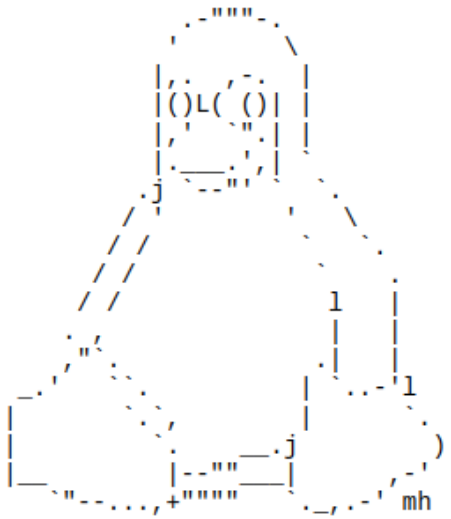

Uczenie Maszynowe

Teoria

Joe Mama Mike Hawk Nick Ger Hugh Jass Bic Didz Mike Oxlong
Geega Neega Sleepy “Slippy” Joe (oceń w końcu kolosy) TrumpGPT

June 29, 2023

Abstract

Przygotujcie się na niezapomnianą przygodę z kursem z uczenia maszynowego. Kurs ten zabierze Was w podróż po klasycznych algorytmach uczenia maszynowego, które potrafią zrobić niesamowite rzeczy, takie jak regresja, klasyfikacja, klastrowanie, redukcja wymiarów i wyszukiwanie wzorców. Poznacie również tajemnice sieci neuronowych i ich różnorodne typy. Kurs ten uświadomi Wam również podstawowe problemy, które mogą Wam się nawarzyć podczas uczenia maszynowego i oświeci ich matematyczne źródła. Będziecie w stanie wybrać odpowiedni algorytm uczenia maszynowego w zależności od konkretnego zadania i dostępnych danych. Ponadto, kurs ten obejmuje tematy związane z walidacją modelu i rozwiązywaniem typowych kłopotów, które mogą Wam się przyplątać podczas trenowania modelu. Zdobędziecie umiejętność przeprowadzania walidacji modelu oraz rozwiązywania występujących bolączek.

Contents

1 Rodzaje uczenia maszynowego

Podziały ze względu na:

- nadzór:
  - **uczenie nadzorowane** - trzeba etykietować zbiór uczący
  - **uczenie nienadzorowane** - nie trzeba etykietować zbioru uczącego (uczenie bez nauczyciela)
  - **częściowo nadzorowane** - część danych jest etykietowana, część nie
  - **uczenie przez wzmacnianie** - polega na dawaniu kar i nagród za konkretne wybory, które algorytm uwzględnia przy kolejnych próbach
- sposób uogólnienia:
  - **instancje (np. KNN, drzewa)** - uczenie na pamięć, szuka najbardziej podobnego do pytanego obiektu i podpisuje go tak samo według poznanych zasad, przykład: K-nearest neighbors; przede wszystkim wielkość modelu jest nieznana przed rozpoczęciem uczenia
  - **model (np. sieci neuronowe, regresja logistyczna, SVM)** - szuka zależności między wartościami i na ich podstawie dobiera parametry funkcji, na podstawie której potem przewiduje wartość; wielkość modelu jest znana przed rozpoczęciem uczenia
- dostęp do danych:
  - **batch** - posiadamy dostęp do kompletnego zbioru danych, jeśli możemy trzymać cały zbiór danych i dane nie zmieniają się zbyt często
  - **online (mini-batches)** - karmimy model ciągle małymi porcjami danych, jeśli nie mamy miejsca na utrzymanie całego zbioru lub dane ciągle się zmieniają

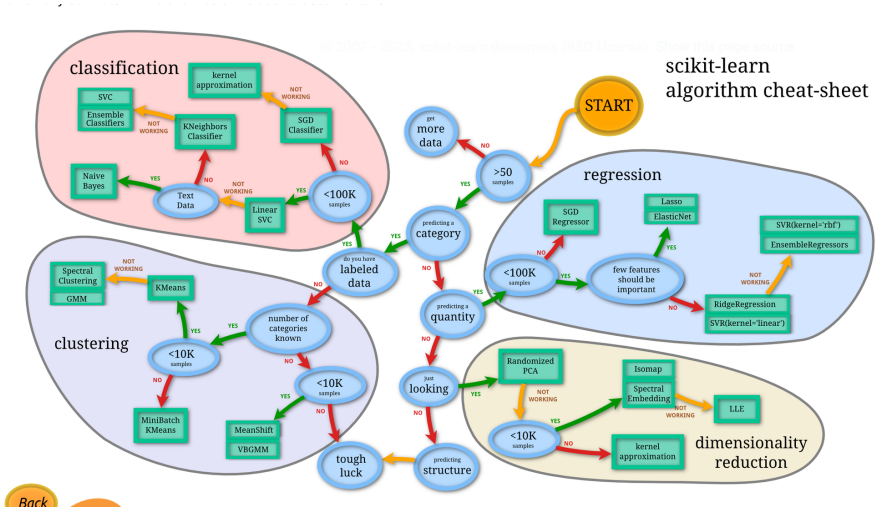

Figure 1: mapka wyboru algorytmu

2 Metryki

2.1 *Klasyfikacja*

- Confusion Matrix

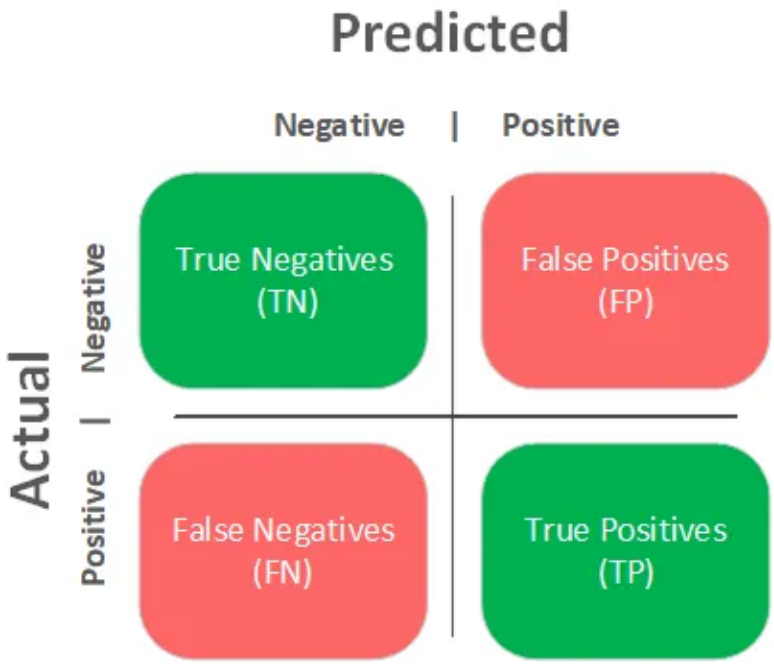

Figure 2: Confusion Matrix

2.1.1 Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- Classification Error
  - Wyraża jaka część instancji została dobrze sklasyfikowana

2.1.2 Precision

$$Precission = \frac{TP}{TP + FP}$$

- Stosowana gdy wymagamy od modelu wysoką wartość *True Positives* i chcemy zminimalizować liczbę *False Positives*
- Proporcja *True Positives* do *sumy True Positives i False Positives*

2.1.3 Recall (True Positive Rate, Sensitivity, Probability of Detection)

$$Recall = \frac{TP}{TP + FN}$$

- Stosowana gdy wymagamy od modelu wysoką wartość *True Positives* i chcemy zminimalizować liczbę *False Negatives*

- Proporcja *True Positives* do *sumy True Positives i False Negatives*

2.1.4 F1-score

$$F1 - score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

- Metryka stosowana do porównywania modeli.
- Korzystne dla modeli z podobną wartością *Precision* i *Recall*.
- Średnia harmoniczna obu wartości.

2.1.5 Kompromis Precision/Recall

- *Precision* zmniejsza *Recall* i vice versa.

2.2 *Regresja*

Mean square error (błąd średnio kwadratowy):

$$MSE(x, y) = \frac{1}{n} \sum_{i=1}^{n} (f(x_i) - y_i)^2$$

- Błąd średnio-kwadratowy, najczęściej stosowany w przypadku regresji liniowej
- Stosowana ogólnie w regresjach
- Gdy funkcja f jest różniczkowalna, to MSE jest różniczkowalny ze względu na parametry funkcji f
- Równoważna z normą l_2 (Norma Euklidesowa)

Mean absolute error (błąd średnio bezwzględny):

$$MAE(x, y) = \frac{1}{n} \sum_{i=1}^{n} |f(x_i) - y_i|$$

- Błąd średniego odchylenia wartości bezwzględnej
- Stosowana ogólnie w regres
- Stosowane gdy jest dużo *outlier'ów* w zbiorze
- Równoważna z normą l_1 (Norma Manhattan)

Entropia:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i)$$

- Wyraża ilość informacji, którą możemy uzyskać po otrzymaniu instancji ze zbioru
- Tworzy zbilansowane drzewa
- Tak dzielimy zbiór tworząc drzewa, aby zysk entropii był jak największy (dowiadujemy się najwięcej dzieląc w ten sposób)

Gini:

$$Gini(X) = 1 - \sum_{i=1}^{n} p(x_i)^2$$

- Wyraża czystość zbioru
- Szybsza do obliczenia (względem entropii, nie trzeba liczyć logarytmu)
- Ma tendencję do izolowania najczęściej występującej klasy w osobnej gałęzi drzewa.
- Jest zerowa gdy wszystkie instancje w zbiorze są tej samej klasy
- Jest maksymalna gdy instancje są równomiernie rozłożone po klasach
- Wykorzystywana w algorytmie *CART* (Classification and Regression Tree).

Entropia krzyżowa:

$$H(p, q) = -\sum_{i=1}^{n} p(x_i) \log q(x_i)$$

- Stosowana w klasyfikacji
- Wyraża oczekiwaną ilość informacji o instancji, jeżeli zakodujemy ją przy użyciu modelu q zamiast p
- $p(x_i)$ - prawdziwy rozkład prawdopodobieństwa
- $q(x_i)$ - rozkład prawdopodobieństwa przewidywany przez model
- Podczas uczenia modelu q staramy się minimalizować entropię krzyżową, ponieważ to oznacza, że potrzebujemy mniejszej liczby bitów, żeby przewidzieć klasę instancji z rozkładu p (dla rozkładu p podczas uczenia zazwyczaj dokładnie znamy klasy każdej z instancji, więc entropia rozkładu p jest równa 0).

3 Regresja

3.1 Regresja Liniowa

- Opiera się na założeniu, że istnieje liniowa zależność między zmiennymi wejściowymi a zmienną wyjściową.
- Dopasowuje hiperpłaszczyznę (określoną funkcją g), dla której średnia odległość instancji od wartości funkcji g jest najmniejsza.
- Mamy zbiór wektorów $A \subseteq \mathbb{R}^{n+\mathbb{1}}$ i funkcję $f : A \to \mathbb{R}$, która przyporządkowuje każdemu wektorowi $x \in A$ wartość $f(x)$
- Każdy wektor ze zbioru A ma postać $x = [x_0, x_1, ..., x_n]$, gdzie $x_0 = 1$
- Chcemy znaleźć funkcję $g : \mathbb{R}^{n+1} \to \mathbb{R}$ taką, że $g(x) = \Theta^T x$ dla pewnego wektora $\Theta \in \mathbb{R}^n$ i wektor Θ minimalizuje $MSE(x, \Theta) = \frac{1}{n} \sum_{i=1} n(\Theta^T x - f(x))^2$
- Można pokazać, że jeżeli mamy wektor z wszystkich wartości $f(x)$ dla wszystkich wektorów ze zbioru A oraz X jest macierzą złożoną ze wszystkich wektorów z A, to $\Theta = (X^T X)^{-1} X^T$

3.2 Gradient Descent

- Stosowany jeżeli nie można znaleźć rozwiązania analitycznego (np. w przypadku regresji logistycznej), a rozważana funkcja jest ciągła i różniczkowalna w rozważanej dziedzinie
- Zaczynamy ze startowym wektorem x z dziedziny analizowanej funkcji
- Obliczamy gradient funkcji w punkcie x
- Przesuwamy się w kierunku przeciwnym do wektora gradientu, ponieważ gwarantuje to najszybsze możliwe zmniejszanie się wartości funkcji
- Znajduje minimum lokalne.

Stochastic Gradient Descent:

- Stosowany w przypadku, gdy zbiór danych jest bardzo duży
- Do obliczania gradientu wybieramy losowo podzbiór danych
- Znajduje minimum lokalne, szybciej niż *Gradient Descent*, ale nie jest tak dokładny.

3.3 Regresja wielomianowa

- Regresja liniowa, ale zamiast liniowej funkcji g używamy wielomianu g stopnia n
- Do każdej instancji x dodajemy nowe cechy $x_2 = x^2, x_3 = x^3, ..., x_n = x^n$, następnie stosujemy regresję liniową na nowym zbiorze cech.

3.4 Learning Curves

3.4.1 Bias

- Błąd generalizacji wynikający ze złych założeń. Prowadzi do *underfittingu*
- Model jest najprawodopodobniej zbyt prosty.

3.4.2 Variance

- Nadmierna wrażliwość na małą wariancję w zbiorze danych. Prowadzi do *overfittingu*
- Model jest najprawodopodobniej zbyt skomplikowany.

3.4.3 Irreducible Error

- Wynika z zaszumionego zbioru danych.

3.4.4 Kompromis między *Bias* a *Variance*

- Zwiększenie złożoności modelu prowadzi do zwiększenia *Variance* i zmniejszenia *Bias'u* i vice versa.

3.5 Regularyzowane modele liniowe

3.5.1 Ridge Regression

- Regularyzowana wersja *Regresji Liniowej*
- Zmusza model do utrzymywania małych wag
- Używa normy l_2

3.5.2 Lasso Regression

- Regularyzowana wersja *Regresji Liniowej*
- Używa normy l_1
- Ma tendencje do usuwania wag dla najmniej ważnych cech
- Zwraca *Rzadki model* (Dużo zer w polach wag)

3.5.3 Early Stopping

- Zatrzymuje proces uczenia w momencie gdy *błąd walidacji* osiąga minimum.

3.6 Regresja Logistyczna

- Wykorzystuj funkcję aktywacji $f(x) = \frac{1}{1+e^{-x}}$ (Sigmoid)
- Szacuje prawdopodobieństwo przynależności instancji do pewnej klasy.
- Stosuje funkcji *sigmoid* do zwrócenia prawdopodobieństwa (Sigmoid zwraca wartości między 0 a 1).

4 SVM (Support Vector Machines)

- Algorytm klasyfikacji oparty o zasadę największego marginesu.

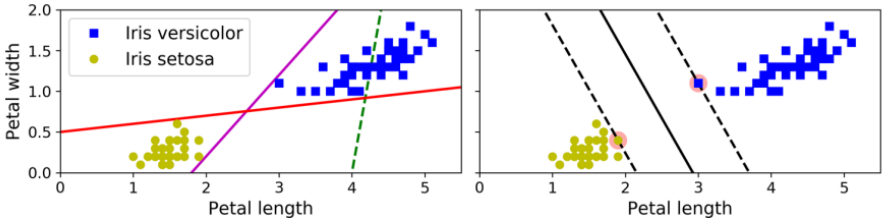

Figure 3: Porównanie regresji liniowej (lewy wykres) z SVM (prawy wykres)

- Wrażliwy na skalowanie danych (Zawsze skalować przed użyciem)
-

4.1 Hard Margin Classification

- Wszystkie instancje muszą się znaleźć poza marginesem.
- Działa tylko wtedy, gdy dane da się liniowo rozdzielić.
- Wrażliwy na *outliers'y*

4.2 Soft Margin Classification

- Elastyczny model
- Szyka balansu między posiadaniem jak największego marginesu, a limitowaniem liczby jego naruszeń.

4.3 Nieliniowa klasyfikacja SVM

- Używaj kiedy dane nie da się rozdzielić liniowo.

4.3.1 Polynomial Kernel

- Sztuczka dzięki której możemy dostać wyniki, jakbyśmy korzystali z wielomianowego modelu bez użycia go.

4.4 Regresor SVM

- By działał musimy odwrócić jego zadanie- zmieścić jak najwięcej instancji w jak najmniejszym marginesie.
- Model jest ϵ niewrażliwy, czyli dodawanie więcej instancji znajdujących się w marginesie nie wpływa na zdolność przewidywania modelu.
- Do rozwiązywania nieliniowych modeli użyj **kernelized SVM model**

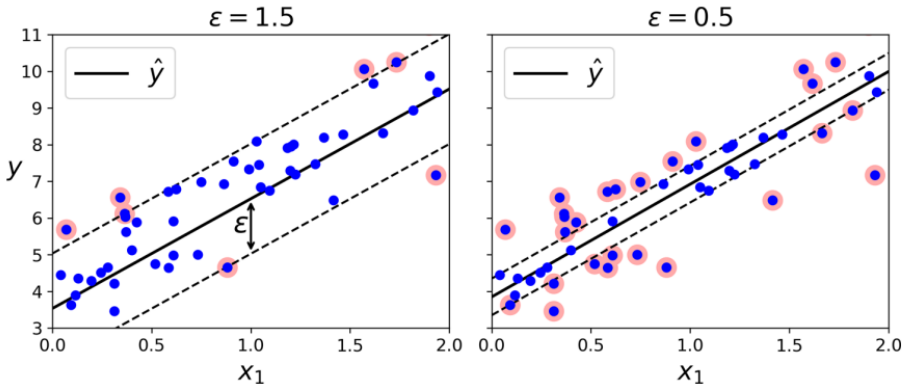

Figure 4: Wpływ ϵ na wydajność regresji

4.5 Drzewa Decyzyjne

- Stosowany do klasyfikacji i regresji
- Nie wymaga przygotowania danych, nie trzeba skalować ani centrować
- **Model Nieparametryczny**
  - Liczba parametrów nie jest zdefiniowana przed ćwiczeniem modelu
  - Model może się przeuczyć.
- **White Box model**
  - Prosty do zinterpretowania- wiemy dlaczego podjął taką a nie inną decyzję.
- Scikit używa algorytmu **CART** (próbuje zachłannie minimalizować współczynnik Gini) do trenowania drzew decyzyjnych
- Algorytm **CART** w celu ustalenia miejsca podziału oblicza wartość $J(k, t_k) = \frac{m_{lewa}}{m} * G_{lewa} + \frac{m_{prawa}}{m} * G_{prawa}$, gdzie G_{lewa} i G_{prawa} wyrażają nieczystości lewej i prawej części po podziale, a m_{lewa} i m_{prawa} to liczba instancji w lewej i prawej części, m to liczba wszystkich instancji
- Obrót przestrzeni instancji może całkowicie zmieniać wygenerowane drzewo i jego złożoność.

4.5.1 White Box vs Black Box

- W przypadku *Black Box* ciężko jest sprawdzić dlaczego dany model podjął taką decyzję
- Dla modeli, które nie są *White Box* bardzo trudnym zadaniem jest dokładne określenie wnioskowania przeprowadzonego przez model, które może być łatwo zrozumiane przez człowieka0
- Przykłady *White Box*:
  - Drzewa decyzyjne
  - Regresja Liniowa
  - SVM
- Przykłady *Black Box*:
  - Sieci neuronowe
  - Random Forests

4.5.2 Hiperparametry

- Bez żadnych ograniczeń model bardzo szybko przeucza się (Wtedy go nazywamy nieparametrycznym, opisany wyżej)
- ***Regularyzacja*** jest procesem mającym przeciwdziałać przeuczeniu, przez dobranie odpowiednich hiperparametrów
  - Najważniejszą wartość jaką możemy dostrajać jest ograniczenie maksymalnej *głębokości drzewa* (Domyślnie jest ∞)

4.5.3 Regresja

- Struktura drzewa przypomina tą z problemu klasyfikacji.
- Możemy uznać problem regresji jako problem klasyfikacji z nieograniczoną liczbą klas, którą możemy regulować przez maksymalną głębokość drzewa.

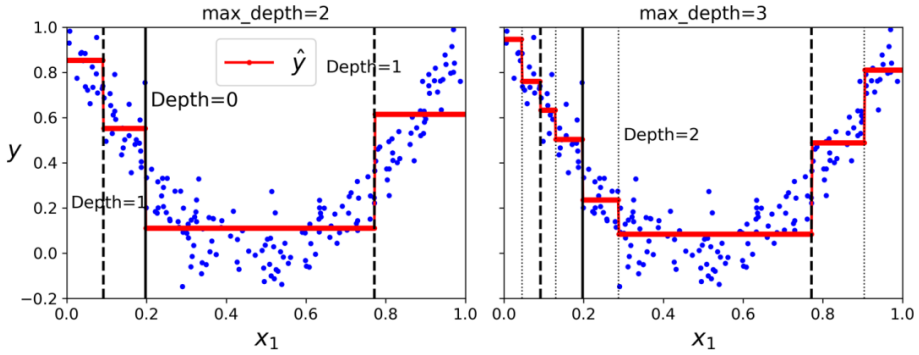

Figure 5: Predykcje 2 regresorów o różnych maksymalnych głębokościach

5 Ensemble Learning i Random Forests

- Stosujemy zasadę *mądrości tłumu* - jeżeli mamy wiele klasyfikatorów, to możemy je zagregować w grupę klasyfikatorów znacznie zwiększając wydajność modelu.
- Wszystkie klasyfikatory powinny być od siebie niezależne
- Redukuje *Bias* i *Variance*

5.1 W problemie klasyfikacji rozróżniamy 2 rodzaje klasyfikatorów:

Wykorzystywana jest moc przyjaźni (ang. *Power of friendship*).

5.1.1 *Hard Voting Classifier*

- Wybiera klasę, która jest dominantą zbioru propozycji klas zwróconych przez klasyfikatory.

5.1.2 *Soft Voting Classifier*

- Wykorzystuje prawdopodobieństwa zwracane przez model, następnie uśrednia je i wybiera klasę z najwyższym średnim prawdopodobieństwem.

5.2 Bagging i Pasting

- Wykorzystują wiele instancji klasyfikatora tego samego typu, ale trenowanych na różnych podzbiorach danych.
- **Bagging** (Bootstrap Aggregating) polega na losowaniu instancji ze zwracaniem (zastępowaniem) i trenowaniu na nich różnych klasyfikatorów, a następnie wykorzystaniu metody *hard voting* do wyboru klasy.
- **Pasting** jest podobny do *Bagging'u*, ale zamiast losować instancje ze zwracaniem, losuje je bez zwracania, co oznacza, że każdy klasyfikator może być trenowany tylko na części danych, a liczba klasyfikatorów jest ograniczona przez liczbę instancji w zbiorze treningowym.

5.3 Random Forests

- Zbiór drzew decyzyjnych
- Dodaje extra losowość
- Umożliwia łatwe sprawdzenie istotności pewnej cechy
- Jeżeli zastosujemy *Bagging* na drzewach decyzyjnych, to otrzymamy *Random Forest*
- Agreguje predykcje ze wszystkich drzew i wybiera klasę o największej ilości głosów (hardvoting)
  - Grupa drzew decyzyjnych
  - Każdy uczy się na innym podzbiorze zbioru danych

5.3.1 Extremely Randomized Trees Ensembly

- Szybciej się uczy
- Stosuje losowe progi dla każdej cechy

5.4 Boosting

- Łączy wiele *weak learners* w *strong learner*
- Trenuje predyktory sekwencyjnie
  - Każdy kolejny próbuje poprawić błędy poprzedniego

5.4.1 AdaBoost

- Adaptive Boosting
- Zwraca uwagę na instancje słabo dopasowane przez poprzednie predyktory.
- Nie skaluje się dobrze.

5.4.2 Gradient Boosting

- Możemy go użyć z różnymi funkcjami straty
- Dopasowuje nowy predyktor do pozostałego błędu przez poprzedni model
- **XGBoost**
  - Aktualnie najlepszy klasyfikator (razem z CatBoostem).

5.5 Stacking

- Metoda podobna do *Voting Classifier'a*, ale zamiast używać prostych funkcji do agregacji predykcji, trenuje model, aby nauczył się jak łączyć predykcje innych modeli
- Możliwe jest stosowanie bardziej zagnieżdżonych architektur, w których występują kolejne warstwy modeli.

6 Redukcja Wymiarów

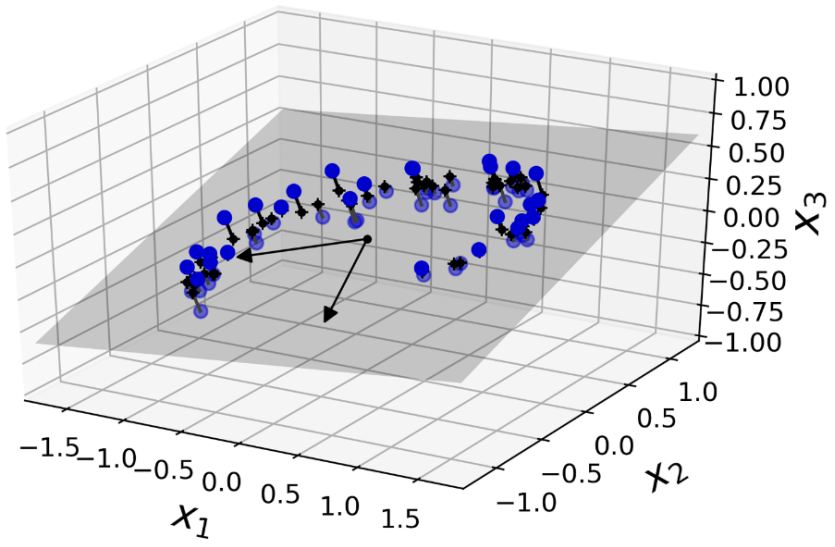

Figure 6: Rzutowanie danych na inną przestrzeń

- Stosujemy do uproszczenia zbioru danych w celu przyspieszenia procesu uczenia modelu
- Prowadzi do utraty części informacji, umożliwiając jednocześnie lepszą wydajność modelu
- Może być również wykorzystywana do wizualizacji danych.

6.1 Curse of Dimensionality

- Odnosi się do zjawiska, w którym dodanie kolejnych wymiarów do zbioru danych powoduje znaczny (eksponencjalny) wzrost wymaganej ilości danych do zachowania odpowiedniej gęstości danych.

6.2 PCA - Principal Component Analysis

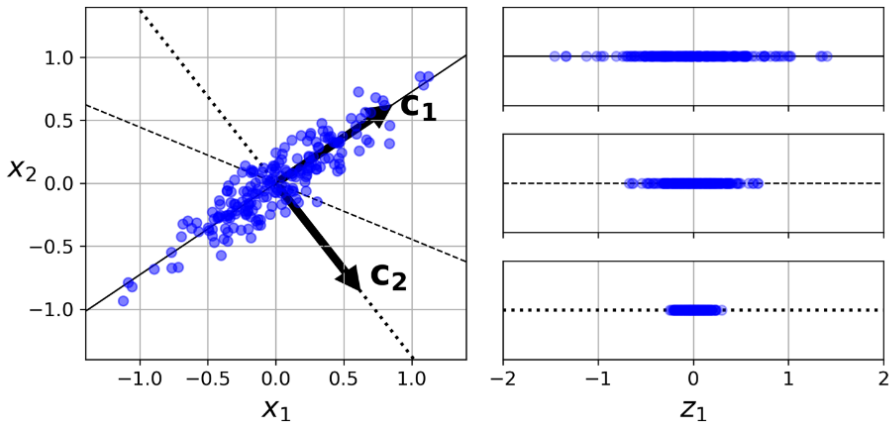

Figure 7: PCA - Principal Component Analysis

- Jest to metoda redukcji wymiarów, w której wybieramy kierunki, które zachowują najwięcej informacji
- Kierunki te są nazywane **principal components**
- PCA znajduje kierunki, które minimalizują *średnią kwadratową odległość* między punktami danych a ich rzutami na kierunki
- Staramy się znaleźć takie kierunki, dla których występuje największa wariancja danych
- Na początku standaryzujemy dane, aby średnie wartości były równe 0
- Znajdujemy bazę przestrzeni, która jest najbardziej zbliżona do danych pod względem *średniej kwadratowej odległości* dla punktów danych i ich rzutów na bazę
- Istnieje szybszy algorytm randomizowany, który znajduje przybliżone rozwiązanie.

6.2.1 SVD - Singular Value Decomposition

- Jest to metoda rozkładu macierzy na iloczyn 3 macierzy
- Umożliwia wyznaczenie kierunków, które zachowują najwięcej informacji
- Stosowana w PCA
- Uogólnienie wartości własnych i wektorów własnych na macierze niekwadratowe
- Największe wartości singularne odpowiadają kierunkom, które zachowują najwięcej informacji.

6.3 Incremental PCA

- minibatch, out-of-core, praca na strumieniach, trzeba podać liczbę wymiarów
- Czyli w sumie po prostu PCA na online(minibatches), gdzie nie ładujemy całego zestawu danych na raz do modelu

6.4 Rozmaitości

- Są to zbiory danych, które mogą być zredukowane do mniejszej liczby wymiarów, ale nie muszą być przestrzeniami liniowymi
- W małej skali wyglądają jak przestrzenie liniowe, ale w większej skali mogą mieć kształty przeróżne
- Zastosowanie dla nich algorytmu PCA może prowadzić do zbyt intensywnej utraty informacji
- Istnieją algorytmy, które pozwalają na redukcję wymiarów dla takich zbiorów danych.

6.4.1 LLE - Locally Linear Embedding

- Algorytm ten znajduje lokalne zależności między punktami danych, a następnie próbuje zachować te zależności w niższej wymiarowości
- Jest to algorytm nienadzorowany
- Może prowadzić do zniekształcenia danych w dużej skali
- W pierwszym kroku znajduje najbliższych sąsiadów dla każdego punktu danych
- Następnie znajduje wagi, które pozwalają na rekonstrukcję każdego punktu danych jako kombinacji liniowej jego najbliższych sąsiadów
- W ostatnim kroku rzutuje dane na przestrzeń o niższej wymiarowości, zachowując lokalne zależności.

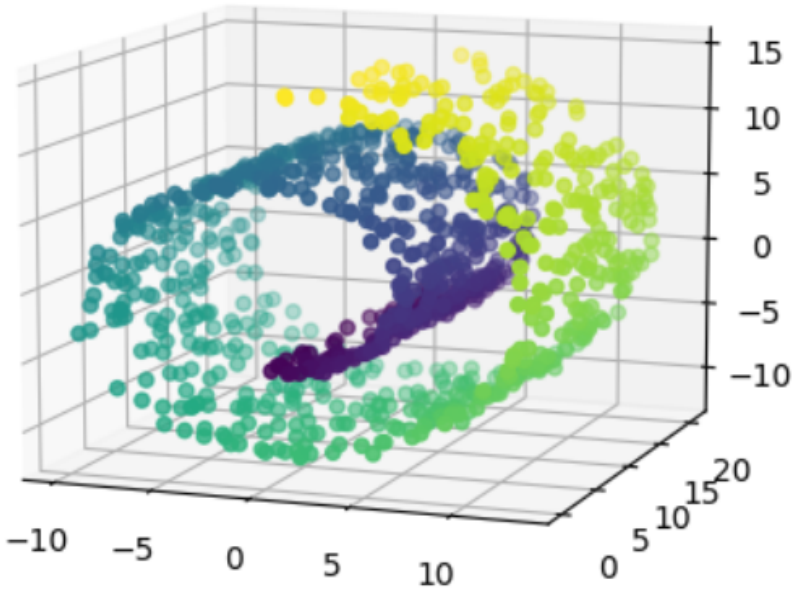

Figure 8: Rozmaitość - przykład Swiss Roll

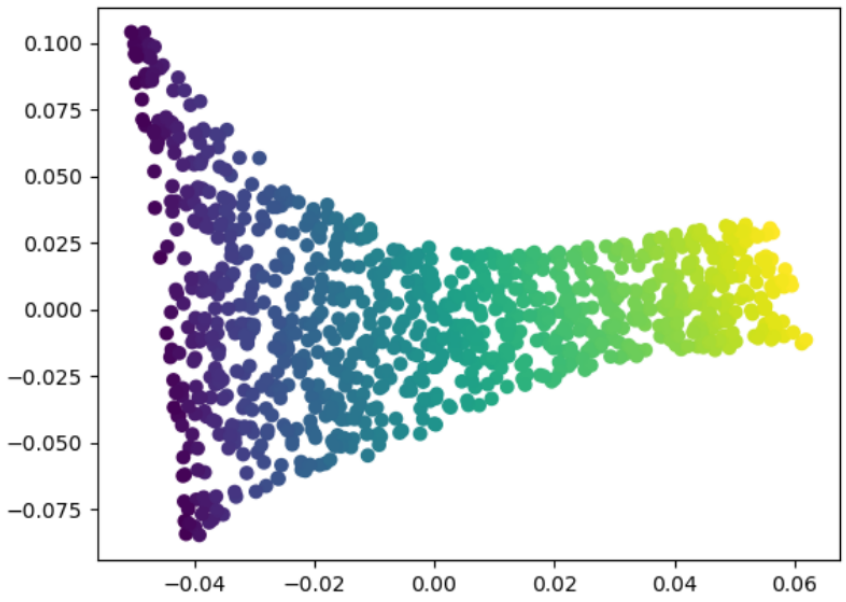

Figure 9: Swiss Roll po zastosowaniu LLE

7 Uczenie nienadzorowane

Kategorie uczenia nienadzorowanego:

- Klasteryzacja *clustering*
  - identyfikacja klas
  - redukcja wymiarów
  - analiza danych (po klasteryzacji, dla każdej klasy osobno)
  - uczenie częściowo nadzorowane
  - segemntacja obrazu, detekcja, kompresja
- Detekcja naomalii
  - detekcja wartości odstających, *outlierów*
- Estymacja gęstości *density estimation*

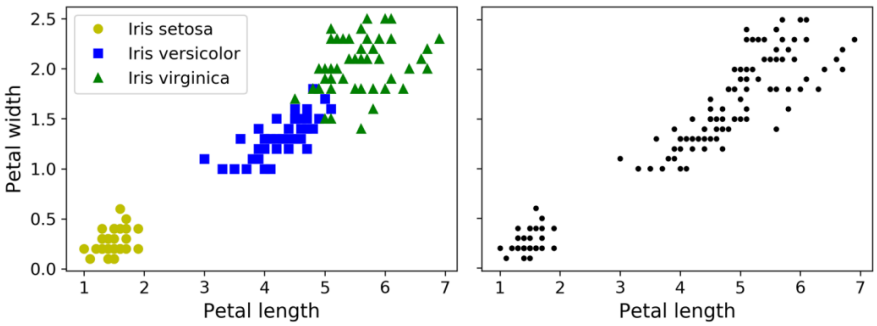

Figure 10: Podobne do klasyfikacji, ale nie wiadomo ile jest klas

7.1 Algorytm centroidów (k-średnich) *K-Means*

- Algorytm centroidów (k-średnich) *K-Means* jest jednym z najpopularniejszych algorytmów klasteryzacji.
- Algorytm stara się znaleźć środek każdego z k skupisk
- Algorytm ten przypisuje każdy punkt danych do najbliższego centroidu, a następnie przesuwa centroidy tak, aby minimalizować średnią kwadratową odległość między punktami danych a ich centroidami
- k jest parametrem algorytmu, który musi zostać określony przez użytkownika
- Jest zbieżny
- Nie gwarantuje znalezienia optimum (zależy od kroku 1)
  - Domyślnie algorytm uruchamiany jest 10 razy
  - Wybierany jest model z najmniejszą **inercją**: średnio-kwadratowa odległość między instancjami i ich centroidami
    - zmierz odległość między instancjami a ich centroidami
    - zsumuj kwadraty w/w odległości w ramach klastra
    - zsumuj wartości inercji dla wszystkich klastrów
- Przedstawieniem wyniku działania algorytmu jest Diagram Woronoja *Voronoi*

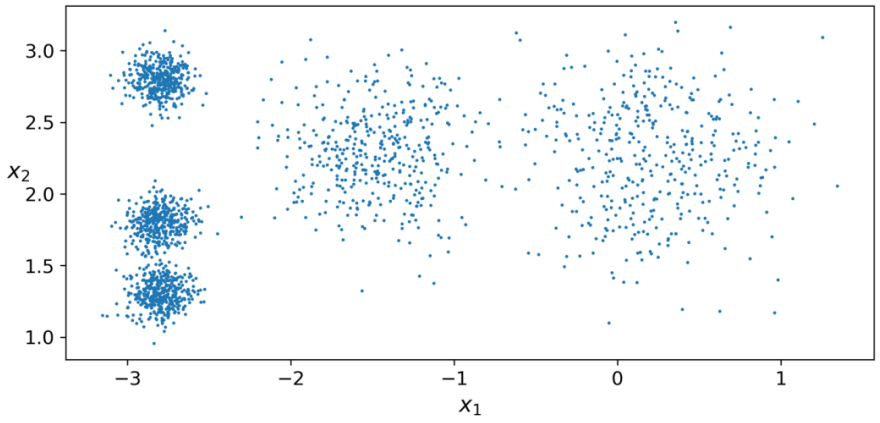

Figure 11: Przykładowy rozkład danych

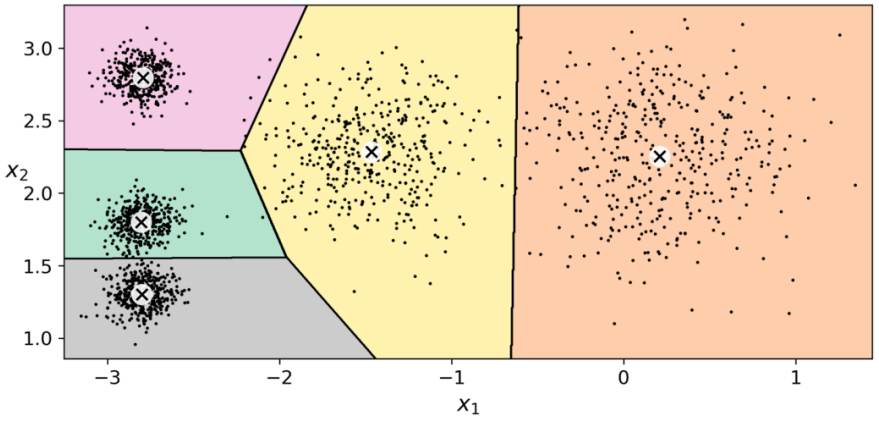

Figure 12: Diagram Woronoja *Voronoi* wyznaczony przez alg. K-Means

7.1.1 Wyznaczanie liczby klastrów

Do wyznaczenia liczby klastrów nie wystarcza sama inercja, ponieważ maleje ona wraz ze zwiększaniem się liczby klastrów.

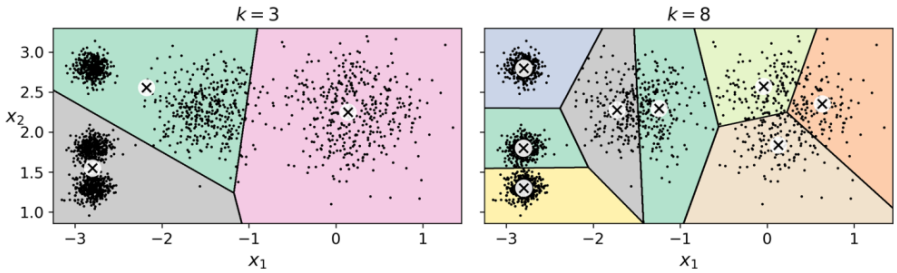

Figure 13: Przykład podziału na niepoprawną liczbę klastrów

Inercja nie wystarcza, ale można ją wykorzystać. Wystarczy wyznaczyć inercję dla różnych wartości k i wybrać tę, która jest na 'zgięciu' wykresu.

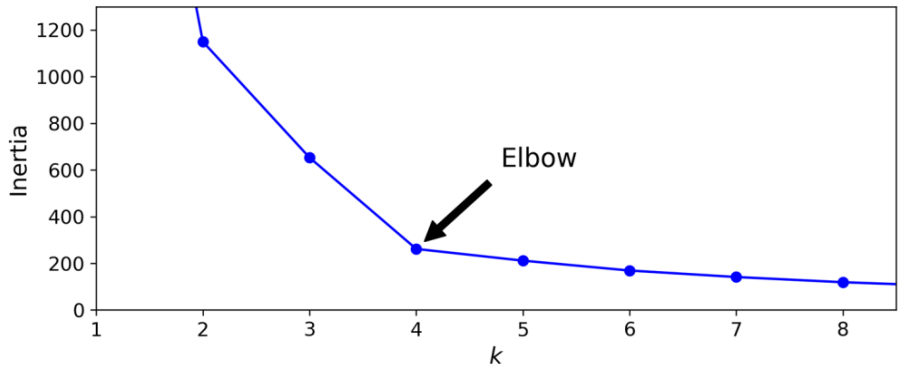

Figure 14: Wykorzystanie inercji do wyznaczenia liczby k

Do wyznaczenia liczby klastrów możemy również wykorzystać **Wskaźnik sylwetkowy, *silhouette score***. Wskaźnik bierze pod uwagę średnią odległość pomiędzy obserwacjami wewnątrz grupy (a) i średnią odległość pomiędzy obserwacjami do najbliższej "obcej" grupy (b) i dany jest wzorem:

$$s = \frac{a - b}{max(a, b)}$$

7.2 DBSCAN

- Algorytm DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) jest algorytmem klasteryzacji, który znajduje skupiska o wysokiej gęstości
- Algorytm ten znajduje skupiska o wysokiej gęstości, a także punkty odstające
- Algorytm ten nie wymaga określenia liczby klastrów
- Wymaga określenia dwóch parametrów: *eps* i *min_samples*

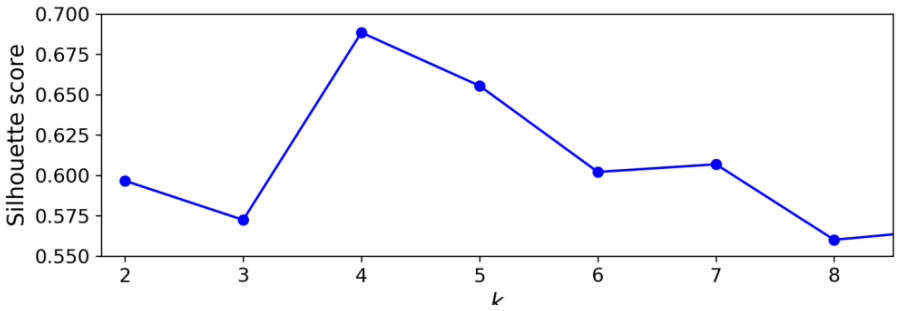

Figure 15: Wykorzystanie wskaźnika sylwetkowego do wyznaczenia liczby k

  - *eps* - maksymalna odległość między dwoma punktami, aby zostały one uznane za sąsiadów
  - *min_samples* - minimalna liczba punktów, aby uznać je za rdzeń (wliczając w to punkt, dla którego szukamy sąsiadów)
- Wszystkie instancje, które nie są rdzeniami, ale mają sąsiadów, są uznawane za brzegi, wchodzą w skład tego samego klastra, co ich rdzeń
- Instancje, które nie są ani rdzeniami, ani brzegami, są uznawane za anomalią (nie należą do żadnego klastra)

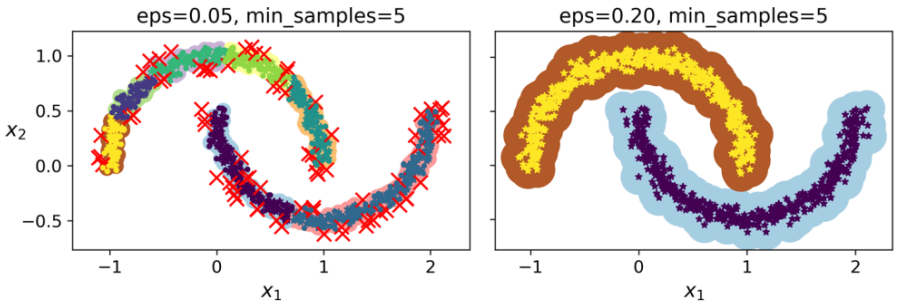

Figure 16: Alt text

8 Sieci neuronowe - wprowadzenie

8.1 Perceptron

- Składają się z jednej warstwy neuronów
- Każdy neuron jest jednostką liniową, po której następuje funkcja aktywacji
- Sposób działania:
  - oblicz sumę wejść $z = w_1x_1 + w_2x_2 + ... + w_nx_n = x^T w$
  - zastosuj funkcję schodkową: $h_w(x) = step(z)$
- Ograniczenia:
  - Nie potrafią rozwiązać pewnych trywialnych problemów, np. XOR. W takich przypadkach stosuje się **sieci wielowarstwowe (MLP)**

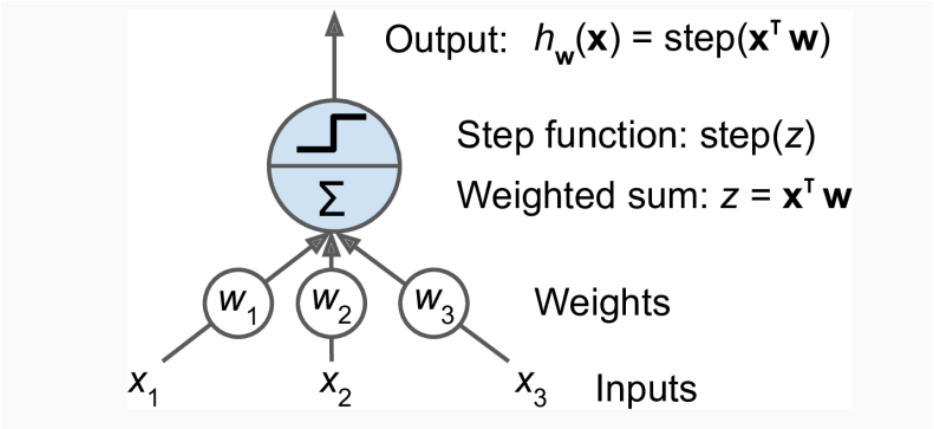

Figure 17: Alt text

8.1.1 Uczenie perceptronu

- Uczenie perceptronu polega na znalezieniu wektora wag w, który pozwoli na poprawne sklasyfikowanie jak największej liczby instancji
- Wagi są aktualizowane na podstawie błędu predykcji według wzoru $w_{i,j}^{(nastpnaiteracja)} = w_{i,j} + \eta(y_j - \hat{y_j})x_i$
  - $w_{i,j}$ - waga połączenia między neuronem i a neuronem j
  - η - współczynnik uczenia
  - y_j - wartość oczekiwana
  - $\hat{y_j}$ - wartość przewidziana
  - x_i - wartość wejścia

8.2 Funkcje aktywacji

- Istotne jest, aby funkcja aktywacji była nieliniowa, ponieważ w przeciwnym razie sieć neuronowa będzie zachowywać się jak jedna warstwa neuronów (dla macierzy W_1 i W_2 będzie można znaleźć macierz W, która będzie równoważna działaniu sieci neuronowej, $W = W_2W_1$)

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

$$tanh(z) = 2\sigma(2z) - 1$$

$$ReLU(z) = max(0, z)$$

$$LeakyReLU(z) = max(\alpha z, z)$$

$$ELU(z) = \begin{cases} \alpha(e^z - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

$$SeLU(z) = \begin{cases} \lambda\alpha(e^z - 1) & \text{if } z < 0 \\ \lambda z & \text{if } z \geq 0 \end{cases}$$

$$Softmax(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Softmax - funkcja aktywacji wykorzystywana w warstwie wyjściowej klasyfikatorów wieloklasowych, generuje rozkład prawdopodobieństwa.

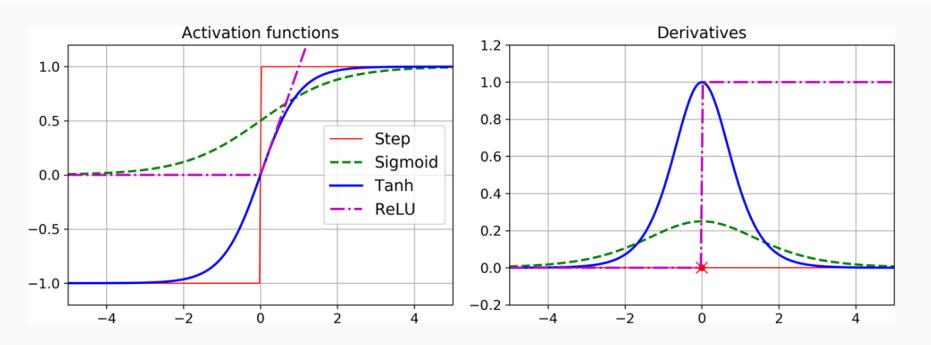

Figure 18: Niektóre funkcje aktywacji wykorzystywane w sieciach neuronowych

8.3 Warstwy

8.3.1 Warstwa gęsta

- Każdy neuron jest połączony z każdym neuronem z poprzedniej warstwy
- Wagi połączeń są zapisane w macierzy wag $W*$

- Każdy neuron ma dodatkowy parametr b, który jest nazywany *biasem* - w innym przypadku dla wektora zerowego na wejściu, na wyjściu otrzymalibyśmy wektor zerowy (jeżeli funkcja aktywacji ma punkt stały w 0)

9 Głębokie sieci neuronowe

9.1 Budowa modelu

9.1.1 Keras Sequential API

- Najprostszy sposób tworzenia sieci neuronowej
- Zakłada, że sieć jest sekwencją warstw
- Warstwy dodajemy jako instancje odpowiednich klas z pakietu keras.layers
- parametr można przekazywać jako ciągi znaków. Jest to zapis uproszczony: zamiast `"relu"` można przekazać `keras.activations.relu`
- Normalizację danych można wykonać za pomocą warstwy `keras.layers.Normalization`, lub `keras.layers.Flatten`, albo zrobić samemu wcześniej

```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    tf.keras.layers.Dense(300, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])
```

9.1.2 Keras Functional API

- Pozwala na tworzenie bardziej skomplikowanych architektur sieci neuronowych
- Pozwala na tworzenie grafów obliczeniowych, w których nie wszystkie warstwy są połączone ze sobą w sekwencji
- Pozwala na tworzenie wielu modeli, które mają współdzielone warstwy
- Do tworzenia modelu wykorzystujemy klasę `tf.keras.Model`, podaje się w niej warstwy wejściowe i wyjściowe
- Do tworzenia warstw wykorzystujemy klasę `tf.keras.layers`, podobnnie jak w przypadku `Sequential` API
- Łączenie warstw odbywa się za pomocą operatora `(warstwa)(wejście)`, podobnie jak w przypadku wywoływania funkcji, co oznacza, że warstwa jest wywoływana na wejściu otrzymanym z poprzedniej warstwy będącej argumentem wywołania

```
import tensorflow as tf

input_ = tf.keras.layers.Input(shape=[28, 28])
flatten = tf.keras.layers.Flatten(input_shape=[28, 28])(input_)
hidden1 = tf.keras.layers.Dense(300, activation="relu")(flatten)
hidden2 = tf.keras.layers.Dense(100, activation="relu")(hidden1)
concat = tf.keras.layers.Concatenate()([input_, hidden2])
```

```
output = tf.keras.layers.Dense(10, activation="softmax")(concat)
model = tf.keras.Model(inputs=[input_], outputs=[output])
```

9.2 Kompilacja i uczenie modelu

Po utworzeniu modelu należy go skompilować za pomocą metody `compile()`. Metoda ta przyjmuje następujące parametry:

- `optimizer`: Określa **optymalizator** używany do aktualizacji wag modelu podczas procesu uczenia.Optymalizator reguluje sposób, w jaki model aktualizuje wagi na podstawie straty i algorytmu optymalizacji. Ich argumentem jest m.in. `learning_rate`. Przykładowe optymalizatory:
  - ***SGD*** - Stochastic Gradient Descent
  - ***Momentum*** - SGD z pędem
  - ***Nesterov Accelerated Gradient*** - SGD z pędem Nesterova
  - ***AdaGrad*** - Adaptive Gradient, nie wykorzystuje pędu, ale dostosowuje współczynnik uczenia dla każdego parametru na podstawie jego historii aktualizacji
  - ***Adam*** - Adaptive Moment Estimation, wykorzystuje pęd i historię aktualizacji
- `loss`: Określa **funkcję straty**, która jest używana do oceny odchylenia między przewidywaniami modelu a rzeczywistymi wartościami. Przykładowe funkcje straty to 'mean_squared_error', 'categorical_crossentropy', 'binary_crossentropy' itp. Wybór odpowiedniej funkcji straty zależy od rodzaju problemu i rodzaju wyjścia modelu.
- `metrics`: Określa **metryki**, które będą używane do oceny wydajności modelu. Przykładowe metryki to 'accuracy', 'precision', 'recall', 'mean_absolute_error' itp. Metryki służą do monitorowania wydajności modelu podczas uczenia i ewaluacji.
- Inne opcjonalne argumenty, takie jak loss_weights, sample_weight_mode, weighted_metrics, które pozwalają na bardziej zaawansowane konfigurowanie procesu kompilacji modelu.

```
model.compile(loss="adam",
              optimizer="sgd",
              metrics=["accuracy"])
```

A następnie wytrenować model za pomocą metody `fit()`. Metoda ta przyjmuje następujące parametry:

- `x`: **Dane wejściowe** do modelu.
- `y`: **Dane wyjściowe** (etykiety) odpowiadające danym wejściowym x.
- `batch_size`: Określa liczbę próbek, które są przetwarzane jednocześnie przez model w trakcie jednej iteracji.
- `epochs`: Określa liczbę **epok uczenia** - pełnych przebiegów przez zbiór treningowy. Każda epoka oznacza jedno przejście przez cały zbiór treningowy.
- `validation_data`: **Dane walidacyjne** używane do oceny wydajności modelu na każdej epoce. Może to być krotka (x_val, y_val) zawierająca dane wejściowe i oczekiwane wyjście dla danych

walidacyjnych.

- `callbacks`: Lista obiektów zwrotnych (callbacks), które są wywoływane podczas treningu w różnych momentach. Przykłady to ModelCheckpoint, EarlyStopping, TensorBoard itp. Callbacks pozwalają na dostosowywanie zachowania treningu w zależności od określonych warunków.
- `verbose`: Określa tryb wyświetlania informacji podczas treningu. Może przyjąć wartość 0 (bez wyświetlania), 1 (wyświetlanie paska postępu) lub 2 (wyświetlanie jednej linii na epokę).
- Inne opcjonalne argumenty, takie jak validation_split, shuffle, class_weight itp., które pozwalają na bardziej zaawansowane konfigurowanie procesu treningu modelu.

```
history = model.fit(
    X_train,
    y_train,
    batch_size=32,
    epochs=10,
    validation_data=(X_valid, y_valid),
    callbacks=[early_stopping_cb],
    verbose=1
)
```

9.3 Callbacks

Callbacki pozwalają na wykonywanie dodatkowych operacji w trakcie uczenia modelu. Przykładowe zastosowania:

- `ModelCheckpoint` - Zapisywanie punktów kontrolnych
- `EarlyStopping` - zatrzymanie uczenia, jeżeli nie nastąpi poprawa wyniku przez 10 epok (bardzo częste zastosowanie)
- `TensorBoard` - zapisywanie logów do wykorzystania w TensorBoard

```
checkpoint_cb = keras.callbacks.ModelCheckpoint(
    "my_keras_model.h5",
    save_best_only=True
)


early_stopping_cb = keras.callbacks.EarlyStopping(
    patience=10,
    restore_best_weights=True
)


tensorboard_cb = keras.callbacks.TensorBoard(
    log_dir="./my_logs",
    histogram_freq=1,
```

```
    profile_batch=100
)
```

Callbacki dodajemy w parametrze `callbacks` metody `fit`

9.4 Analiza procesu uczenia

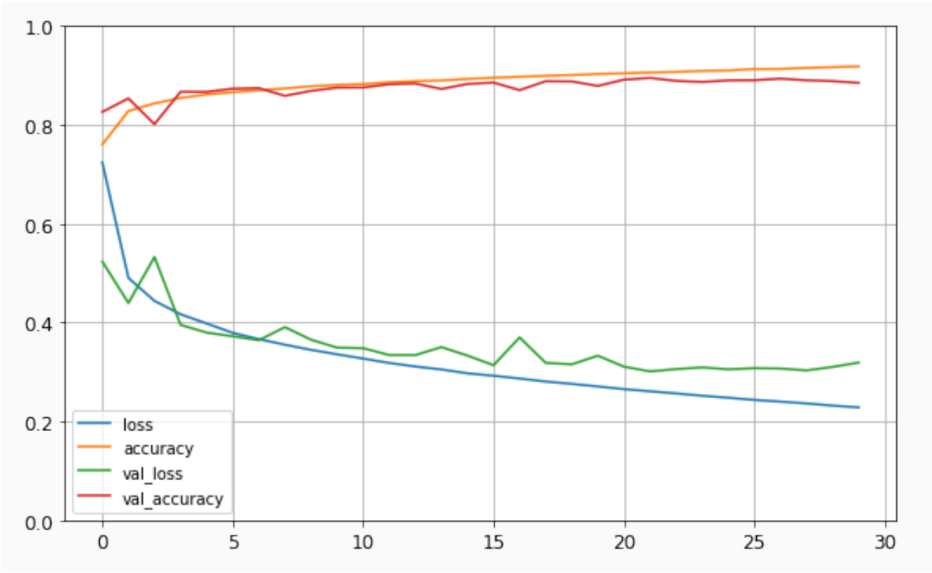

Figure 19: Wykres wartości miar w trakcie procesu uczenia

- ***Loss*** - miara, która określa, jak bardzo wyniki modelu różnią się od oczekiwanych wartości.
- ***Accuracy*** - miara, która określa, jak dokładnie model przewiduje klasy lub etykiety dla danych.
- ***Recall*** - miara, która określa, jak wiele pozytywnych przypadków zostało wykrytych przez model.
- ***Precision*** - miara, która określa, jak wiele pozytywnych przypadków zostało poprawnie określonych przez model.
- ***Val_loss*** - strata obliczana na danych walidacyjnych, służy do monitorowania uczenia modelu i unikania przeuczenia.
- ***Val_accuracy*** - dokładność obliczana na danych walidacyjnych, pomaga ocenić, jak dobrze model generalizuje na nowych danych.

Przykłady funkcji strat zostały przedstawione na początku dokumentu.

9.5 Przeszukiwanie przestrzeni hiperparametrów

9.5.1 SciKit-Learn

- `RandomizedSearchCV` - losowe przeszukiwanie przestrzeni hiperparametrów
- `GridSearchCV` - przeszukiwanie przestrzeni hiperparametrów siatką wartości parametrów

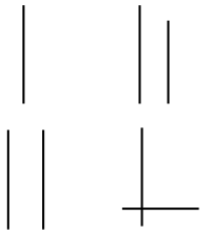

Figure 20: Loss

9.5.2 Keras Tuner

- `RandomSearch` - losowe przeszukiwanie przestrzeni hiperparametrów

10 Konwolucyjne sieci neuronowe (CNN - Convolutional Neural Networks)

- CNN są stosowane do przetwarzania wielowymiarowych danych, takich jak obrazy, wideo itp.
- Wykorzystują specjalny rodzaj warstwy zwanej warstwą konwolucyjną (Convolutional Layer), która wykonuje operację konwolucji na danych wejściowych.

10.1 Konwolucja

- Konwolucja to operacja matematyczna, która łączy dwa zestawy danych za pomocą funkcji matematycznej, aby wygenerować trzeci zestaw danych.
- W przypadku konwolucyjnych sieci neuronowych operacją konwolucji jest iloczyn skalarny (mnożenie element-wise) dwóch zestawów danych.
- Konwolucja jest operacją liniową, która może być używana do wielu celów, takich jak wykrywanie krawędzi i innych wzorców w obrazach, wykrywanie cech w danych itp.
- Polega na wykonywania sum ważonych dla fragmentów funkcji wejściowej ważonej przez jądro (kernel, który jest macierzą wag).
- W przypadku sieci neuronowych dane wejściowe są zwykle macierzą wielowymiarową (np. obrazem) i są one łączone z macierzą wag (kernel), aby wygenerować macierz wyjściową
- Wagi są parametrami, które są uczone podczas treningu modelu
- W przypadku obrazów macierz wejściowa zawiera piksele obrazu, a macierz wag zawiera filtry, które są aplikowane na obrazie
- Konwolucja może być obliczana na całym obrazie, ale zwykle stosuje się ją tylko do fragmentu obrazu, aby uzyskać macierz wyjściową o takich samych wymiarach jak macierz wejściowa
- W przypadku obrazów wagi są zwykle małymi macierzami o wymiarach 3x3 lub 5x5. W przypadku obrazów kolorowych, które mają 3 kanały kolorów (RGB), macierz wag ma wymiary 3x3x3 lub 5x5x3.
- Każda warstwa konwolucyjna składa się z wielu filtrów, które są stosowane do danych wejściowych, aby wygenerować różne macierze wyjściowe w celu wykrycia różnych cech w danych wejściowych

10.2 Pooling

- Pooling jest operacją, która zmniejsza wymiary danych wejściowych poprzez zastąpienie fragmentu danych wejściowych pojedynczą wartością reprezentującą ten fragment zwracaną przez sprecyzowaną wcześniej funkcję
- Najczęściej stosowaną funkcją agregującą jest funkcja max, która zwraca maksymalną wartość w fragmencie danych wejściowych
- Pozwala kolejnym warstwom sieci na wykrywanie cech bardziej ogólnych, poprzez zwielokrotnienie obszaru, na którym bezpośrednio działają
- Często stosowany po warstwie konwolucyjnej, aby zmniejszyć wymiary danych wejściowych
- Najczęściej zmniejsza każdy wymiar danych wejściowych o połowę.

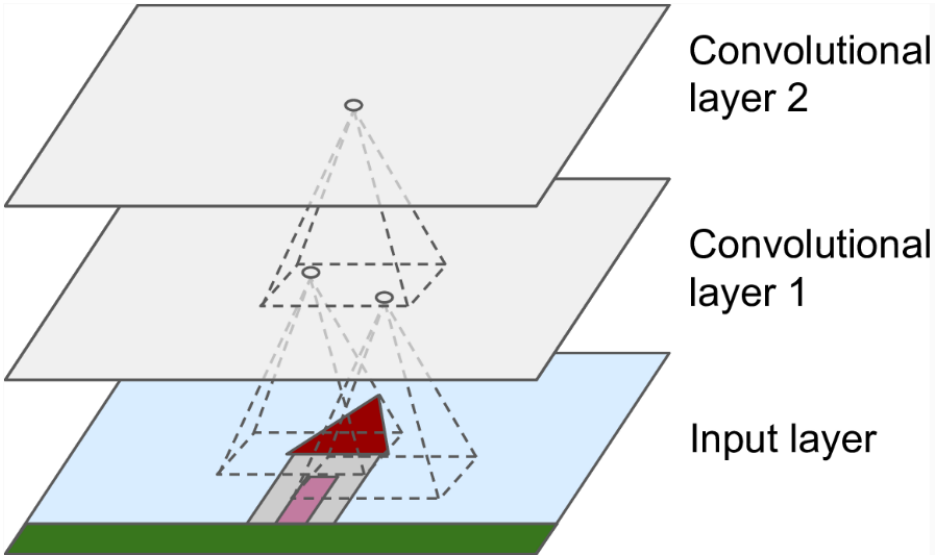

Figure 21: Przykład prostej sieci konwolucyjnej

11 Rekurencyjne sieci neuronowe (RNN - Recurrent Neural Networks)

- RNN są stosowane do przetwarzania sekwencyjnych danych, takich jak tekst, dźwięk, czasowe serie danych itp.
- Często wykorzystywane do predyckji na podstawie sekwencji danych wejściowych (o dowolnej długości), najczęściej do przewidywania przyszłości.
- Wykorzystują specjalny rodzaj warstwy zwanej warstwą rekurencyjną (Recurrent Layer), która przechowuje stan wewnętrzny, który jest aktualizowany za każdym razem, gdy warstwa otrzymuje dane wejściowe.
- Zastosowania: finanse (giełda), pojazdy autonomiczne, sterowanie, wykrywanie usterek

Podstawowym elementem RNN jest komórka rekurencyjna, która ma stan wewnętrzny przechowujący informacje z poprzednich **kroków czasowych (ramek)**. W każdym kroku czasowym komórka otrzymuje dane wejściowe oraz stan wewnętrzny (z poprzedniego kroku) i generuje nowy stan wewnętrzny oraz dane wyjściowe. Ten proces jest powtarzany dla każdego kroku czasowego.

Istnieje kilka różnych typów RNN, takich jak **SimpleRNN**, **LSTM** (Long Short-Term Memory) i **GRU** (Gated Recurrent Unit), które różnią się w sposobie zarządzania i aktualizacji stanu wewnętrznego. Na przykład, LSTM wprowadza bramki, które kontrolują przepływ informacji, pozwalając na efektywne uczenie się zależności na różnych skalach czasowych i unikanie problemu zanikającego gradientu.

Działanie RNN można podsumować w kilku krokach:

- Dane wejściowe sekwencyjne są podzielone na kroki czasowe.
- Na każdym kroku czasowym, dane wejściowe są przetwarzane przez komórkę rekurencyjną, która aktualizuje swój stan wewnętrzny.
- Dane wyjściowe są generowane na podstawie aktualnego stanu wewnętrznego.
- Proces jest powtarzany dla kolejnych kroków czasowych, przekazując informacje z poprzednich

kroków.

11.1 Unrolling (rozwijanie)

Proces rozwinięcia lub dekompresji sieci rekurencyjnej na wielu krokach czasowych. W standardowej definicji RNN, model jest reprezentowany jako powtarzające się jednostki, które operują na danych wejściowych w każdym kroku czasowym. Jednak w celu lepszego zrozumienia i wizualizacji działania sieci, często stosuje się unrolling.

Podczas unrollingu, sieć rekurencyjna jest rozwinięta wzdłuż osi czasu, tworząc sekwencję powiązanych ze sobą jednostek. Każda jednostka reprezentuje stan wewnętrzny (np. LSTM lub GRU) oraz warstwę wyjściową, która otrzymuje dane wejściowe z danego kroku czasowego i generuje dane wyjściowe dla tego kroku. Te powiązane jednostki są połączone ze sobą, przechodząc informacje z jednego kroku czasowego do drugiego.