

## Uczenie maszynowe - laboratoria 2

Klasyfikacja, Regresja logistyczna, Neuron, Sieć neuronowa, Uczenie sieci neuronowej

Dawid Wiśniewski

14 Stycznia 2017

# Plan zajęć

1 Przypomnienie - regresja liniowa/wielomianowa

2 Regresja logistyczna

3 Sieci neuronowe

4 Sieć wielowarstwowa

## Potrzebne narzędzia do pobrania

Wejdź do konsoli (cmd) i uruchom `conda update scikit-learn`

# Przypomnienie - Regresja liniowa/wielomianowa

- Model regresji liniowej :  $y = ax + b$
- Model regresji wielomianowej :  $y = a_1x^1 + a_2x^2 + \dots + a_nx^n + b$
- **Regresja liniowa/wielomianowa służy do przewidywania zmiennych liczbowych**
- Funkcja kosztu – błąd średniokwadratowy
- Uczenie – spadek gradientowy

# Klasyfikacja

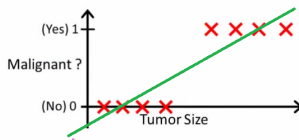
Chcemy stworzyć model, który sprawdzi, czy warto kupić dany samochód (model zwróci wartość TAK, jeśli warto lub NIE, jeśli nie warto kupować danego egzemplarza.)

Czy warto kupić zadany samochód			
Przebieg	moc silnika	...	Decyzja (TAK/NIE)
120 000	54	...	TAK
214 000	90	...	NIE
412 000	84	...	NIE
32 100	170	...	TAK
117 000	60	...	???

Jak w takim przypadku otrzymać odpowiednią etykietę ?

## Regresja liniowa do klasyfikacji ?

Przeanalizujemy przykład w którym mamy tylko jedną cechę – rozmiar guza. Różne rozmiary guzów mapujemy na wartości 1 (guz złośliwy), 0 (guz niezłośliwy). Wygenerujmy prostą regresji (zielona linia) :



Moglibylibyśmy ustawić próg na osi Y, powyżej którego uznamy, że guz jest złośliwy, a w przeciwnym przypadku łagodny. Regresja liniowa, reprezentowana przez

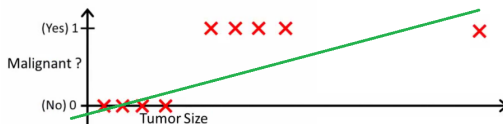
$$y = h(x) = ax + b \quad (1)$$

Może zostać zamieniona na klasyfikator poprzez progowanie :

$$y = \begin{cases} \text{zlosliwy}, & \text{if } h(x) > 0.5. \\ \text{lagodny}, & \text{w przeciwnym przypadku.} \end{cases} \quad (2)$$

## Regresja liniowa do klasyfikacji ? - cd

Jednak takie rozwiązanie będzie mocno zależne od przedstawionych danych treningowych. Wyobraźmy sobie, że dodamy jeszcze jeden przypadek bardzo dużego guza :



Po dodaniu nowego przypadku, prosta regresji zmieniła wartości parametrów i teraz optymalnym progowaniem byłoby :

$$y = \begin{cases} \text{złotliwy}, & \text{if } h(x) > 0.2. \\ \text{lagodny}, & \text{w przeciwnym przypadku.} \end{cases} \quad (3)$$

Nie powinniśmy drastycznie zmieniać hipotezy za każdym razem, kiedy widzimy nową obserwację.

Ponadto – regresja liniowa zwraca nam wartości rzeczywiste, które nie są w żaden sposób ograniczone. Wnioskowanie o przynależności do klas w takich przypadkach jest utrudnione.

## Regresja liniowa do klasyfikacji ? - cd

Co powinien zrobić dobry klasyfikator :



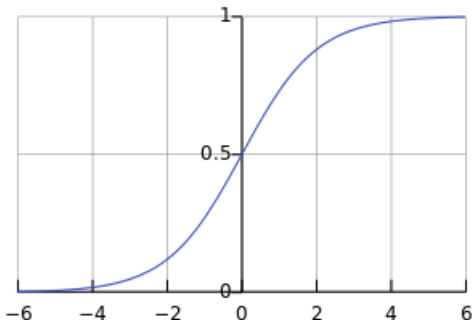
Dobry klasyfikator powinien stworzyć taką reprezentację, w której zauważy, że wszystko po prawej stronie fioletowej linii - jest złośliwe, wszystko po lewej zaś - niezłośliwe. Taka reprezentacja hipotezy uniezależni nas od dodanych nowych przypadków odstających (granica pozostanie poprawną granicą pozostając w tym samym miejscu).

### Granica decyzyjna

Hiperpłaszczyzna rozdzielająca klasy w przestrzeni atrybutów nazywana jest granicą decyzyjną.



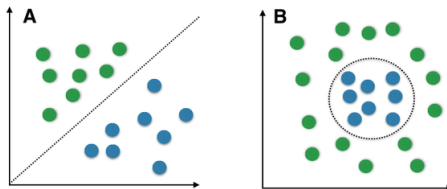
# Regresja logistyczna



$$P(y = 1|\vec{x}) = \frac{1}{1 + e^{-(\vec{w}^T \vec{x} + b)}} \quad (4)$$

# Cechy regresji logistycznej

- Prosta metoda klasyfikacji
- Polegająca na "opakowaniu" wyrażenia  $y = \vec{w}^T \vec{x} + b$  funkcją sigmoidalną
- generująca liniową granicę decyzyjną
- Wartości generowane na wyjściu funkcji są zawsze z zakresu od 0..1
- Wartości generowane na wyjściu funkcji mogą być interpretowane jako prawdopodobieństwo przynależności do jednej z dwu klas.
- Dobrze sprawdzi się w przypadku, kiedy klasy są liniowo separowalne, nie zadziała w przypadku problemów nieliniowych (patrz obrazek)



Regresja logistyczna dobrze sprawdzi się w przypadku problemu A, w przypadku problemu B nie poradzi sobie (może wygenerować w tym przypadku jedynie granicę decyzyjną będącą linią prostą)

# Uczenie regresji logistycznej

Wyznaczenie optymalnych wartości wag regresji logistycznej odbywa się, podobnie jak w przypadku regresji liniowej – w procesie uczenia.

Podobnie też jak poprzednio mamy tutaj dwie fazy :

- Określenia wartości funkcji kosztu – np.  $\sum_{i=1}^N \log(1 + e^{-y_i h(x)})$
- Iteracyjne poprawianie wag używając gradientu po nowej funkcji kosztu (identycznie jak na poprzednich zajęciach)

## Przypadek wieloklasowy

Nie zawsze jednak nasze dane mają tylko dwie etykiety, czasami chcielibyśmy nauczyć się większej ich ilości.

Chcemy np. zaklasyfikować obiekty do jednej z 3 kategorii : Pies, Kot, Koń

W tym celu tworzymy 3 klasyfikatory, które sprawdzają :

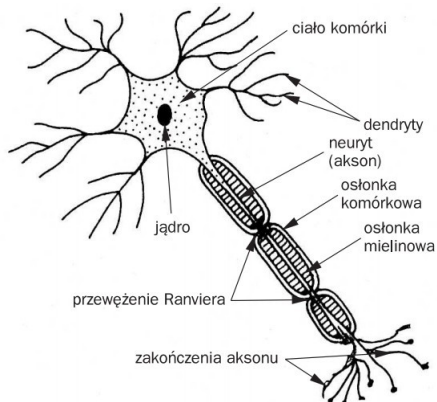
- czy to pies, czy to nie jest pies
- czy to kot, czy to nie jest kot
- czy to koń, czy to nie jest koń

Gdzie jako nie-psy uważamy wszystkie koty i konie.

Każdy z klasyfikatorów zwróci nam prawdopodobieństwo tego, że obiekt jest odpowiednio psem, kotem, koniem. Ostateczną klasę wybieramy poprzez wybór klasy, dla której klasyfikator zwrócił największe prawdopodobieństwo sukcesu.

# Ludzki mózg

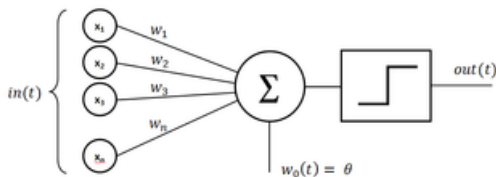
Ludzki mózg składa się z około  $10^{11}$  neuronów i około  $10^{14}$  połączeń między nimi.



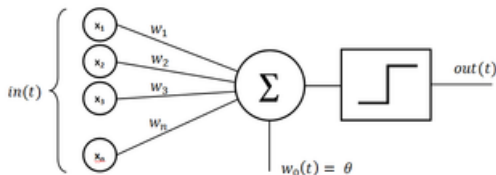
**Schemat budowy neuronu**

# Mózg komputerowy

A może by tak odtworzyć sposób działania mózgu używając komputera ?  
Model przedstawiony na rysunku poniżej nazywamy perceptronem – został on wprowadzony w późnych latach 50 przez Rosenblatta jako model działania ludzkiego neuronu.



# Zasada działania



Jak widzmy na rysunku, pojedynczy perceptron(neuron) przyjmuje  $n$  informacji na wejściu (kółka po lewej). Każdą z tych informacji możemy traktować jako cechę, mówiącą nam np. o wzroście danego człowieka/wieku samochodu itp. (jak w poprzednio omawianych przykładach) Ponadto każdy neuron posiada dodatkową wagę – bias, która działa jak współczynnik "b" w regresji liniowej i logistycznej. Waga ta nie jest podpięta pod żadną istniejącą cechę i przyjmuje się, że wielkość tej wagi jest zawsze mnożona przez 1 (sztuczna cecha stworzona w celu uproszczenia obliczeń).

Aby obliczyć wartość wyjściową neuronu, należy zagregować wartości cech wejściowych uwzględniając wagi (kółko ze literą  $\Sigma$  z obrazka), co wyznaczyć możemy poprzez wzór :

$$y = \sum_{i=1}^n w_i x_i + b = \vec{w}^T \vec{x} + b \quad (5)$$

(gdzie  $n$  to liczba cech na wejściu perceptronu),  
a następnie zsumowaną wartość przekazać do funkcji aktywacji  $\sigma(x)$ , która decyduje o natężeniu wyjścia perceptronu, otrzymując wynik :

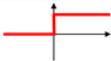
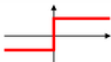

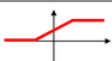


$$y = \sigma(\vec{w}^T \vec{x} + b) \quad (6)$$

Jest to działanie analogiczne do działania ludzkiego neuronu, który "zbiera" wzbudzenia od przyłączonych sąsiadów, a następnie w zależności od tego, czy potencjał elektryczny jest wystarczająco silny – decyduje czy i jakie napięcie wygenerować na swoim "wyjściu".



# Rodzaje funkcji aktywacji

Istnieje wiele funkcji aktywacji  $\sigma(x)$ , których możemy użyć w naszym perceptronie :

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer NN	

Jedną z tych funkcji jest sigmoida, która sprawi, że nasz perceptron zwróci wartość wyjściową równą :

$$y = \frac{1}{1 + e^{-(\vec{w}^T \vec{x})}} \quad (7)$$

Co jest tożsame z poznaną wcześniej regresją logistyczną(!!!).

Model regresji logistycznej jest więc jednym z rodzajów neuronów.

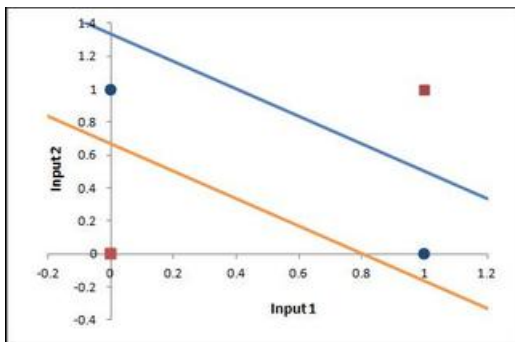
## Po co łączyć neurony w sieć ?

Pojedynczy neuron generuje (podobnie jak regresja logistyczna) "liniową granicę decyzyjną". Są jednak problemy, w których uwzględnienie nieliniowości jest wymagane w celu rozwiązania ich.

Złożenie perceptronów pozwala na sprawienie, że problem rozbity zostanie na serię podproblemów, z których każdy można rozwiązać generując odpowiednią liniową granicę. Aby to wyjaśnić prześledźmy przykład problemu nieliniowego - XOR :

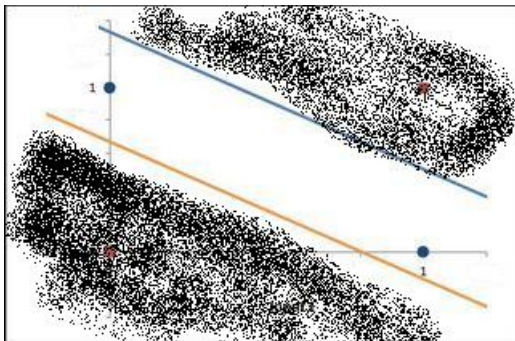
XOR		
Input 1	Input 2	XOR
0	0	0
1	0	1
0	1	1
1	1	0

Poniższy obrazek obrazuje problem XOR. w problemie tym mamy dwa wejścia, które przekładają się na wartość wyjścia wg. powyższej tabelki.



Nie istnieje pojedyncza prosta, która dobrze rozdzieli te dwie klasy.

Musielibyśmy użyć osobno dwóch linii, a następnie połączyć odpowiednio informacje z nich, aby wyznaczyć przestrzeń, w której wartość przyjmie wartość 1 (niezaciemiony obszar na obrazku).



Użycie dwóch osobnych neuronów pozwoli nam na stworzenie takiej przestrzeni :

	input 1	input 2	output			input 1	input 2	output
object 1	0	0	0			object 1	0	0
object 2	0	1	1			object 2	0	0
object 3	1	0	1			object 3	1	0
object 4	1	1	1			object 4	1	1

	input 1	input 2	output
object 1	0	0	0
object 2	1	0	1
object 3	1	0	1
object 4	1	1	0

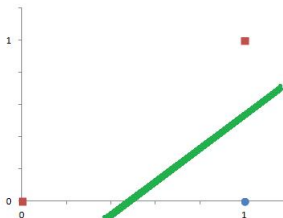
Górna lewa tabelka reprezentuje neuron, który odpowiada pomarańczowej granicy decyzyjnej.

Górna prawa tabelka reprezentuje neuron, który odpowiada niebieskiej granicy decyzyjnej.

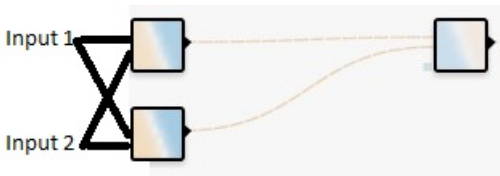
Jeśli stworzymy nowy neuron, który pobierze wyjścia z obu wcześniejszych neuronów, to jego tablica prawdy wyglądać będzie tak, jak trzecia tabelka (na dole).

Zauważmy, że problem z dolnej tabelki jest już separowalny liniowo !

Poniżej wizualizacja problemu w przestrzeni przetransformowanej przez poprzednie neurony :



Poniżej wizualizacja sieci wielowarstwowej rozwiązującej ten problem (jedna warstwa ukryta z dwoma neuronami transformującymi wejściową przestrzeń cech) :



# Uczenie wielowarstwowej sieci neuronowej

Uczenie wielowarstwowej sieci odbywa się kilkunastkowo :

- Najpierw oblicza się wyjścia poszczególnych warstw zaczynając od pierwszej, kończąc na warstwie wyjściowej
- Po obliczeniu aktualnych wartości na warstwie wyjściowej oblicza się błąd sieci neuronowej
- używając algorytmu propagacji wstecznej (backpropagation) przydziela się poszczególnym wagom odpowiednią "winę" za wygenerowany błąd w zależności od udziału konkretnej wagi w stworzeniu tego błędu. Algorytm zaczyna się od ostatnich warstw sieci, propagując błąd w kierunku warstw początkowych. Modyfikacje wag sprawiają, że sieć jest coraz lepiej dostrajana do problemu.

Przykład użycia propagacji wstecznej :

<https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>

