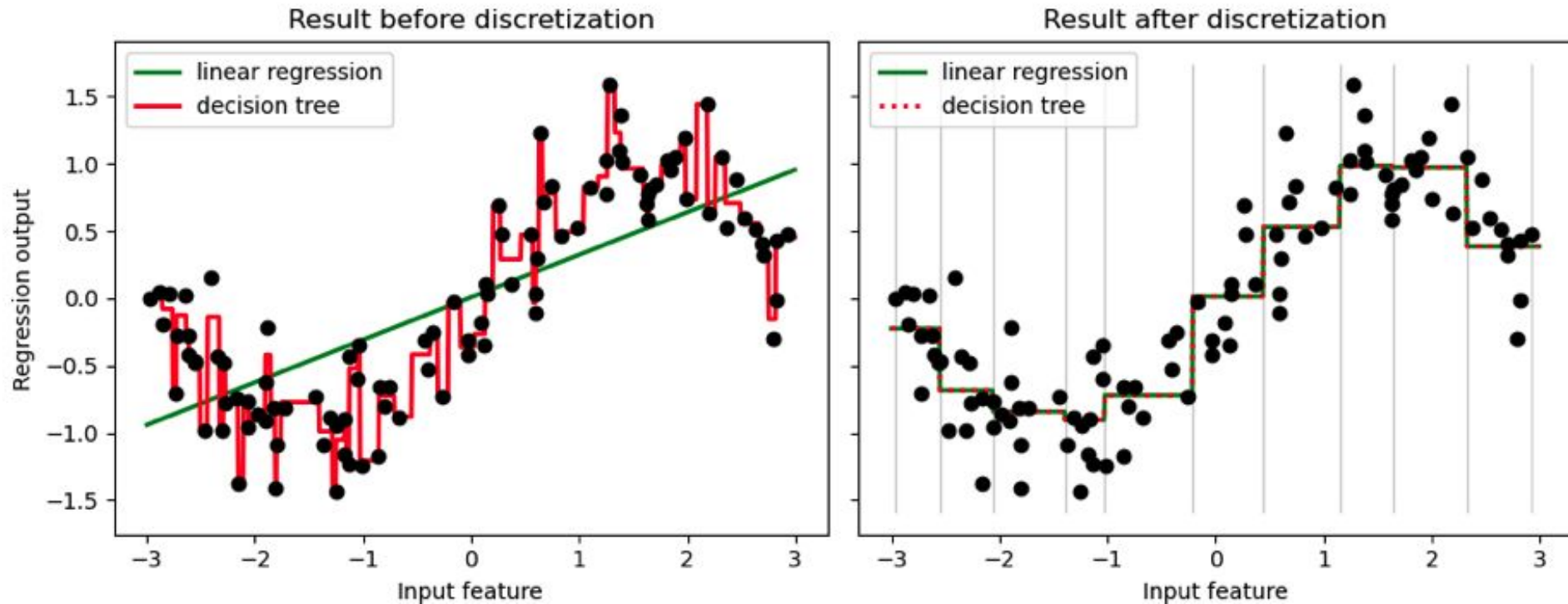
$X = [10, 15, 16, 18, 20, 30, 35, 42, 48, 50, 52, 55]$

$x = 4$	
$w = (55-10)/4 = 12$	
$[\min, \min+w-1]$	[10, 21]
$[\min+w, \min+2*w-1]$	[22, 33]
$[\min+2*w, \min+3*w-1]$	[34, 45]
$[\min+3*w, \max]$	[46, 55]

AGE	AGE_bins
10	[10, 21]
15	[10, 21]
16	[10, 21]
18	[10, 21]
20	[10, 21]
30	[22, 33]
35	[34, 45]
42	[34, 45]
48	[46, 55]
50	[46, 55]
52	[46, 55]
55	[46, 55]

AGE	AGE_bins
10	[10, 16]
15	[10, 16]
16	[10, 16]
18	[17, 30]
20	[17, 30]
30	[17, 30]
35	[31, 48]
42	[31, 48]
48	[31, 48]
50	[49, 55]
52	[49, 55]
55	[49, 55]

# Problem – ciągłość danych



# KBinsDiscretizer

```
[ ] from sklearn.preprocessing import KBinsDiscretizer
```

```
[ ] kb = KBinsDiscretizer(n_bins=10, strategy='uniform')
kb.fit(X)
print("bin edges: \n", kb.bin_edges_)
```

```
bin edges:
[array([-2.967, -2.378, -1.789, -1.2   , -0.612, -0.023,  0.566,  1.155,
        1.744,  2.333,  2.921])]
```

```
[ ] X_binned = kb.transform(X)
X_binned
```

```
<120x10 sparse matrix of type '<class 'numpy.float64'>'
with 120 stored elements in Compressed Sparse Row format>
```

```
▶ print(X[:10])
X_binned.toarray()[:10]
```

```
⦿ [[-0.753]
   [ 2.704]
   [ 1.392]
   [ 0.592]
   [-2.064]
   [-2.064]
   [-2.651]
   [ 2.197]
   [ 0.607]
   [ 1.248]]
array([[0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]])
```



# Czas na praktykę!

Będziemy korzystać z danych, które znajdziesz w zeszycie z naiwnym Bayesem.

Możesz go pobrać z sekcji  
“Dodatkowe materiały do bloku” [tutaj](#).

# Drzewa decyzyjne



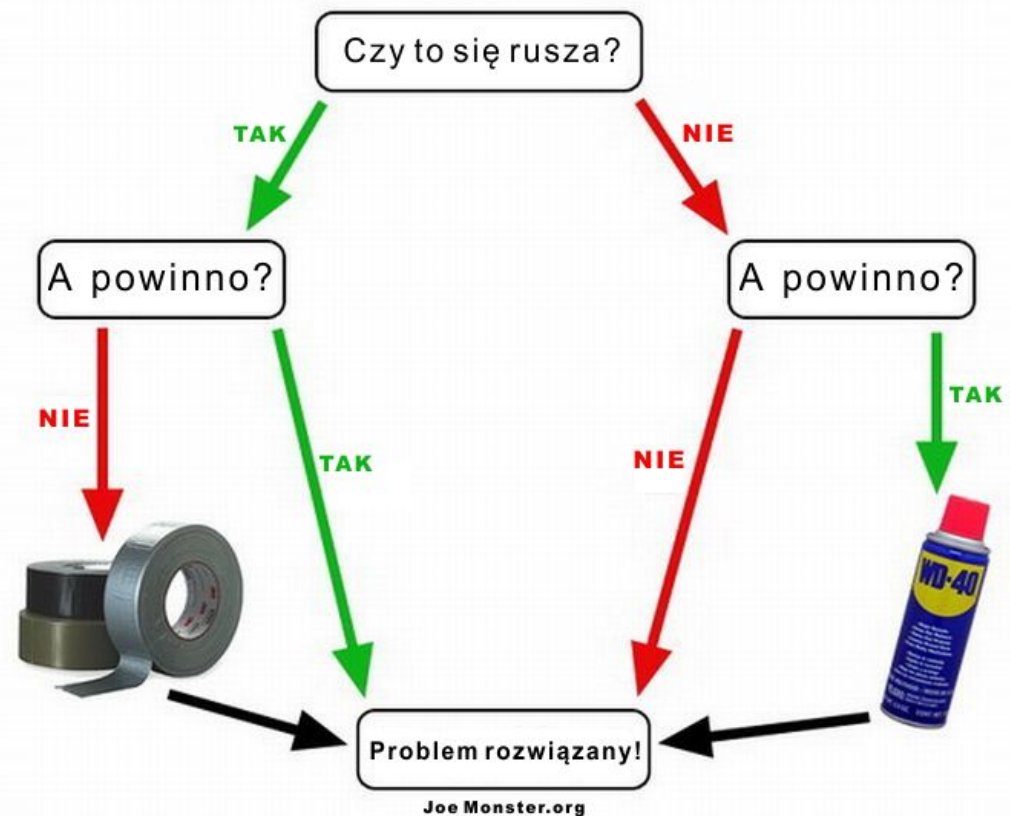
# Drzewo decyzyjne

Wykorzystywane nie tylko w ML, ale również w życiu codziennym.

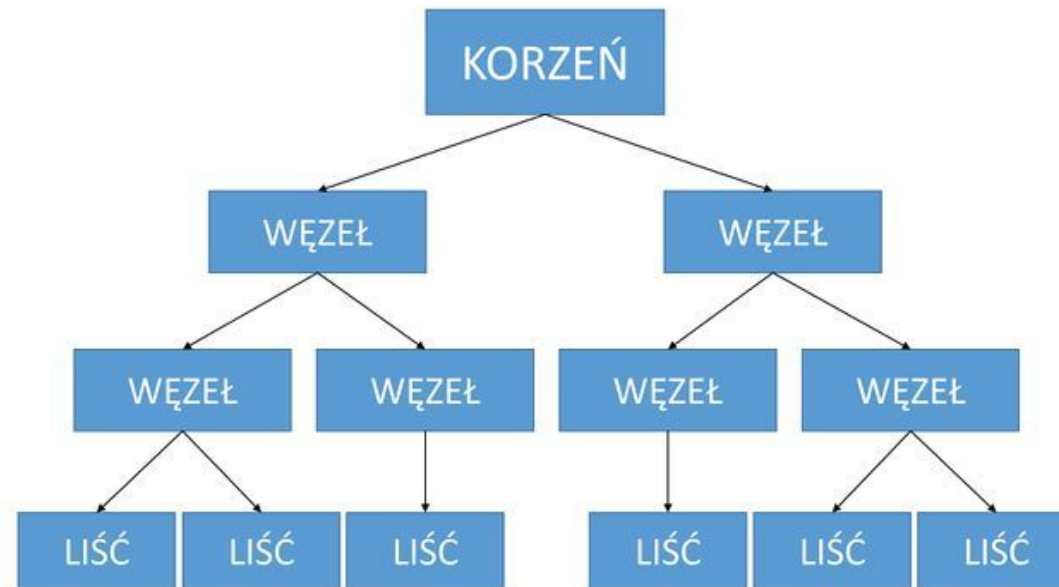
## Jak naprawić **COKOLWIEK?**

EATLIVER.COM

JOE MONSTER.ORG

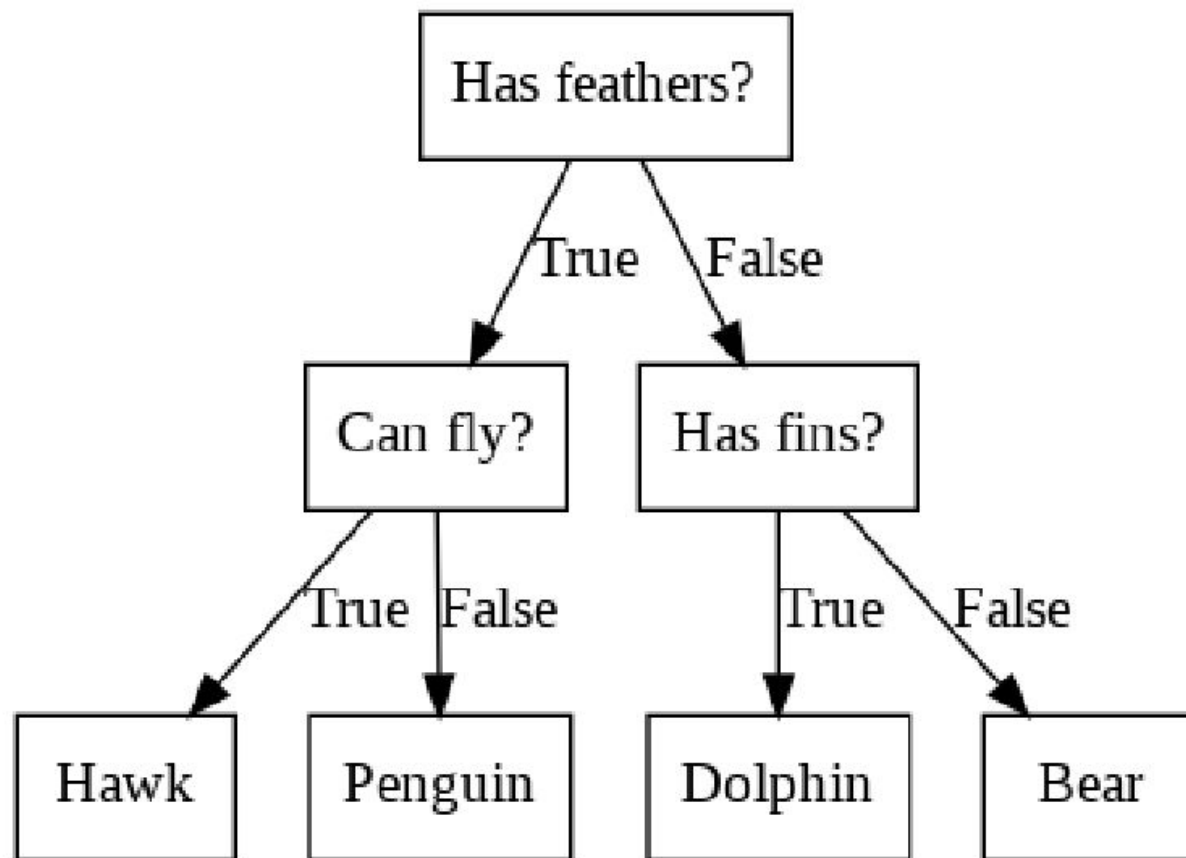


# Struktura drzewiasta

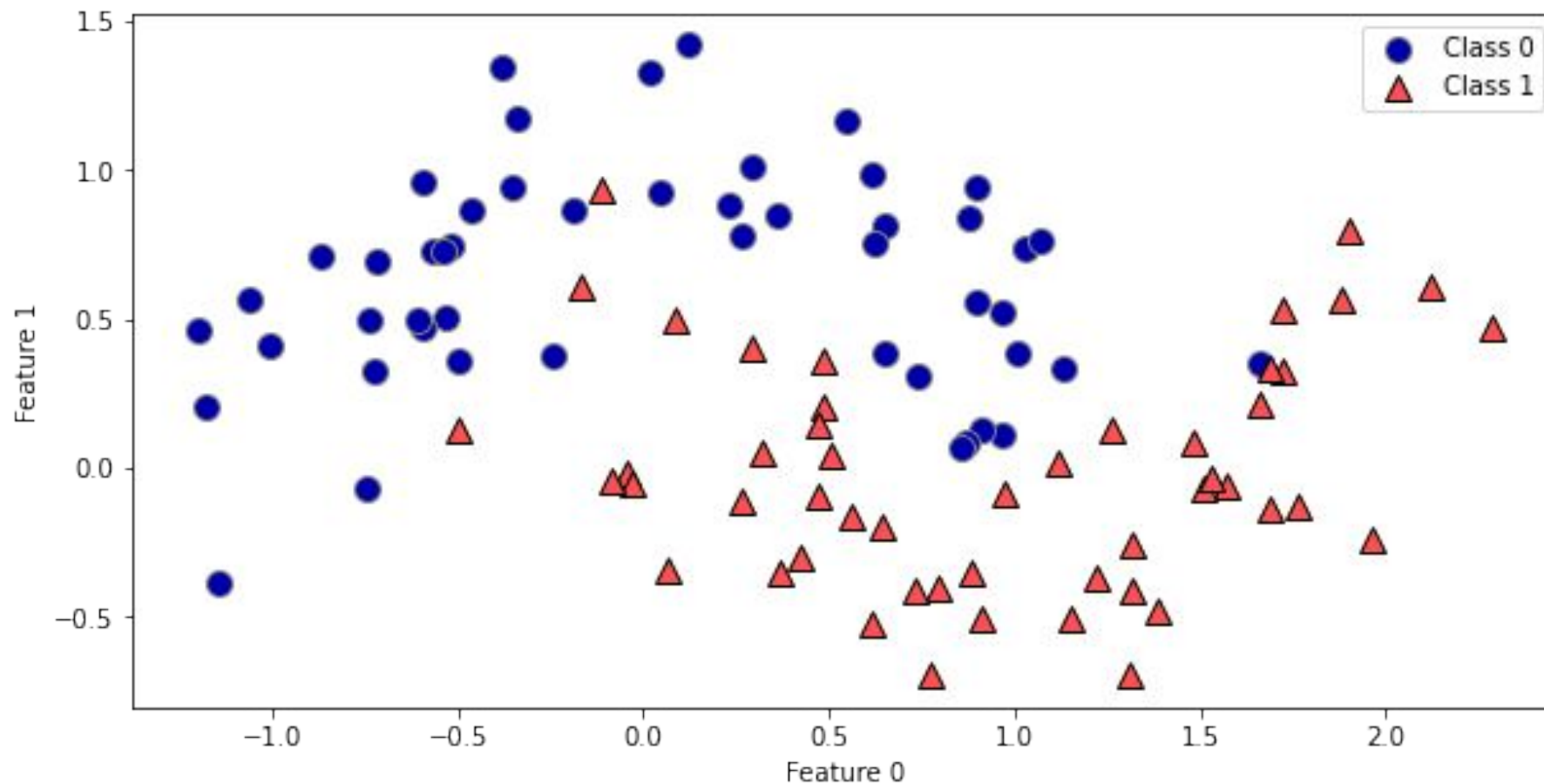




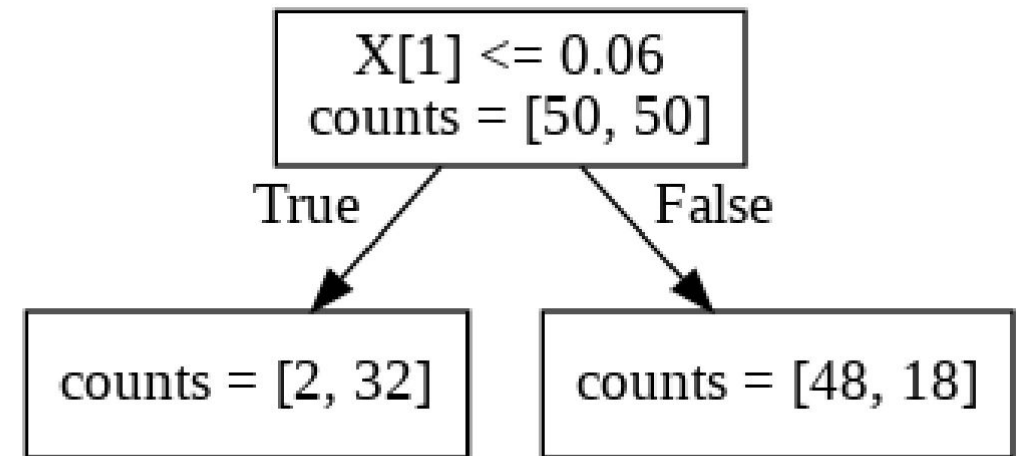
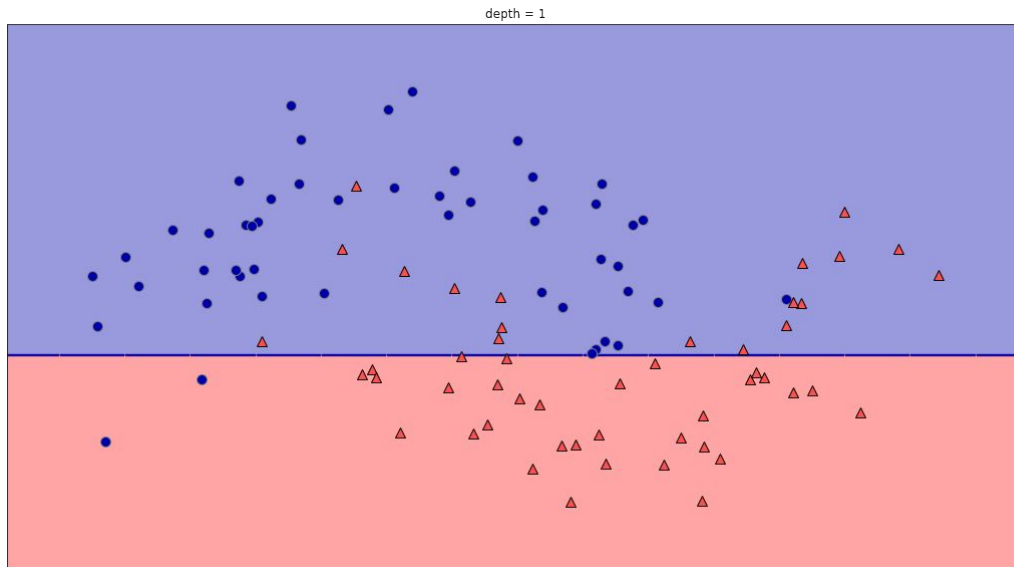
# Drzewo decyzyjne



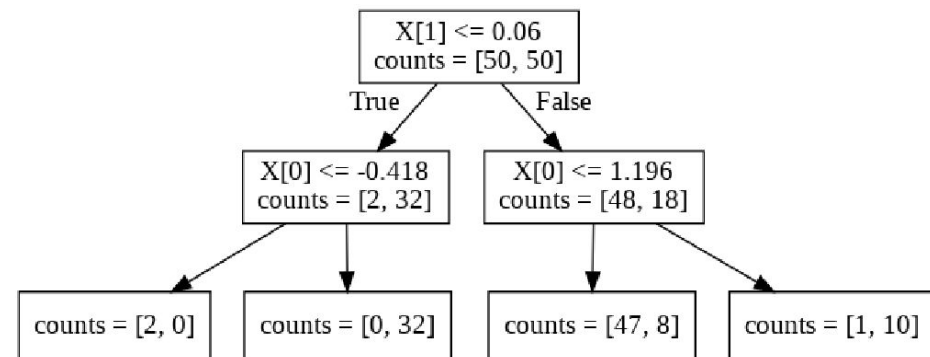
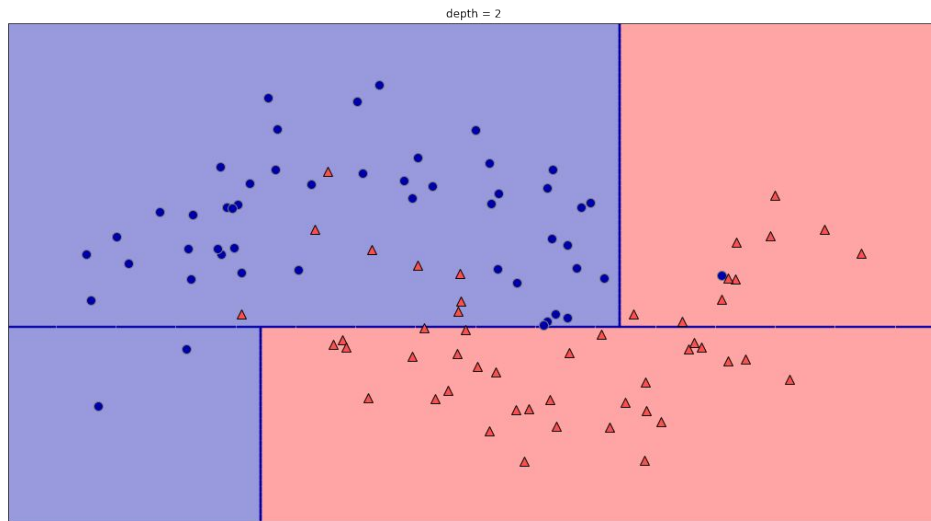
# Drzewo decyzyjne - działanie



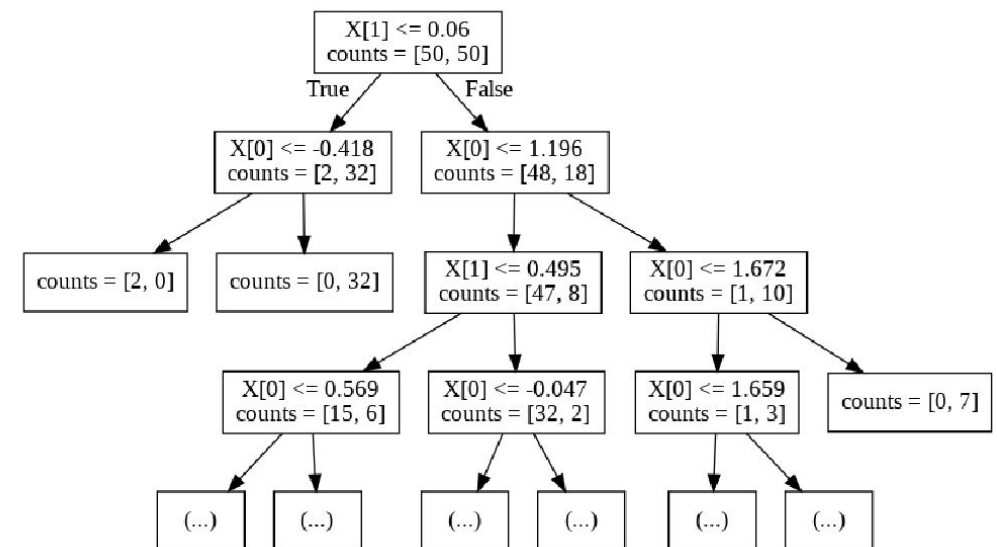
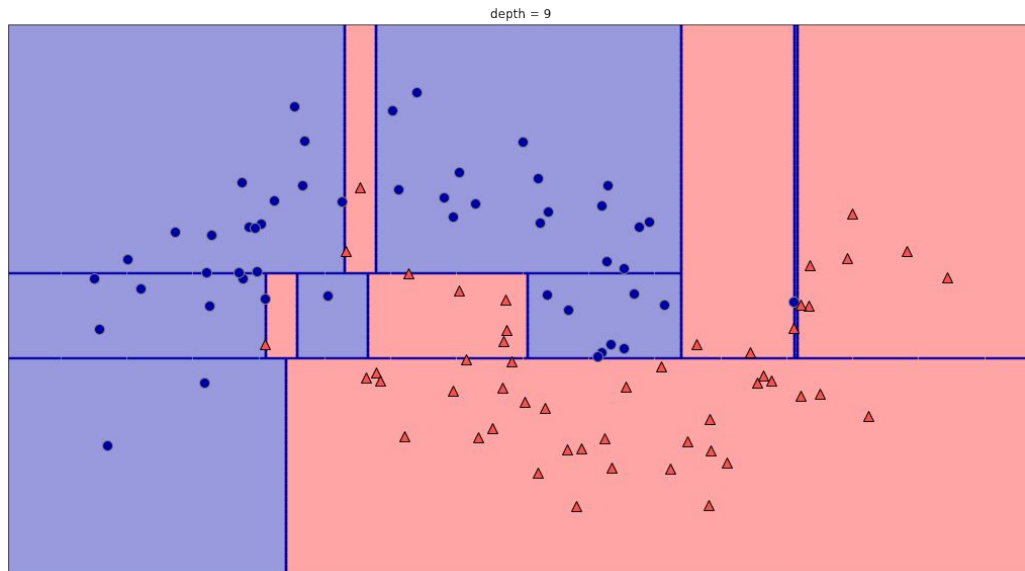
# Drzewo decyzyjne - działanie



# Drzewo decyzyjne - działanie



# Drzewo decyzyjne - działanie





# Jak zbudować drzewo?

- wybór testowanych atrybutów
- wybór wartości
- wybór atrybutu i podziału wartości według **kryterium maksymalnego zysku informacji**
- **drzewo budujemy zaczynając od atrybutu, który niesie najwięcej informacji – umożliwia najlepszy podział na klasy. Jest to korzeń drzewa. Następnie przechodzimy do poziomów niżej, gdzie występują kolejne podziały (węzły), a jeśli już nie ma podziału wskazywana jest klasa wartości (liście)**



# Entropia

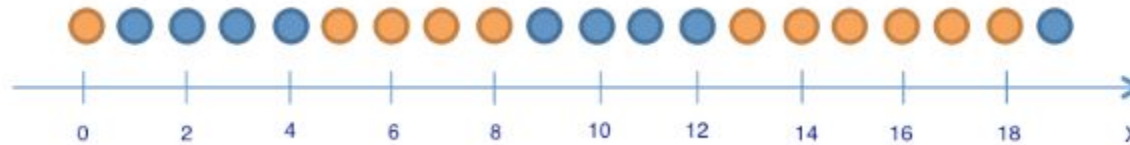


- jedna z metod podziału drzewa decyzyjnego
- miara informacji
- im większa entropia, tym mniejsze uporządkowanie układu
- podział drzew dąży do zmniejszenia entropii
- podział drzew polega na wydzielaniu wartości poszczególnych atrybutów jako węzłów, biorąc najpierw pod uwagę atrybuty najbardziej znaczące

$$S = - \sum_{i=1}^N p_i \log_2 p_i,$$

# Przykład

Mamy zbiór punktów dwóch klas, mających jedną cechę. Aktualnie ich entropia jest równa 1:

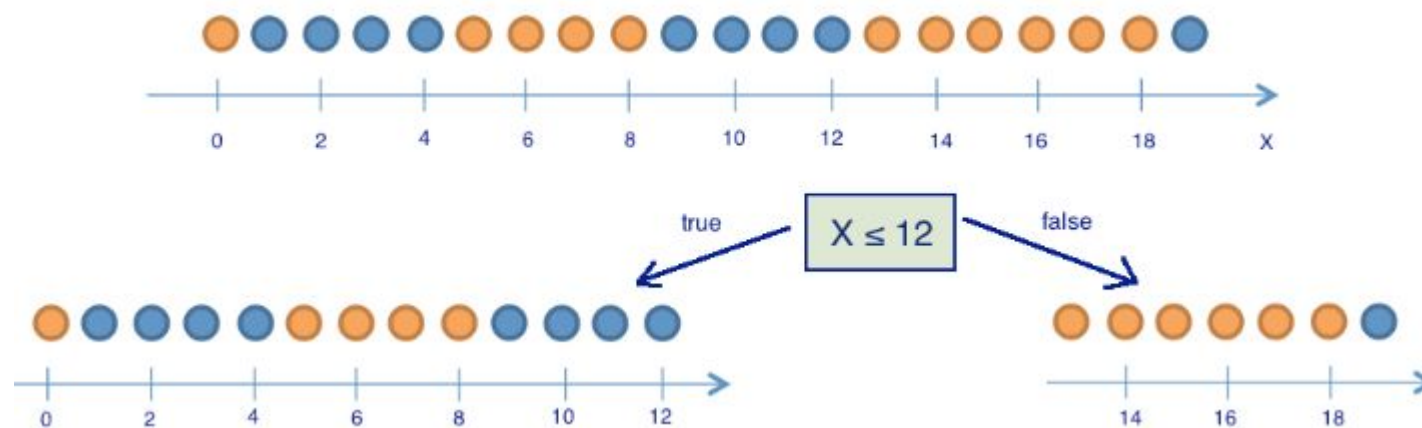


$$S_0 = -\frac{9}{20}\log_2\left(\frac{9}{20}\right) - \frac{11}{20}\log_2\frac{11}{20} \approx 1$$




# Przykład

Po podziale entropia każdego podzbioru maleje:




$$S_1 = -\frac{5}{13} \log_2\left(\frac{5}{13}\right) - \frac{8}{13} \log_2\left(\frac{8}{13}\right) \approx 0.96$$

$$S_2 = -\frac{1}{7} \log_2\left(\frac{1}{7}\right) - \frac{6}{7} \log_2\left(\frac{6}{7}\right) \approx 0.6$$



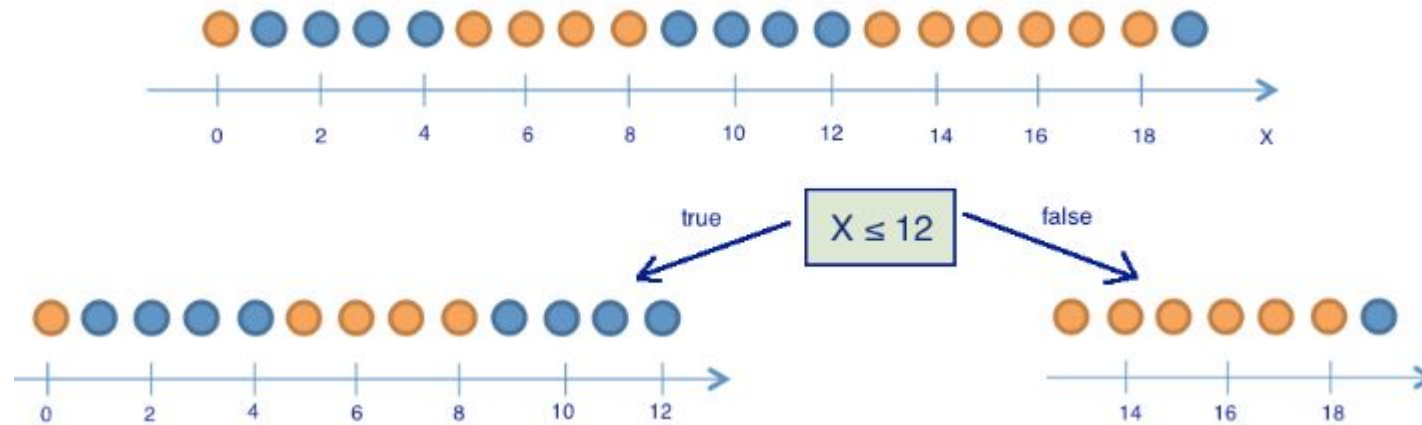
# Information Gain – zysk informacyjny



- Obecnie stosowany do podziału atrybutów w drzewie
- Mierzy ile zyskamy na danym podziale (jak bardzo zmniejszy się entropia)
- Drzewa przy podziałach dążą do maksymalizacji IG

$$IG(Q) = S_O - \sum_{i=1}^q \frac{N_i}{N} S_i,$$

# Przykład



$$S_1 = -\frac{5}{13} \log_2\left(\frac{5}{13}\right) - \frac{8}{13} \log_2\frac{8}{13} \approx 0.96$$

$$S_2 = -\frac{1}{7} \log_2\left(\frac{1}{7}\right) - \frac{6}{7} \log_2\frac{6}{7} \approx 0.6$$

$$IG(x \leq 12) = S_0 - \frac{13}{20} S_1 - \frac{7}{20} S_2 \approx 0.16$$



# Problemy algorytmów budowy drzew



- wybór optymalnego drzewa (głębokość – zależna od wybranych kryteriów)
- zachłanny wybór kryterium maksymalizującego IG (czy jest zawsze najlepszy?)
- metody ograniczania overfittingu:
  - early-stopping (kończenie algorytmu przed osiągnięciem zerowej entropii)
  - przycinanie drzewa – usuwanie gałęzi nie przynoszących dużego zysku informacji



# Zalety i wady drzew decyzyjnych



- + łatwe w interpretacji, wizualizacji, w pełni zrozumiałe reguły działania
- + nie wymagają przygotowania danych (normalizacji, dyskretyzacji)
- + szybkie w użyciu (przy predykcji ma postać zagnieżdżonych instrukcji warunkowych)
- + radzą sobie z danymi kategorycznymi i liczbowymi
  
- problem z przetrenowaniem
- mogą być niestabilne (małe zmiany wartości mogą drastycznie zmieniać predykcje)
- drzewo działa źle przy dużym niezbalansowaniu zbioru danych



# Algorytm ID3



- jeden z algorytmów budowania drzewa
- najstarszy algorytm (1986)
- pojedynczy atrybut w węźle – przeszukiwanie zachłanne
- maksymalizuje Information Gain lub minimalizuje entropię
- Schemat działania:
  1. Oblicz entropię każdego atrybutu w zbiorze danych.
  2. Wybierz atrybut, który najbardziej zmniejszy entropię.
  3. Wykonaj węzeł za pomocą atrybutu.
  4. Powtarzaj wykorzystując pozostałe atrybuty.

PRZYKŁAD



# Algorytm C4.5



- ulepszone ID3
- wartości ciągłe i dyskretne
- w algorytmie ID3 głównym kłopotem był niepotrzebny rozrost drzewa i brak mechanizmów przeciwdziałających zjawisku overfitting-u, co prowadziło do dość wysokiego poziomu błędów dla rzeczywistych danych. Aby tego uniknąć stosuje się tzw. przycinanie (ang. pruning) , w celu zwiększenia generalizacji oceny. Konkretnie działa ono w następujący sposób
  - zaczyna od liści i działa BottomUp
  - mając dany węzeł nie będący liściem i jego poddrzewo oblicza w heurystyczny sposób wartość przewidywanego błędu dla aktualnego poddrzewa.
  - oblicza wartość przewidywanego błędu dla sytuacji, gdyby rozpatrywane poddrzewo zastąpić pojedynczym liściem z kategorią najpopularniejszą wśród liści.
  - porównuje te dwie wartości i ewentualnie dokonuje zamiany poddrzewa na pojedynczy liść propagując tę informację do swych przodków.
- PRZYKŁAD



# Algorytm CART

- Classification and Regression Trees
- drzewa binarne (każdy węzeł ma dwie gałęzie)
- wartości ciągłe i dyskretne
- dostępny w scikit-learn
- Podobny do C4.5, umożliwia obliczanie prawdopodobieństw wyniku lub wartości ciągłych (regresja)





# Główne parametry drzew:

- max\_depth – maksymalna głębokość drzewa
- max\_features – maksymalna liczba cech, wśród których algorytm szuka najlepszego podziału
- min\_samples\_leaf – minimalna liczba przykładów w liście
- criterion – stosowane kryterium podziału (entropia, zysk informacyjny)
- min\_samples\_split – minimalna liczba przykładów wymaganych do podziału w węźle
- itd. dokumentacja



# Czas na praktykę!

Będziemy korzystać z danych, które znajdziesz w zeszycie z drzewami decyzyjnymi.

Możesz go pobrać z sekcji “Dodatkowe materiały do bloku” [tutaj](#).

# Zespoły klasyfikatorów





# Czym są zespoły (ensemble) klasyfikatorów



- algorytmy uczenia zespołów klasyfikatorów tworzą zbiór hipotez
- każda hipoteza (zbiór reguł, tj. wyrażeń “JEŻELI X TO Y”) dokonuje klasyfikacji przykładu
- następnie odbywa się głosowanie hipotez (klasyfikatorów), decydujące o klasyfikacji przykładu
- tworzenie zbioru klasyfikatorów polega na doborze wag i parametrów każdego klasyfikatora

# Głosowanie klasyfikatorów

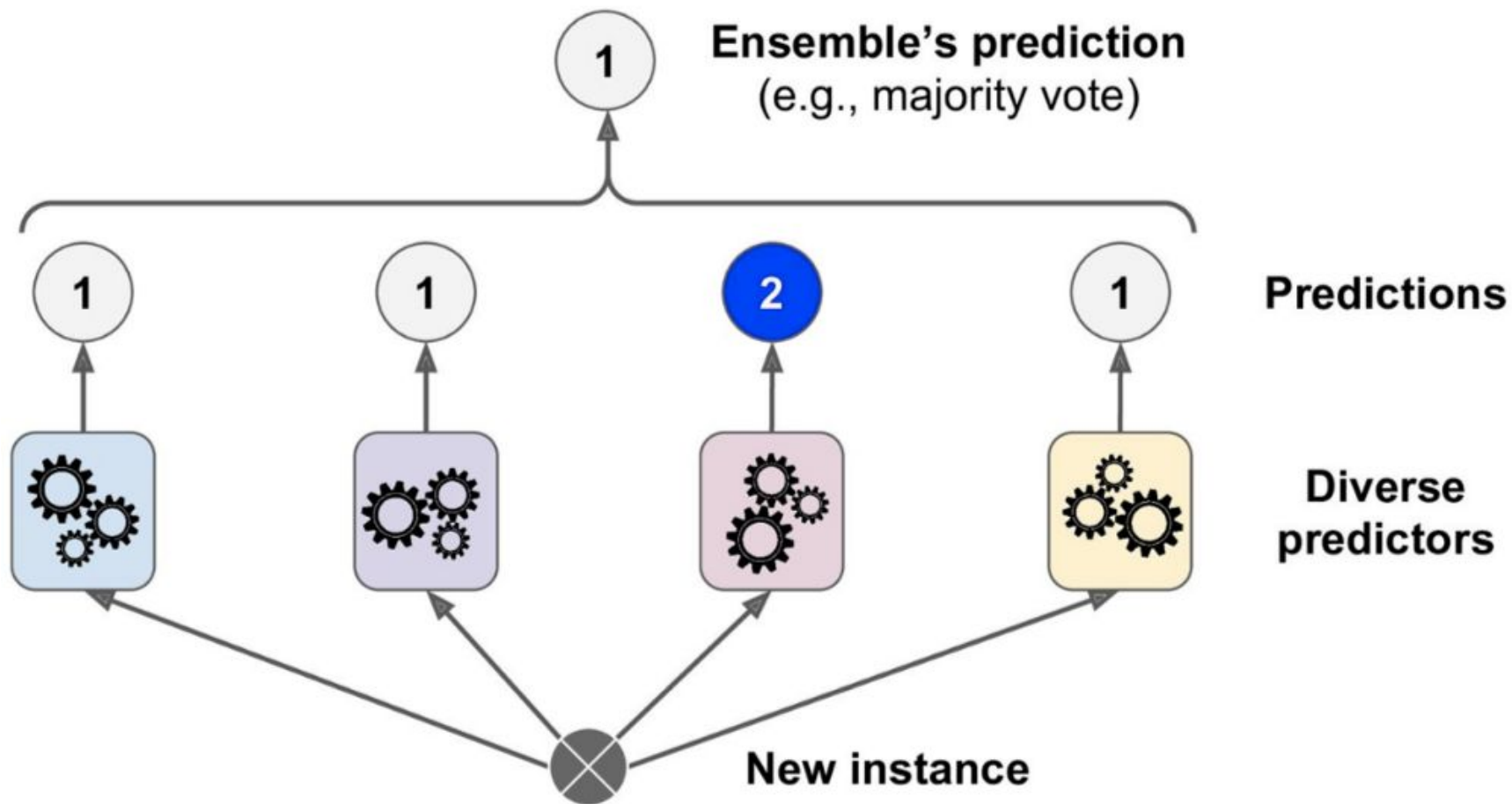
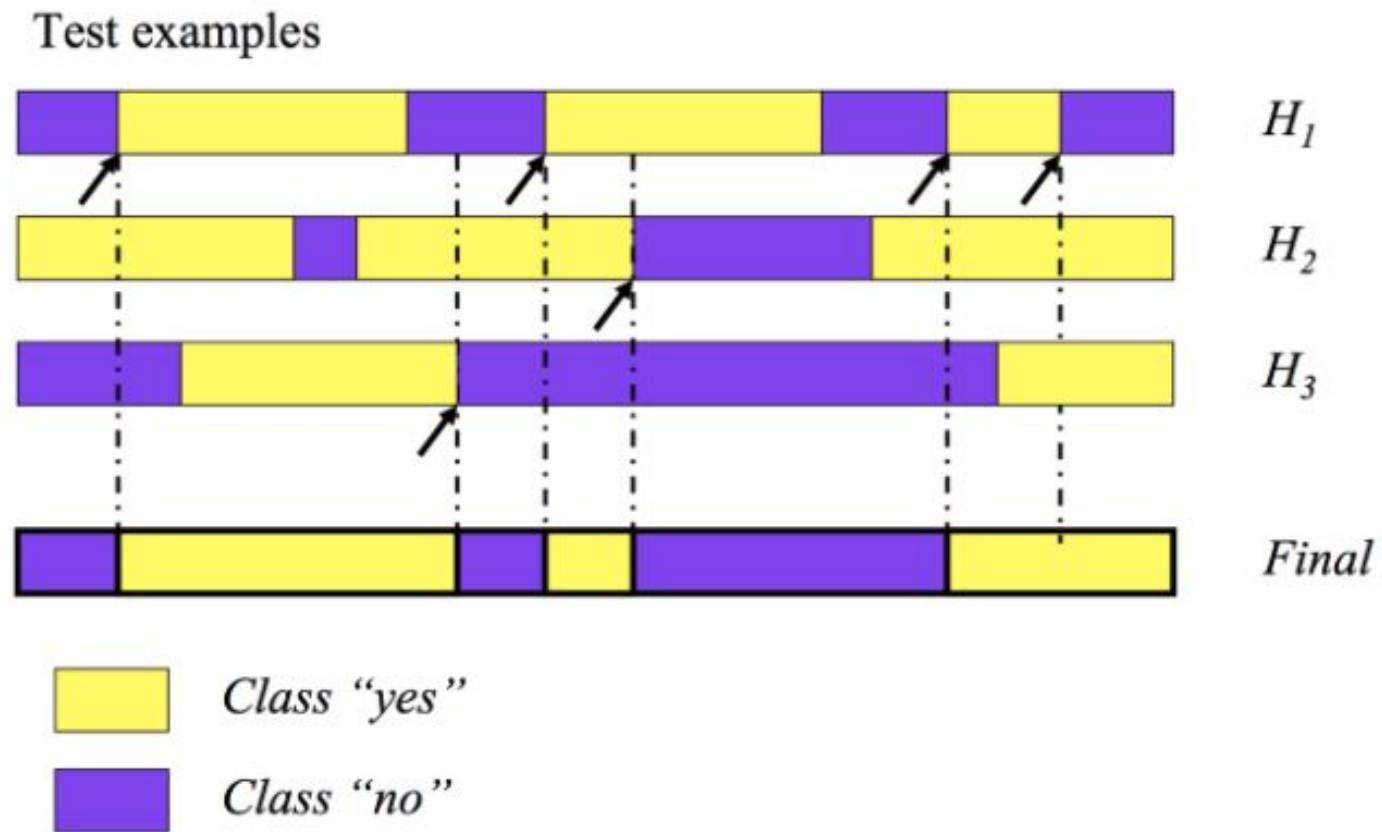


Figure 7-2. Hard voting classifier predictions

# Głosowanie klasyfikatorów





# Wymagania wobec klasyfikatorów

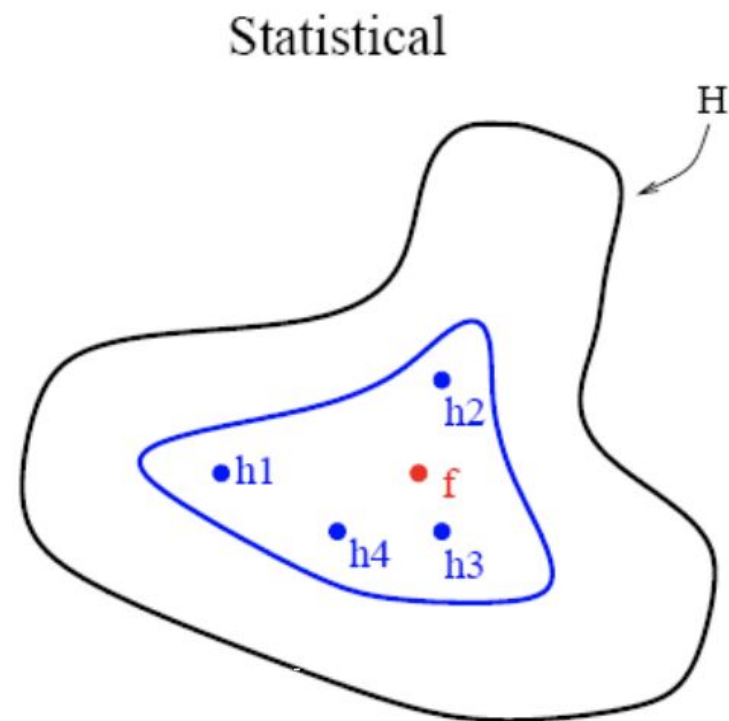
Klasyfikatory w zespole muszą być:

- dokładne – poprawność działania wyższa niż przy losowej odpowiedzi (random guessing)
- zróżnicowane – klasyfikatory mylą się dla różnych przykładów

# Dlaczego zespoły klasyfikatorów?

Główne problemy pojedynczych klasyfikatorów

- statystyczny – przeszukiwanie dużej przestrzeni hipotez  $H$  na podstawie małej ilości danych

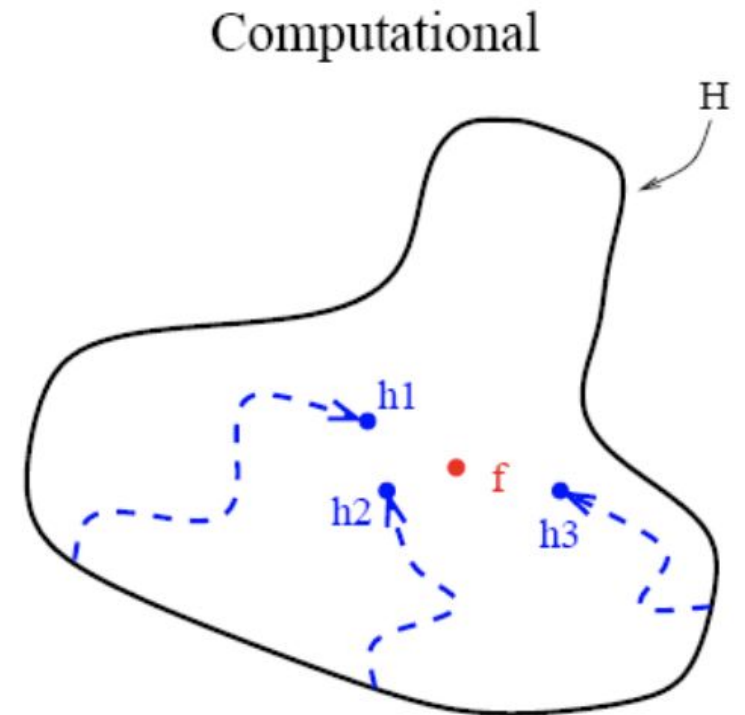




# Dlaczego zespoły klasyfikatorów?

Główne problemy pojedynczych klasyfikatorów:

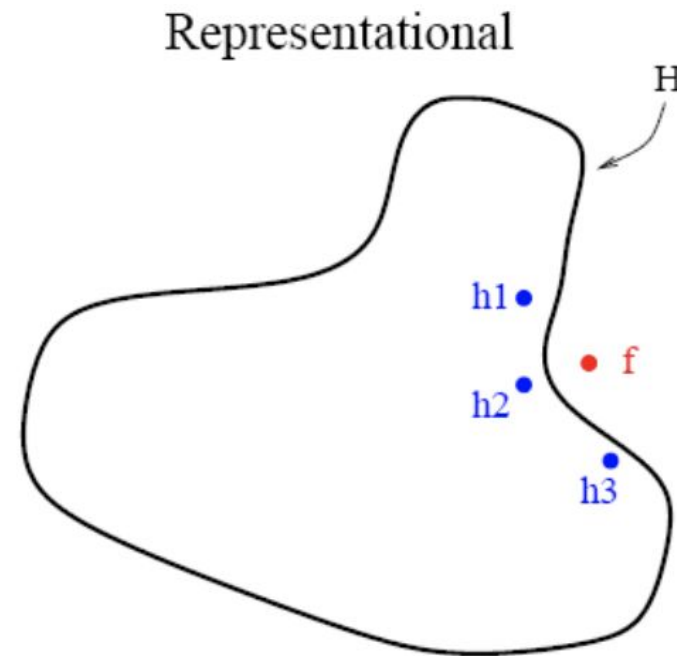
- statystyczny – przeszukiwanie dużej przestrzeni hipotez  $H$  na podstawie małej ilości danych
- obliczeniowy – brak gwarancji znalezienia optymalnej hipotezy (problem optimum lokalnego)



# Dlaczego zespoły klasyfikatorów?

Główne problemy pojedynczych klasyfikatorów:

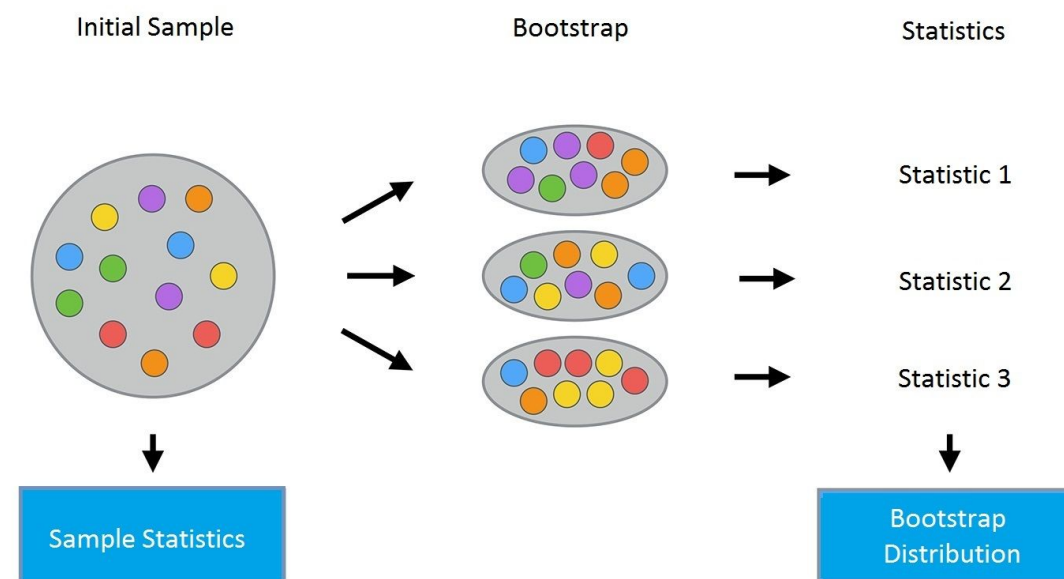
- statystyczny – przeszukiwanie dużej przestrzeni hipotez  $H$  na podstawie małej ilości danych
- obliczeniowy – brak gwarancji znalezienia optymalnej hipotezy (problem optimum lokalnego)
- reprezentacji – przestrzeń hipotez nie zawsze zawiera dobre przybliżenie szukanej funkcji



# Bagging – Bootstrap aggregation

Algorytm oparty na metodzie bootstrappingu:

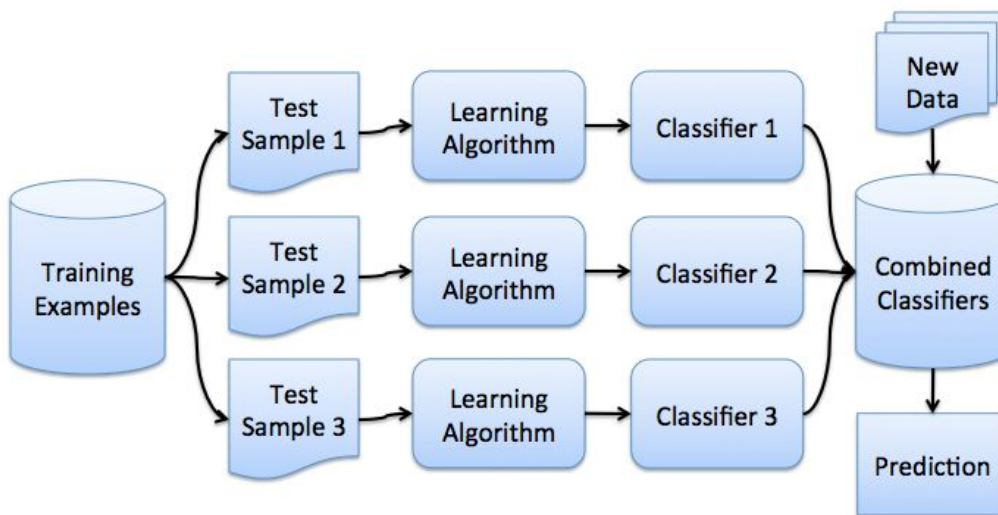
- mamy próbkę o rozmiarze  $N$
- utwórz nową próbkę przez losowanie ze zwracaniem  $N$  elementów
- powtórz procedurę  $M$  razy – otrzymamy  $M$  próbek na podstawie zbioru początkowego
- zmniejszamy w ten sposób wariancję



# Bagging - Bootstrap aggregation

Algorytm oparty na metodzie bootstrappingu:

- trenowanie odrębnych klasyfikatorów dla każdej z próbek uzyskanych w bootstrappingu
- głosowanie przy użyciu klasyfikacji każdego klasyfikatora





# Główne parametry Baggingu

- `base_estimator` – wykorzystywany klasyfikator
- `n_estimators` – liczba klasyfikatorów w zespole
- `max_samples` – liczba próbek w zbiorze każdego klasyfikatora



# Bagging – zalety i wady

- + wysoka dokładność
  - + odporność na obserwacje odstające
  - + niewrażliwe na zmienną wariancję danych
  - + nie wymagają strojenia (działają out-of-the-box)
  - + rzadko się przetrenowują
  - + skalowalne
- 
- nie są interpretowalne jak drzewa
  - gorzej działają dla danych rzadkich
  - duże wymagania pamięciowe



# Las drzew losowych – Random Forest




- Dalsze rozwinięcie Baggingu
- Random subspace method – wybór podzbioru cech dla każdego drzewa (budujemy kilka osobnych drzew, do każdego dajemy tylko określoną liczbę atrybutów – nie wszystkie)
- Dodatkowo: Extremely Randomized Trees może wybierać losowy podzbiór cech w każdym węźle (przydatne gdy mamy do czynienia z overfittingiem)
- w scikit-learn: oparte na uśrednianiu `predict_proba`, a nie samej predykcji



# Główne parametry Random Forest



- `n_estimators` – liczba klasyfikatorów w zespole
  - `max_features` – liczba rozpatrywanych cech (domyślnie pierwiastek z całkowitej liczby cech)
  - parametry drzew decyzyjnych:
    - `max_depth` – maksymalna głębokość drzewa
    - `min_samples_leaf` – minimalna liczba przykładów w liście
    - `criterion` – stosowane kryterium podziału
    - `min_samples_split` – minimalna liczba przykładów wymaganych do podziału w węźle
- 





# Boosting

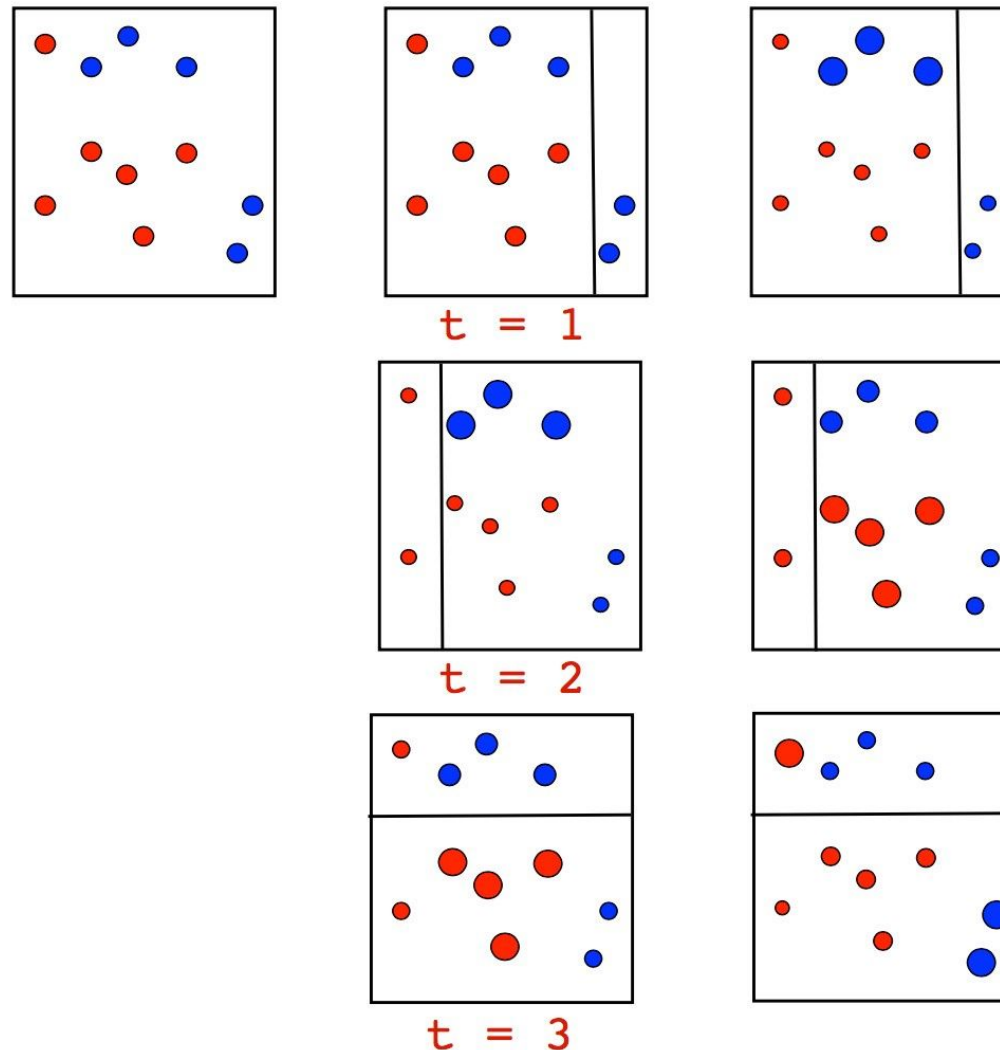


- model budowany iteracyjnie
- podejście zachłanne:
  - pierwszy model trenowany jest na całym zbiorze
  - kolejne modele trenowane są na zmodyfikowanych zbiorach – źle zaklasyfikowane przykłady są rozpatrywane z większą wagą
  - Przykładem jest AdaBoost
- w przeciwieństwie do baggingu, tworzenie zbioru nie polega na losowości, ale zależy od rezultatów poprzednich modeli: każdy nowy podzbiór zawiera te przykłady, które były źle klasyfikowane przez poprzednie modele

# AdaBoost – trenowanie

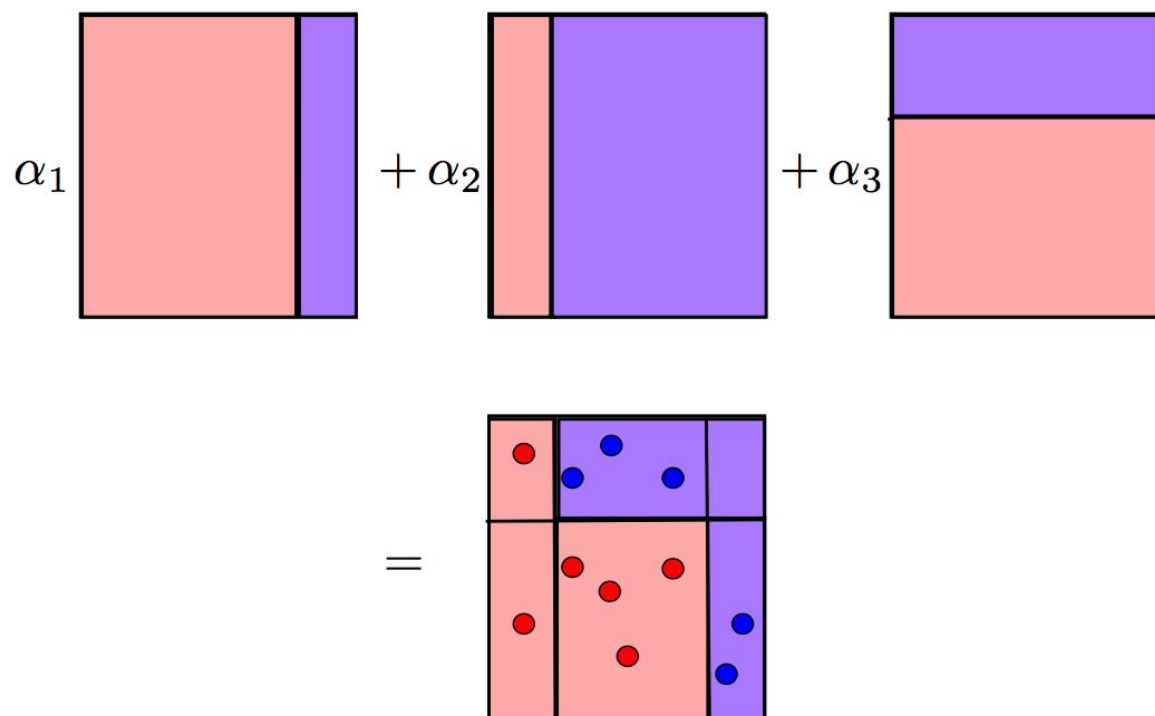
Poszczególne poziomy prezentują iteracje.

Wielkość próbek odzwierciedla wagi w danej iteracji.



# AdaBoost – predykcja

Modele z kolejnych iteracji nakładają się na siebie:





# Główne parametry AdaBoost

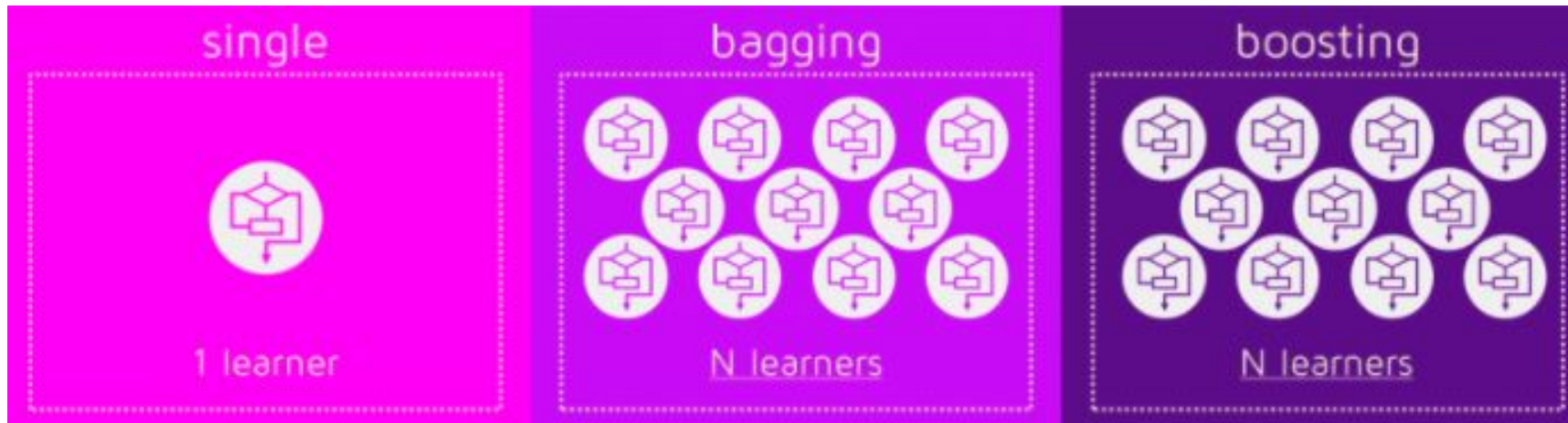
- `base_estimator` – wykorzystywany klasyfikator
- `n_estimators` – liczba estymatorów (maksymalna)
- parametry używanego klasyfikatora



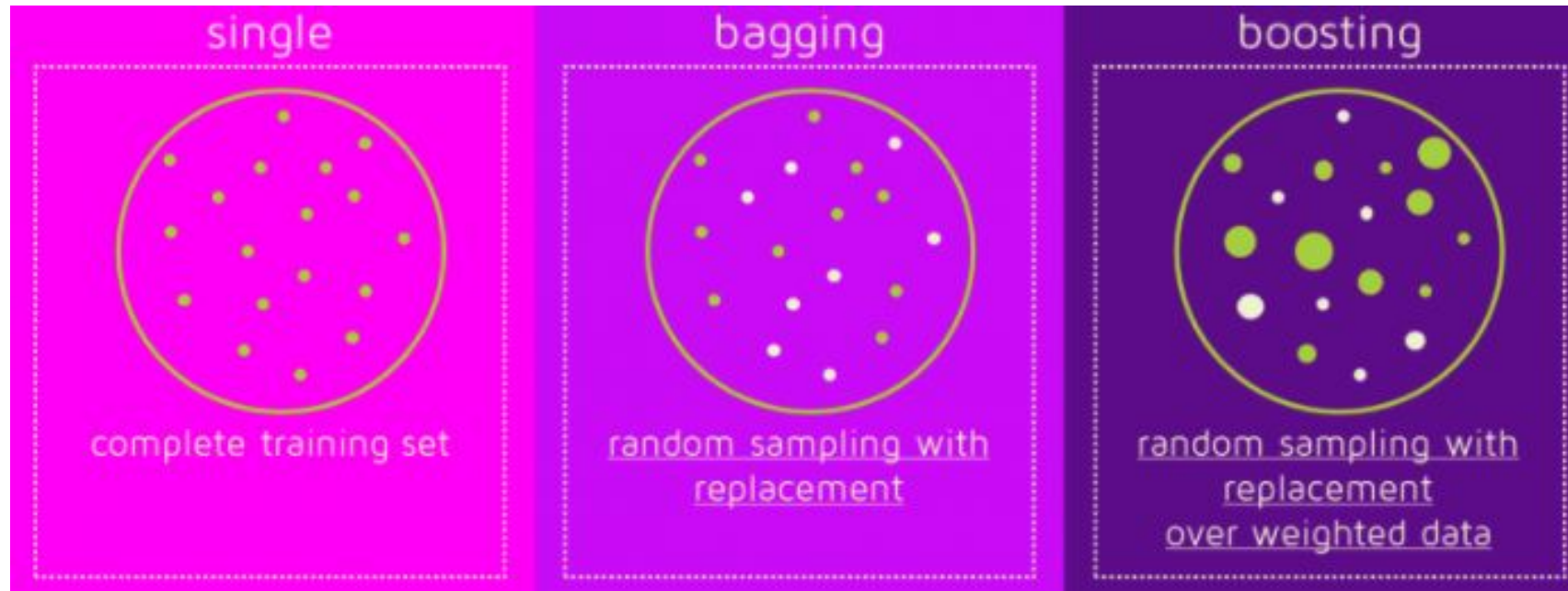
# Zalety i wady AdaBoost

- + wysoka poprawność działania
- + prostota algorytmu
- ryzyko overfittingu dla błędnie zaetykietowanych danych
- brak pełnego zrozumienia zasady działania

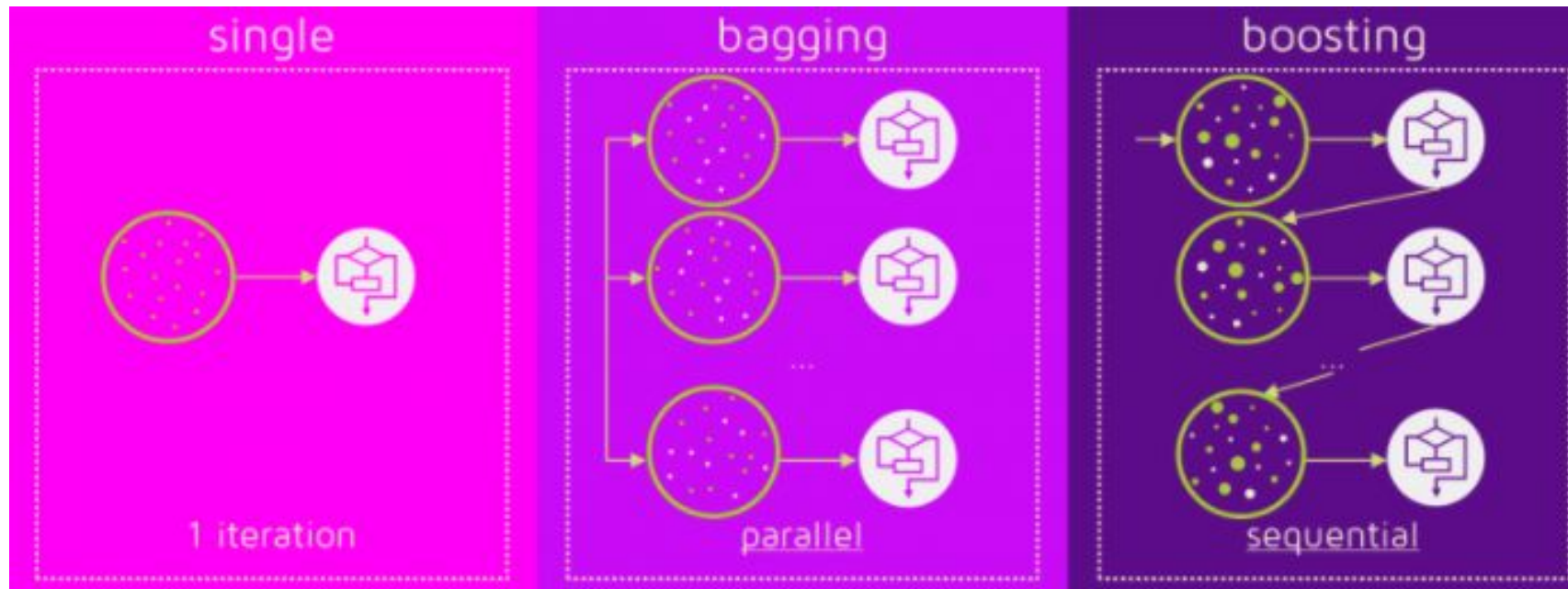
# Podsumowanie



# Podsumowanie

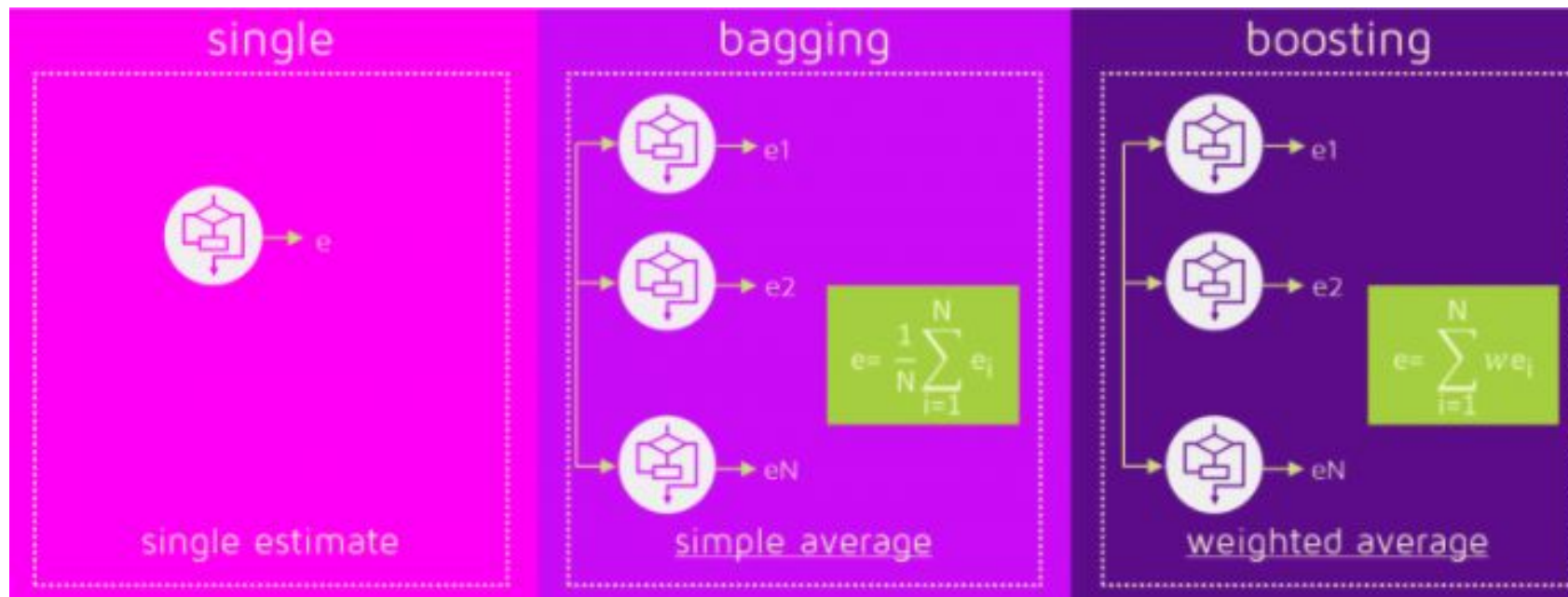


# Podsumowanie





# Podsumowanie





# Implementacja w scikit-learn

<https://machinelearningmastery.com/ensemble-machine-learning-algorithms-python-scikit-learn/>



# Czas na praktykę!

Będziemy korzystać z danych, które znajdziesz w zeszycie z **zespołami klasyfikatorów**.

Możesz go pobrać z sekcji  
“Dodatkowe materiały do bloku” [tutaj](#).



# Czas na praktykę!

Będziemy korzystać z danych, które znajdziesz w zeszycie z **dodatkowymi materiałami** oraz z załącznika **Transformed Data Set**.



# Polecane materiały związane z tematyką bloku



- [Dokumentacja sklearn](#)
- Dane: <https://github.com/matzim95/ML-datasets>
- [Klasyfikacja – kompletny przykład](#)
- [Drzewa decyzyjne - objaśnienie](#)
- [SVM - wykład MIT](#)