

Przeskalowanie wartości tablicy do przedziału [0,1]

$$x_i^{0-1} = \frac{(x_i - \min(X))}{\max(X) - \min(X)}$$

```
arr1_0_1 = (arr1-arr1.min())/(arr1.max()-arr1.min())
print(arr1) #wyświetli [1 2 3 4 5 6]
print(arr1_0_1) #wyświetli [0. 0.2 0.4 0.6 0.8 1. ]
```

Dla 2D, skalowanie w kolumnach

$$x_{i,j}^{0-1} = \frac{(x_{i,j} - \min(x_{:,j}))}{\max(x_{:,j}) - \min(x_{:,j})}$$

```
arr_0_1 = (arr - arr.min(axis=0))/(arr.max(axis=0)-arr.min(axis=0))
print(arr_0_1)
Wyświetli: [[0. 1. 0.75 0.22222222]
            [1. 0. 0. 1. ]
            [1. 0. 1. 0. ]]
```

Metody/atributy ndarray (tablica numpy) np

Atrybut	Przeznaczenie
.ndim	liczba wymiarów
.shape	rozmiary poszczególnych wymiarów
.T	zwraca macierz transponowaną
Metody	przeznaczenie
.sum(oś)	zwraca sumę
.prod(oś)	zwraca iloczyn
.mean(oś)	zwraca średnią
.std(oś)	zwraca odchylenie standardowe
.reshape(nowe_rozmiary)	zwraca tablicę przekształconą do rozmiarów, podanych jako lista albo krotka
.sort(oś, metoda, kolejność)	sortuje tablicę za pomocą wybranej metody w podanej kolejności

## Pandas pd

przetwarzanie danych, zapis/odczyt danych z pliku, podgląd danych. Ładuje dane do zmiennej typu DataFrame.

```
data_csv = pd.read_csv("dane_oddzielone_srednikami.csv", sep=';')
data_excel = pd.read_excel("dane_w_ekselu.xlsx")
nazwy_kolumn = list(data_csv.columns)
wartosci_kolumn = data_csv.values
```

## Matplotlib plt

```
x = np.arange(0, 10, 0.1)
y = np.sin(x**2 - 5*x + 3)
plt.scatter(x,y)
plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('y')
```

Kilka poleceń - wykresy nakładają się na siebie. Nie chcesz tak? Subplots.

## Subplots

```
fig, ax = plt.subplots(1,2, figsize = (10, 5))
ax[0].plot(x,y)
ax[0].set_xlabel("x")
ax[0].set_ylabel("y")
ax[1].scatter(x,y)
ax[1].set_xlabel("x")
ax[1].set_ylabel("y") fig.tight_layout()
```

Wiele wykresów w postaci dwuwymiarowej tablicy

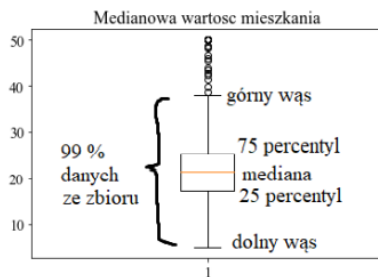
```
fig, ax = plt.subplots(2,2, figsize = (10, 10))
ax[0,0].scatter(x,y)
ax[0,1].plot(x,y)
ax[1,0].hist(y)
ax[1,1].boxplot(y)
```

Miary jakości pracy modelu - metryki regresji

```
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
```

## Wykres pudełkowy

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, shuffle=True)
plt.boxplot(y_train)
plt.title("Medianowa wartosc mieszkania")
```



Rys. 2.5. Wykres pudełkowy cechy zależnej.

- pudełko:
  - o dolna granica pudełka: 25 percentyl – dla 25 % obserwacji cecha ma wartość poniżej tej wartości,
  - o pomarańczowa linia: mediana – dla 50 % obserwacji cecha ma wartość poniżej tej wartości,
  - o górna granica pudełka: 75 percentyl – dla 75 % obserwacji cecha ma wartość poniżej tej wartości,
- wąsy: istnieją różne definicje, w przybliżeniu można powiedzieć: 99% danych znajduje się pomiędzy wąsami, czyli wszystkie „normalne” obserwacje znajdują się między wąsami, można w przybliżeniu przyjąć, że 99% obserwacji należy do przedziału:  $[\bar{x} - 3 \cdot \sigma, \bar{x} + 3 \cdot \sigma]$
- wartości odstające (ang. outliers) – obserwacje, których wartość cechy nie należy do podanego przedziału.

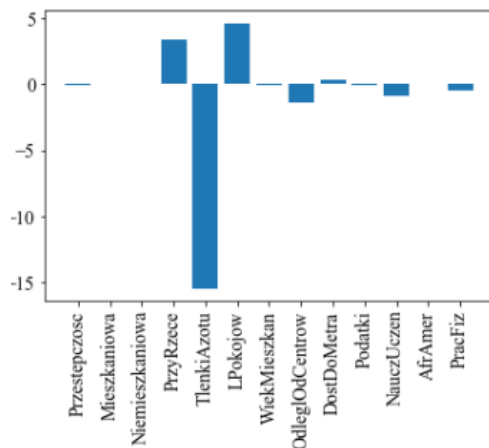
Odstające zakłócają pracę modelu, więc się je usuwa lub zastępuje wartością średnią.

```
outliers = np.abs((y_train - y_train.mean()) / y_train.std()) > 3
X_train_no_outliers = X_train[~outliers, :]
y_train_no_outliers = y_train[~outliers]
y_train_mean = y_train.copy()
y_train_mean[outliers] = y_train.mean()
```

Współczynniki modelu regresji liniowej = wagi pokazują, jak ważna jest cecha, jak wpływa na wynik

```
1.linReg = LinearRegression()
linReg.fit(X_train, y_train_mean)
niezależne_cechy = bh_cechy[:-1]
fig, ax = plt.subplots(1, 1)
x = np.arange(len(niezależne_cechy))
wagi = linReg.coef_
ax.bar(x, wagi)
ax.set_xticks(x)
```

`ax.set_xticklabels(niezleżne_cechy, rotation = 90)`



Mogą wpływać pozytywnie lub negatywnie. Poniżej osi - negatywnie. Powyżej - pozytywnie.

Można połączyć cechy, tworząc nowe z iloczynów/ilorazów cech.

```
nowe_dane = np.stack([X[:, 4]/X[:, 7],
X[:, 4]/X[:, 5],
X[:, 4]*X[:, 3],
X[:, 4]/X[:, -1]], axis=-1)
X_additional = np.concatenate([X, nowe_dane], axis=-1)
```

Regresja liniowa to przypadek aproksymacji wielu zmiennych.

Przy regresji liniowej metryką jakości jest średni procent błędu przy predykcjach.

Klasyfikacja - jakościowo - tak/nie, należenie do zbioru

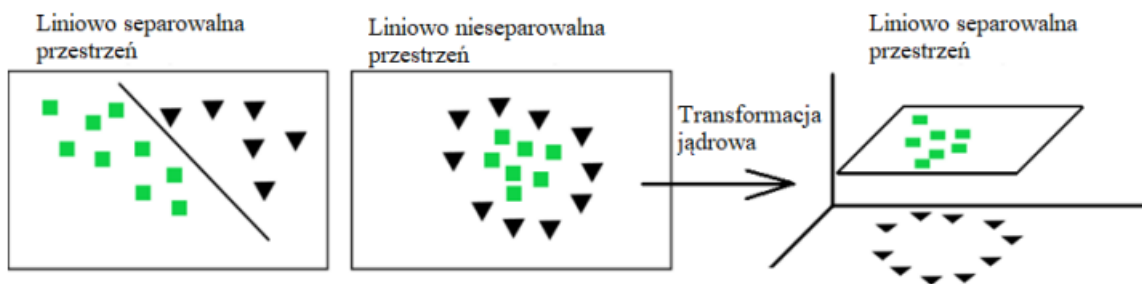
k Nearest Neighbors (kNN)

Support Vector Machine (SVM)

w dla każdej klasy (dzielnicy) zostaje utworzona specjalna funkcja, wyznaczająca jej granicę. Ta funkcja, przyjmując na wejściu obserwację, zwróci 1 albo -1 w zależności od tego, czy ta obserwacja znajduje się w tej dzielnicy, czy nie.

granica dzieląca klasy zostaje wyznaczona w taki sposób, że odległość od granicy najbliższych obserwacji poszczególnych klas jest możliwie największa. Takie rozwiązanie ma zapewnić najlepszą jakość separacji.

Czasami separacja liniowa jest zupełnie niemożliwa, tak, jak w przypadku Śródmieścia: jest ono otoczone innymi osiedlami (klasami), zatem konieczne jest zastosowanie jakiejś transformacji, która mogłaby wyróżnić tę klasę. Takie transformacje nazywane są jądrami, przykład takiego jądra przedstawia Rys. 3.4. Ideą takiej transformacji jest wprowadzenie nowych cech będących kombinacjami istniejących: na przykład ich iloczyny, ilorazy itd.



Rys. 3.4. Przykład liniowej separalności/nieseparowalności oraz zastosowania jądra w SVM.

cecha zależna - udzielona pożyczki lub nie -> zbiór wartości cechy zależnej zawiera 2 elementy -> klasyfikacja binarna

jakościowe na ilościowe -> binarnie

```
mask = data ['Gender'].values == 'Female'
```

```
data['Gender'][mask] = 1
```

```
data ['Gender'][~mask] = 0
```

One-Hot encoding 1-z-n

```
cat_feature = pd.Categorical(X.Property_Area) //przekształcenie jakościową na zmienną kategoryjną
```

```
one_hot = pd.get_dummies(cat_feature) //zmiana na tabelę zakodowaną 1-z-n
```

```
data = pd.concat([data, one_hot], axis = 1) //dołączenie nowej
```

```
data = data.drop(columns = ['Property_Area']) //usunięcie starej
```

Cecha jakościowa	Urban	Semiurban	Rural
Urban	1	0	0
Urban	1	0	0
Rural	0	0	1
Semiurban	0	1	0

Macierz pomyłek

		Pozytywny	Negatywny
Prawdziwe dane	Chory	Osoba chora – test pozytywny <b>True Positive (TP)</b>	Osoba chora – test negatywny <b>False Negative (FN)</b>
	Zdrowy	Osoba zdrowa – test pozytywny <b>False Positive (FP)</b>	Osoba zdrowa – test negatywny <b>True Negative (TN)</b>

czułość recall =  $TP/(TP+FN)$  ile chorych wykryje z chorych

precyzja precision =  $TP/(TP+FP)$  ile chorych prawidłowo z pozytywnych

dokładność accuracy =  $(TP + FN)/(TP+TN+FP+FN)$  poprawne odpowiedzi do wszystkich odpowiedzi

swoistość specyfity specyficzność =  $TN/(TN+FP)$

Sama dokładność nie znaczy, że jest super. Konieczność skalowania danych.

```
cm = confusion_matrix(y_test, y_pred)
```

Standaryzacja - ujednolicenie skali cech analizowanego zbioru

$$x_{i,j}^{std} = \frac{(x_i - mean(x_{.,j}))}{std(x_{.,j})}$$

scaler = StandardScaler(), MinMaxScaler(),

RobustScaler()

Scaler wyłącznie na zbiorze treningowym

scaler = StandardScaler()

scaler.fit(X\_train)

X\_trainscaler.transform(X\_train)

Drzewo decyzyjne - wyjątek od reguły, że należy skalować dane

Dlaczego? Ponieważ w procesie uczenia modelu (budowy drzewa) cechy są rozpatrywane niezależnie od siebie.

Najbardziej dostosowany do odczytania przez człowieka model uczenia maszynowego.

Drzewo binarne. Każdy węzeł zawiera warunek podziału zbioru (cechę) oraz wartość graniczną. Lewo to wartości  $\leq$  granicznej, prawo  $>$

Skalowanie niewskazane, pogarsza czytelność.

Najważniejszy parametr - maksymalna głębokość.

Przeuczenie - model za bardzo przystosowuje się do zestawu, na którym pracuje.

Jeśli drzewo zbyt głębokie, to zbyt szczegółowo rozpatruje każdy przypadek i się dostosowuje.

Nagrania dźwiękowe mają za wiele wymiarów, więc nie można bezpośrednio analizować. Przed dokonaniem klasyfikacji często wykonuje się wstępną ekstrakcję cech. Cechami mogą być cechy statystyki opisowej - średnia, odchylenie standardowe, współczynnik skośności.

FFT Analiza Fouriera

fft(tabela, oś) - tak łatwo. wynik to tablica liczb zespolonych. Trzeba to jeszcze przetworzyć

Uzyskane widmo amplitudy jest symetryczne - taka cecha. Odrzuca się połowę widma.

Analiza składowych głównych PCA primary component analysis

Obrót przestrzeni z punktami cech, że cechy będą zorientowane w kierunku największej zmienności naszej chmury. Największej ilości informacji odpowiada największa zmienność.

Po transformacji pierwsza cecha (kolumna) najwięcej informacji zawiera.

Można zastosować do zbioru o dowolnej liczbie cech (wymiarów). Jeśli cecha nie wnosi informacji, można się jej pozbyć.

Można zastosować do wstępnej analizy danych - obrazowanie zbioru o dużej liczbie wymiarów w układzie dwuwymiarowym. PCA ekstrahuje 2 składowe główne.

Po dokonaniu transformacji uzyskany zbiór traci bezpośrednie powiązanie ze światem rzeczywistym, bo składowe główne są kombinacjami liniowymi cech pierwotnych. Trudno interpretować model wytrenowany na takich składowych głównych.

X\_pcaed = PCA(2).fit\_transform(X\_train)

Analiza składowych niezależnych ICA independent component analysis

podział sygnału na składowe niezależne (najmniej powiązane ze sobą)

```
transformer = FastICA(n_components=7, random_state=0)
```

### Pipeline

W obiekcie tej klasy łączy się wszystkie elementy procedury przetwarzania danych. Trening obiektu odbywa się za pomocą fit. do Pipeline przekazuje się listę par nazwa-obiekt

```
pipe = Pipeline([['transformer', PCA(62)],
['scaler', StandardScaler()],
['classifier',
kNN(weights='distance')]]) //ostatni obiekt musi mieć zaimplementowane metody fit i predict
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
```

### Sieć neuronowa, model sztucznego neurona

sztuczny neuron - funkcja matematyczna modelująca proces podjęcia decyzji. Jako dane wejściowe przyjmuje ona wartości cech, na wyjściu zwraca liczbę będącą odpowiedzią. Odpowiedź jest wynikiem zastosowania funkcji aktywacyjnej na sumie ważonej wartości cech. W najbardziej podstawowym modelu sztucznego neurona funkcją aktywacyjną jest skokowa funkcja, która może zwracać jedną z dwóch wartości: 0 albo 1. Innymi słowy, taki neuron będzie dokonywał klasyfikacji binarnej, odpowiadając 'Tak'(1) lub 'Nie'(0).

Odpowiedź neuronu (wyjście) - funkcja aktywacji.

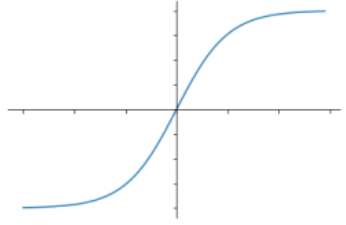
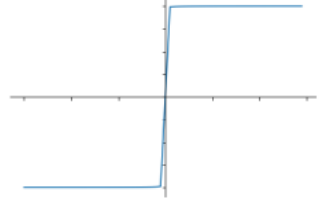
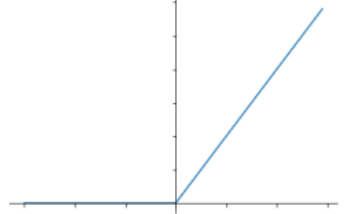
Wartość wyjścia to prawdopodobieństwo sieci. Suma wartości = 1.

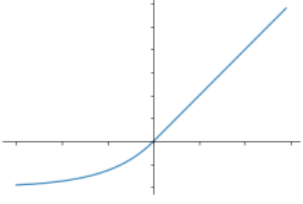
Tyle prawdopodobieństw, ile klas.

### Funkcje aktywacji

Logistyczna =  $1/(1+e^{-a*s})$

Skokowa = 0  $s \leq \text{próg}$ , 1  $s > \text{próg}$

Nazwa	Wzór	Wykres
Tangens hiperboliczny	$f_{akt}(x) = \tanh x \quad (5.5)$	
SoftSign	$f_{akt}(x) = \frac{x}{\epsilon +  x } \quad (5.6)$ gdzie $\epsilon$ to parametr od którego zależy stromość krzywej. Im jest on mniejszy, tym bardziej stroma jest krzywa.	
ReLU – Rectifier Linear Unit	$f_{akt}(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases} \quad (5.7)$	

ELU – Exponential Linear Unit	$f_{akt}(x) = \begin{cases} \alpha(e^x - 1), & x \leq 0 \\ x, & x > 0 \end{cases} \quad (5.8)$	
SELU – Scaled Exponential	$f_{akt}(x) = \lambda \cdot ELU(x) \quad (5.9)$	Taki sam, jak dla ELU, tylko z uwzględnieniem współczynnika $\lambda$
Softmax - typowa funkcja wyjściowa w zadaniach klasyfikacji wieloklasowej	$f_{akt_i}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^N e^{x_j}} \quad (5.10)$	-

Do konstruktora pierwszej warstwy trzeba przekazać liczbę cech wejścia jako rozmiar wejścia.

Do konstruktora ostatniej warstwy trzeba przekazać liczbę klas jako rozmiar wyjścia.

Minimalną jednostką sieci neuronowej w bibliotece Keras jest warstwa

W przypadku, kiedy mamy do czynienia z klasyfikacją wieloklasową, wynik jest zwracany jako wektor prawdopodobieństw przynależności obserwacji do poszczególnych klas, dlatego jako funkcja aktywacji ma być zastosowana funkcja 'softmax'

Jako funkcję strat w przypadku klasyfikacji wieloklasowej najczęściej używa się funkcji o nazwie categorical\_crossentropy

```
model.fit(X_train, y_train, batch_size=32, epochs=5000, validation_data=(X_test, y_test), verbose=2)
```

batch - karmimy porcjami

epochs - uczy się na przeciągu kilku iteracji.

verbose - komunikaty o postępach uczenia

Przetrenowanie sieci neuronowej

wzrostowi jakości klasyfikacji na zbiorze treningowym przestaje towarzyszyć wzrost na zbiorze testowym

jakość pracy modelu na zbiorze testowym zaczyna się pogarszać

zbyt wysokim dopasowaniem modelu do danych uczących, w skutek czego zaczyna on tracić zdolność do uogólnienia

zaczyna uczyć się rozpoznawać szum

Przetrenowanie występuje dla wszystkich modeli uczenia maszynowego, ale najbardziej zauważalne jest w przypadku sieci neuronowych. Jest to spowodowane dużą liczbą parametrów, czyli zdolnością do aproksymacji praktycznie każdej funkcji. Im więcej ma sieć parametrów (warstw, neuronów) tym bardziej jest ona skłonna do przetrenowania.

Walidacja krzyżowa

sieci neuronowe odróżniają się od innych metod uczenia maszynowego tym, że posiadają



bardzo dużą liczbę hiperparametrów, które można regulować, szukając najlepszego ustawienia. Do takich hiperparametrów można zaliczyć takie wartości, jak:

1. liczba warstw, 2. liczba neuronów w poszczególnych warstwach,
3. funkcja aktywacji,
4. rozmiar porcji podawanej do modelu podczas uczenia (batch),
5. prędkość nauczania,
6. stosowany optymalizator.

Jak dobierać hiperparametry?

zbiór należy podzielić na dwa podzbiory: uczący i testowy oraz przeprowadzić walidację krzyżową na zbiorze treningowym

1. dane zostają podzielone na dwa podzbiory: uczący oraz testowy, testowy zostaje odłożony, nie występuje w procesie uczenia,
2. wybieramy kombinację hiperparametrów,
3. zbiór uczący zostaje podzielony na  $N$  części (na rysunku – 5)
4. jedna z  $N$  części (numer  $i$ ) zostaje odłożona – będzie ona pełniła rolę zbioru walidacyjnego – w tym obrocie walidacji nie będzie brała udziału
5. dla wybranej kombinacji hiperparametrów zostaje przeprowadzony proces uczenia z wykorzystaniem pozostałych  $N-1$  części,
6. na odłożonej części zostaje przeprowadzone testowanie nauczonego modelu – wynik (wartość wybranej metryki, na przykład – dokładności) zostaje zapisany
7. kroki 3-5 zostają powtórzone  $N$  razy dla pozostałych części, w wyniku uzyskujemy  $N$  wartości metryki, uśredniamy je, dostając jedną wartość z kroku.

2 klasy

wyjscie ilość neuronów 1

f aktywacji na wyjściu logistyczna sigmoid

f loss strat binary\_crossentropy

>2 klasy

na wyjściu tyle neuronów, ile klas =  $n$

softmax

categorical\_crossentropy

- Dropout – warstwa, w której zostają deaktywowane losowo wybrane neurony – ma znaczenie regularyzacyjne,
- GaussianNoise – warstwa, w której do wyjść poprzedniej warstwy zostaje dodany szum o rozkładzie Gaussowskim, posiadający zdefiniowane: średnią oraz odchylenie standardowe, ma znaczenie regularyzacyjne,
- BatchNormalization – warstwa, w której obserwacje jednej porcji (batch) zostają znormalizowane w sposób podobny do normalizacji Z-Score (odejmowanie średniej oraz dzielenie przez odchylenie standardowe) - pozwala na przyspieszenie uczenia. Normalizacja w ramach wsadów (tak jak StandardScaler)
- LayerNormalization – warstwa, w której wyjścia jednej warstwy zostają znormalizowane w sposób podobny do normalizacji Z-Score (odejmowanie średniej oraz dzielenie przez odchylenie standardowe)
- Dropout - wyłączenie części neuronów warstwy o numerze  $i$  zanim ich wyjścia zostaną przekazane do warstwy  $i + 1$ ,

- GaussianNoise – dodanie do sygnału oraz wyjść warstw ukrytych losowego szumu o zerowej średniej i niewielkim odchyleniu standardowym,
- dodanie do funkcji strat członu regularyzacyjnego. Najczęściej regularyzacja bywa dwóch rodzajów: L1 oraz L2.

Sieci gęsto połączone

każdy neuron  $i$ -tej warstwy połączony jest z każdym neuronem warstwy  $i - 1$ .

jeżeli  $i$ -ta warstwa ma  $n$  neuronów, a  $i - 1$  warstwa ma  $m$  neuronów, całkowita liczba parametrów  $i$ -tej warstwy będzie równa:  $(m + 1) \cdot n$ .

liczba parametrów rośnie bardzo szybko wraz ze wzrostem liczby neuronów

Rozpoznawanie obrazów - sieci konwolucyjne

przynajmniej jedna warstwa konwolucji (splot)

$Out_{i,j}$  – element wyjścia o indeksach  $i$  oraz  $j$ ,

$In_{.,.}$  - element wejścia o podanych indeksach,

$w_{p,q}$  – wartość wagi filtra konwolucyjnego o indeksach  $p$  oraz  $q$ ,

$w_0$  – wyraz wolny,

$m$  – wysokość filtra konwolucyjnego,

$n$  – szerokość filtra konwolucyjnego.

$$Out_{i,j} = \sum_{p=0}^{m-1} \sum_{q=0}^{n-1} \left( In_{i+p-\frac{m}{2}, j+q-\frac{n}{2}} \cdot w_{p,q} \right) + w_0$$

Aby wyznaczyć jeden element wyjścia, warstwa konwolucyjna uwzględnia tylko ograniczony obszar wejścia, a nie jego całość

Obraz jako macierz kolorów RGB 3 wymiary, BW - 2 wymiary. BW można przerobić na 1 wymiar - słupki. To wejście do funkcji.

Konwolucja sprowadza się do tworzenia filtrów konwolucyjnych, które wykrywają kształty. Do RGB potrzebne 3 filtry, do BTW wystarczy 1.

```
x_train_scaled = (x_train/255).copy()
```

dlatego tutaj było dzielenie  $x\_train/255$  bo piksele wartości 0-255

ważne przy budowie enkodera - należy odpowiednio przygotować dane

`conv_rule`, która definiuje regułę zastosowania filtra konwolucyjnego.

'valid' – wyznaczane są wartości tylko na podstawie pierwotnego wejścia: przez to rozmiary wyjścia są mniejsze od odpowiednich rozmiarów wejścia, 'same' – rozmiar wejścia jest równy rozmiarowi wyjścia: taki efekt zapewnia uzupełnienie wejścia zerami.

`Flatten()` - warstwa spłaszczająca

```
.model.add(Conv2D(input_shape = X_train.shape[1:],
```

```
filters=filter_cnt,
```

```
kernel_size = kernel_size,
```

```
padding = conv_rule,
```

```
activation = act_func))
```

```
model.add(Flatten())
```

```
2model.add(Dense(class_cnt,activation='softmax'))
```

```
model.compile(optimizer=Adam(learning_rate),
```

```
loss='SparseCategoricalCrossentropy',
```

```
metrics='accuracy')
```

Warstwy głosujące (pooling)

Max pooling - pozwalają rozpoznać kształt na obrazku niezależnie od położenia kształtu na obrazku

Avg pooling - zmniejsza, skaluje "obrazek", "wzór"

tensorem nazywamy obiekt przechowujący tablicę wielowymiarową. Na przykład, do sieci chcemy przekazać zbiór zdjęć liczący 100 obserwacji, a każde zdjęcie ma rozmiary 640\*480 pikseli, jest kolorowe, kolory przechowywane są za pomocą przestrzeni RGB. W takim przypadku musimy przekazać dane jako tensor o wymiarach [1000, 640, 480, 3].

Warstwa konwolucyjna oczekuje na wejściu tensor o liczbie wymiarów równej 4, dlatego szykując dane do klasyfikacji za pomocą sieci konwolucyjnej, jeżeli mamy do czynienia ze zdjęciami w skalach szarości, trzeba sztucznie dodać jeszcze jeden wymiar o rozmiarze 1

Jeżeli spojrzymy na sieć neuronową od strony teorii grafów, można będzie powiedzieć, że jest ona grafem obliczeniowym. Węzłami w tym grafie są warstwy, a krawędziami – połączenia warstw między sobą.

```
output_tensor = input_tensor = Input(X_train.shape[1:])
```

sieć Inception

Concat - warstwa scalająca

warstwa lambda - służy do dodania funkcji zdefiniowanej przez użytkownika

```
model = sequential()
model.add(Dense(liczba_neuronów, aktywacja, rozmiar_wejścia))
model.add(Dense(liczba_neuronów, aktywacja))
model.add(Dense(liczba_klas, aktywacja='softmax'))
model.compile(optimizer, f_strat, metryki)
```

tensory:

```
input_tensor = Input(rozmiar_wejścia)
layer=Dense(...)
tensor = layer(input_tensor)
```

Autoenkoder - uczenie maszynowe nienadzorowane lub pół nadzorowane

Autoenkoder przetwarza na inną liczbę wymiarów

Enkoder robi konwersję - z 20 wymiarów do 3

Kompresja

Dekoder odtwarza skompresowane informacje

Autoenkoder = enkoder + dekodek

Cechy skompresowane to cechy ukryte.

Autoenkoder minimalizuje różnicę między przetworzonymi a prawdziwymi. Wyciąga najistotniejsze by dekodek mógł odtworzyć te informacje.

Skompilowany oraz nauczony musi zostać tylko jeden model: autoenkoder. Pozostałe dwa będą korzystały z parametrów jego warstw.

Po co?

- usuwanie szumu (autoenkoder, stare filmy)
- klasteryzacja (sam enkoder)
- wykrywanie anomalii (enkoder)
- generacja nowych danych (dekoder)

Ostatnia warstwa autoenkodera ma funkcję aktywacji sigmoid – czyli jest to funkcja logistyczna

Aby odtworzyć z powrotem tensor wejściowy w dekodерze stosowane są warstwy typu UpSampling. Działają odwrotnie do warstw głosujących – po ich zastosowaniu rozmiary tensora zostają podwojone

```
#ostatnia warstwa ma aktywacje sigmoid.  
#upSampling dziala odwrotnie do maxpool - rozszerza 2 razy  
szerokosc, 2 razy dlugosc  
#ostatnia warstwa ma tez 1 kanal...  
#output_tensor po kolei jest trawiony przez kolejne warstwy  
[przetwarzany]
```

Bardzo ważne w przypadku autoenkodera usuwającego szum jest dodanie warstwy odpowiedzialnej za wprowadzenie szumu do danych: dzięki temu model od samego początku będzie dostawał zaszumione dane oraz uczył się wykrywać w nich najważniejsze szczegóły tak, żeby pominąć szum podczas odtwarzania obserwacji pierwotnej.

Generowanie danych - obroty

```
#expand = 0 - obrazek zachowa rozmiar, moze zostac przycięty  
#expand = 1 - obrazek moze zwiekszyć swój rozmiar, by się zmieścić  
#Image.fromarray(X_train[0,...]).rotate(kat, expand = 1)  
kat = np.random.randint(-60,60)  
obrocony_but = Image.fromarray(X_train[0,...]).rotate(kat, expand  
= 0)  
obrocony_but = np.array(obrocony_but).astype(np.float32)  
angle = np.random.randint(-30,30)  
img = Image.fromarray(img).rotate(angle, expand = 1).resize(img_size)  
rotated_images[i] = np.array(img)
```

```
angle = np.random.randint(-30,30)  
left, upper = np.random.randint(0, 5, 2)  
right, lower = np.random.randint(23, 28, 2)  
img = Image.fromarray(img).crop((left, upper, right, lower)).resize(img_size)  
rotated_images[i] = np.array(img)
```

LSTM - warstwa z krótko i długotrwałą pamięcią  
liczba obserwacji m-n m liczba dni, n rozmiar obserwacji na podstawie której przewidujemy następne

```
X,y = make_X_y(needed_data, 100, 2 ) #100 dni
#nie mozemy zastostowac train test split
#nie mozemy powybierac losowo 20%
#podzial w szeregach czasowych odbywa sie inaczej
#te wartosci sa blisko siebie, moga byc bardzo podobne - dlatego
nie mozna.
```

**podaj wzor ktory pozwala na wykonanie normalizacji srednia std**

```
norm_arr = (arr - arr.mean())/(arr.std())
norm_arr_kolumny=(arr - arr.mean(axis=0))/(arr.std(axis=0)+np.spacing(arr.std(axis=0)))
srednie_kolumn = arr.mean(axis=0)
albo przeskalowanie min max
max_arr=arr.max(axis=0)
max_val= max_arr.max(axis=0)
```

**podaj metody klasyfikacji ktore nie wymagaja skalowania** - drzewo decyzyjne

**podaj poprawna sekwencje czynnosci**

przygotowanie danych  
podzial na zbiory do uczenia i testowania  
podzial na train test  
skalowanie  
uczenie  
wyznaczanie metryk

**skalowanie(przy czym podajemy dane treningowe)**

**funkcje aktywacji do wykonania jedno klasowej(sigmoid)**

**podaj wyrazenie ktore pozwala na sumowanie kolumn parzystych nieparzystych**

```
parzyste_wiersze = arr[:,2,:]
nieparzyste_wiersze = arr[1::2,:]
```

**podaj wyrazenie co pozwala na wyzerowanie wykresu punktowego scatter - ?**

analiza skladowych niezaleznych i furiera nie bedzie, ale glownych tak

**co to srednia kwadratowy blad** - funkcja strat

**co to czulosc precyzja itd** - miary jakosci klasyfikacji (metryki)

czułość recall =  $TP/(TP+FN)$  ile chorych wykryje z chorych  
 precyzja precision =  $TP/(TP+FP)$  ile chorych prawidłowo z pozytywnych  
 dokładność accuracy =  $(TP + TN)/(TP+TN+FP+FN)$  poprawne odpowiedzi do wszystkich odpowiedzi  
 swoistość specificity specyficzność =  $TN/(TN+FP)$   
 Sama dokładność nie znaczy, że jest super. Konieczność skalowania danych.  
`cm = confusion_matrix(y_test, y_pred)`

### 1. Wskazać cechy jakościowe/iłościowe

iłościowe da się policzyć, zrobić  $\geq$ ,  $>$ ,  $<$ ,  $\leq$ . jakościowe można tylko  $=$  lub  $\neq$ , np. rasa psa jest taka sama lub różna. dalmatyńczyk  $>$  owczarek niemiecki nie ma sensu

### 2. Wzór na odległość manhattańską między sąsiadami w kNN

#### Odległość Manhattan (taksówkowa)

Suma odległości po współrzędnych

$$d(x, y) = \sum_i |x_i - y_i|.$$

### 3. Kod do tworzenia kilku wykresów w jednym obrazku

```
fig, ax = plt.subplots(2,2, figsize = (10, 10))
ax[0,0].scatter(x,y)
ax[0,1].plot(x,y)
ax[1,0].hist(y)
ax[1,1].boxplot(y)
```

### 4. Wskazać metryki klasyfikacji czy coś takiego

klasyfikacji?

czułość recall =  $TP/(TP+FN)$  ile chorych wykryje z chorych sensitivity  
 precyzja precision =  $TP/(TP+FP)$  ile chorych prawidłowo z pozytywnych  
 dokładność accuracy =  $(TP + FN)/(TP+TN+FP+FN)$  poprawne odpowiedzi do wszystkich odpowiedzi  
 swoistość specificity specyficzność =  $TN/(TN+FP)$

$F1 = 2 \cdot (\text{sensitivity} \cdot \text{precision}) / (\text{sensitivity} + \text{precision})$  - dokładniejsza

Sama dokładność nie znaczy, że jest super. Konieczność skalowania danych.

`cm = confusion_matrix(y_test, y_pred)`

### 6. Jak przeskalować dane do przedziału [1,0] -> MinMaxScaler()

### 7. Jaka architektura wykonuje sekwencyjne operacje porównania - drzewo decyzyjne?

i chyba trzeba było wybrać wzór który przeskaluje dane tak, aby wartość średnia w kolumnach była równa 0 a odchylenie standardowe równe 1 czy coś takiego

### 8. Wskazać funkcję która utworzy 2 wykresy obok siebie typu punktowego

```
fig, ax = plt.subplots(1,2, figsize = (10, 5))
ax[0].scatter(x,y)
ax[0].set_xlabel("x")
ax[0].set_ylabel("y")
ax[1].scatter(x,y)
ax[1].set_xlabel("x")
```

```
ax[1].set_ylabel("y")
fig.tight_layout()
```

**9. wskazać warstwy głosujące- AveragePooling2D, MaxPooling2D**

**10. Jak odbić obraz w poziomie wskazać komendę -**

```
X_train_odbite_poziomo = X_train[:, :, ::-1]
X_train_odbite_pionowo = X_train[:, ::-1, :]

kat = np.random.randint(-60, 60)
obrocony_but = Image.fromarray(X_train[0, ...]).rotate(kat, expand
= 0)
obrocony_but = np.array(obrocony_but).astype(np.float32)
#expand = 0 - obrazek zachowa rozmiar, może zostać przycięty
#expand = 1 - obrazek może zwiększyć swój rozmiar, by się zmieścić
#Image.fromarray(X_train[0, ...]).rotate(kat, expand = 1)
plt.imshow(obrocony_but)
```

**11. Funkcje aktywacji bez granicy górnej - RELU ELU SELU**

**12. Jaka funkcja strat dla klasyfikacji dwuklasowej - sigmoid**