PRiAD 4

Uczenie nadzorowane

Klasyfikacja zalicza się do metod uczenia nadzorowanego (ang. supervised learning). Zbiór (macierz) danych w tym przypadku składa się z obiektów charakteryzujących się atrybutami opisującymi i atrybutem decyzyjnym. Przyjmuje się przy tym, że pomiędzy atrybutami opisującymi, a atrybutem decyzyjnym zachodzi pewien związek przyczynowo-skutkowy zaś kategoryczny atrybut decyzyjny określa klasę do której przynależy obiekt. Zbiór danych jest traktowany jako źródło wiedzy na podstawie którego określa się rodzaj klasyfikatora, a następnie dobiera jego parametry w tzw. procesie uczenia. Gotowy klasyfikator może następnie zostać wykorzystany do określenia przynależności do właściwej klasy (nowego) obiektu dla którego znane są jedynie wartości atrybutów opisujących.

Z reguły w celu sprawdzenia poprawności danego algorytmu i sprawdzenia jego skuteczności dla danego zadania klasyfikacji podział zbioru danych wykonywany jest na zbiór uczący i testowy (najczęściej w proporcjach 80/20 lub 70/30).

W ćwiczeniu pokazane zostaną następujące metody klasyfikacji:

1. Metoda najbliżego sąsiada
2. Metoda k-najbliższych sąsiadów
3. Metoda najbliższego prototypu
4. Naiwny klasyfikatory Bayesa
5. Drzewa decyzyjne

Na początek jednak, tradycyjnie, zostaną wczytane niezbędne pakiety.

In [1]:
```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

# zmiana sposobu wyświetlania danych typu float
pd.options.display.float_format = "{:.2f}".format
```

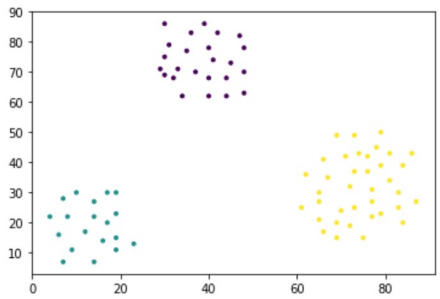
Dodatkowo, wczytane zostaną procedury niezbędne do realizacji zadań klasyfikacji z pakietu `scikit-learn`

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import NearestNeighbors
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import NearestCentroid
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
```

1. Klasyfikator najbliższego sąsiada

Zasadą działania metod najbliższego sąsiada (1-NN) jest poszukiwanie najbliższego sąsiada dla nowego obiektu o nieznanej klasie, wśród obiektów znajdujących się w zbiorze uczącym. Klasa, do której najbliższy sąsiad przynależy jest przypisywana klasyfikowanemu obiektowi. Poniższy przykład pokazuje wyszukiwanie najbliższych sąsiadów.

In [3]:
```
df = pd.read_csv('dane1.csv')
plt.scatter(df.loc[:, 'atrybut1'], df.loc[:, 'atrybut2'], c=df["klasa"].astype('category').cat.codes, marker = '.')
nbrs = NearestNeighbors(n_neighbors=3, algorithm='ball_tree')
nbrs.fit(df[['atrybut1','atrybut2']])
distances, indices = nbrs.kneighbors(df[['atrybut1','atrybut2']])
```



In [4]:
```
nbrs_wynik = pd.DataFrame({'atrybut1':df.loc[:, 'atrybut1'],
                           'atrybut2':df.loc[:, 'atrybut2'],
                           'Najbliższy': indices[:,1],
                           'Odległość od najbl.': distances[:,1],
                           'Drugi najbl.': indices[:,2],
                           'Odległość od drugiego': distances[:,2]})
nbrs_wynik.head(20)
```

Out[4]:

	atrybut1	atrybut2	Najbliższy	Odległość od najbl.	Drugi najbl.	Odległość od drugiego
0	86	43	8	4.47	9	5.00
1	79	50	10	5.10	2	6.08
2	73	49	3	4.00	4	6.08
3	69	49	2	4.00	26	7.28
4	74	43	11	2.24	26	3.16
5	83	25	6	4.47	21	4.47
6	87	27	5	4.47	14	5.00
7	84	20	5	5.10	21	5.83
8	84	39	0	4.47	9	5.00
9	81	43	10	3.61	12	4.47
10	78	45	9	3.61	11	3.61
11	76	42	4	2.24	10	3.61
12	79	39	25	3.61	11	4.24
13	81	34	14	4.47	23	5.00
14	83	30	13	4.47	5	5.00
15	66	17	18	3.61	16	4.12
16	65	21	30	4.12	15	4.12
17	61	25	36	4.47	16	5.66
18	69	15	15	3.61	29	5.00
19	75	15	29	5.00	18	6.00

Pytanie Czy relacja "x jest najbliższym sąsiadem y" jest symetryczna ? Zastanów się czy tak jest i sprawdź w powyższych wyniakach.

W przypadku pojedynczego zbioru danych, w celu sprawdzenia działania (każdego) klasyfikatora, należy podzielić zbiór danych na uczący i testowy. W tym celu wykorzystuje się funkcję `train_test_split` . Funkcja ta zarówno na wejściu jak i na wyjściu wymaga podania osobno atrybutów opisujących i atrybutu decyzyjnego. Taki sposób reprezentacji będzie wykorzystywany dalej przez funkcje realizujące zadania klasyfikacji. Dla ułatwienia przygotowana została funkcja realizująca podział zbioru i zapisująca zbiór uczący i testowy w jednej strukturze (słowniku).

In [5]:
```
def podziel(df,proporcja):
    # dzieli macierz (ramkę) danych na zbiór uczacy i testowy
    # df - ramka danych; proporcja - proporcja podzialu (0-1)
    # zwraca słownik z kluczami:
    # opis_ucz/opis_test - macierz atrybutów opisujących zbioru uczącego/testowego
    # dec_ucz/dec_test - wektor wartosci atrybutu decyzyjnego zbioru uczącego/testowego
    # uwaga: atrybut opisujący jest zawsze na końcu (ostatnia kolumna ramki)
    opis_ucz, opis_test, dec_ucz, dec_test = train_test_split(df.iloc[:,0:-1], df.iloc[:,-1].astype('category').cat.codes,
test_size=proporcja)#, random_state=0)
    return {"opis_ucz":opis_ucz, "opis_test":opis_test, "dec_ucz":dec_ucz, "dec_test":dec_test}

dane = podziel(df,0.3)
print('Liczba obiektów zbioru uczącego: ', len(dane["opis_ucz"]))
print('Liczba obiektów zbioru testowego: ', len(dane["opis_test"]))
```

```
Liczba obiektów zbioru uczącego:  56
Liczba obiektów zbioru testowego:  24
```

Następnie zostanie utworzony model klasyfikatora najbliższego sąsiada. Do tego celu wykorzystana zostanie funkcja `KNeighborsClassifier` , której parametr *n_neighbors* określa zadaną liczbę sąsiadów - w tym przypadku równą 1.

In [6]:
```
model = KNeighborsClassifier(n_neighbors=1)
```

Jakość klasyfikacji można oceniać przy pomocy np. macierzy pomyłek (zwanej także macierzą kontyngencji lub tabelą krzyzową), która zawiera informacje o liczbie obiektów przypisanych do klas przez wybrany model klasyfikatora dla poszczególnych wartosci atrybutu decyzyjnego. Macierz taką wyznacza się zarówno dla zbioru uczącego jak i dla zbioru testowego. Procedura `weryfikuj` wyświetla macierze pomyłek dla obu zbiorów.

In [7]:
```
def weryfikuj(model,dane,atryb):
    # wyswietla wynik weryfikacji klasyfikatora w postaci macierzy pomyłek
    # dla zbioru uczącego i testowego
    # model - model klasyfikatora
    # dane - dane (słownik zwracany przez funkcje podziel)
    # atryb - lista atrybutów uwzględnianych w weryfikacji
    model.fit(dane["opis_ucz"].iloc[:,atryb], dane["dec_ucz"])
    wynik_ucz = model.predict(dane["opis_ucz"].iloc[:,atryb])
    wynik_test = model.predict(dane["opis_test"].iloc[:,atryb])
    print("Macierz pomyłek dla zbioru uczącego")
    print(pd.crosstab(dane["dec_ucz"],wynik_ucz))
    print("Macierz pomyłek dla zbioru testowego")
    print(pd.crosstab(dane["dec_test"],wynik_test))
```

Badanie klasyfikatora wymaga wykonania następującej sekwencji czynności:

In [8]:
```
# wczytanie badanego zbioru danych
df = pd.read_csv('dane1.csv')
print(df.info())
#
#sns.pairplot(df, kind="scatter", hue = "klasa")
#plt.show()

# podział zbioru danych
d = podziel(df,0.3)
# zdefiniowanie modelu klasyfikatora
model = KNeighborsClassifier(n_neighbors=1)
# weryfikacja
weryfikuj(model,d,[0,1])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80 entries, 0 to 79
Data columns (total 3 columns):
atrybut1    80 non-null int64
atrybut2    80 non-null int64
klasa       80 non-null object
dtypes: int64(2), object(1)
memory usage: 2.0+ KB
None
Macierz pomyłek dla zbioru uczącego
col_0   0   1   2
row_0
0      19   0   0
1       0  14   0
2       0   0  23
Macierz pomyłek dla zbioru testowego
col_0  0  1   2
row_0
0      5  0   0
1      0  5   0
2      0  0  14
```

Wynik testu dla zbioru uczącego w przypadku klasyfikatora 1-NN jest oczywisty, niezależnie od danych wejściowych. Jak będzie się można przekonać w dalszej części ćwiczenia, nie będzie to juz takie oczywiste dla innych klasyfikatorów i zbiorów danych.

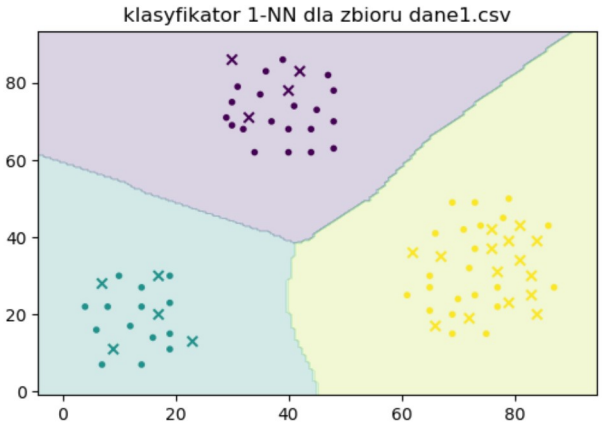
Granice decyzyjne oddzielaja obszary w przestrzeni atrybutów, które odpowiadaja poszczególnym klasom. Sposób podziału przestrzeni atrybutów zalezy przy tym od rodzaju klasyfikatora. Ponieważ granice decyzyjne mogą być czytelnie uwidocznione na wykresie punktowym dwuwymiarowym. W przypadku wiekszej liczby takich atrybutów nalezy wiec wybrac dwa sposród nich i dla niech przeprowadzic operacje wyznaczania i wizualizacji granic decyzyjnych. Do wizualizacji granic decyzyjnych została przygotowana procedura `granice`.

In [9]:
```
def granice(model,dane,atr_x, atr_y,tytul,kontur = 1):
    # wyswietla granice decyzyjne
    # model - model klasyfikatora
    # dane - dane (słownik zwracany przez funkcje podziel)
    # atr_x/atr_y - artybut wyswietlany na osi x/y
    # tytul - wyswietlany tytul wykresu
    # kontur - par. opcjonalny (=0 -> brak konturu)
    if (kontur == 1):
        model.fit(dane["opis_ucz"].iloc[:,[atr_x,atr_y]], dane["dec_ucz"])
        x_min = min(dane["opis_ucz"].iloc[:, atr_x].min(),dane["opis_test"].iloc[:, atr_x].min())
        x_max = max(dane["opis_ucz"].iloc[:, atr_x].max(),dane["opis_test"].iloc[:, atr_x].max())
        y_min = min(dane["opis_ucz"].iloc[:, atr_y].min(),dane["opis_test"].iloc[:, atr_y].min())
        y_max = max(dane["opis_ucz"].iloc[:, atr_y].max(),dane["opis_test"].iloc[:, atr_y].max())
        rozst_x = x_max - x_min
        rozst_y = y_max - y_min
        x_min = x_min - 0.1*rozst_x
        x_max = x_max + 0.1*rozst_x
        y_min = y_min - 0.1*rozst_y
        y_max = y_max + 0.1*rozst_y
        xx, yy = np.meshgrid(np.arange(x_min, x_max, (x_max-x_min)/150),
                             np.arange(y_min, y_max, (y_max-y_min)/150))
        Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)
    plt.figure(dpi = 100)
    plt.title(tytul)
    if (kontur == 1):
        plt.contourf(xx, yy, Z, levels = 4, alpha=0.2)
    plt.scatter(dane["opis_ucz"].iloc[:, atr_x], dane["opis_ucz"].iloc[:, atr_y], c=dane["dec_ucz"], marker = '.')
    plt.scatter(dane["opis_test"].iloc[:, atr_x], dane["opis_test"].iloc[:, atr_y], c=dane["dec_test"], marker = 'x')
```

Wizualizacja granic decyzyjnych ułatwia analizę klasyfikatora.

In [10]:
```
nazwa_pliku = 'dane1.csv'
# wczytanie badanego zbioru danych
df = pd.read_csv(nazwa_pliku)
# podział zbioru danych
d = podziel(df,0.3)
# zdefiniowanie modelu klasyfikatora
model = KNeighborsClassifier(n_neighbors=1)
# wybór atrybutów
ax, ay = 0,1
# granice dycyzyjne
granice(model,d,ax,ay,"klasyfikator 1-NN dla zbioru " + nazwa_pliku)
# weryfikacja
weryfikuj(model,d,[ax,ay])
```

```
Macierz pomyłek dla zbioru uczącego
col_0   0   1   2
row_0
0      20   0   0
1       0  14   0
2       0   0  22
Macierz pomyłek dla zbioru testowego
col_0  0  1   2
row_0
0      4  0   0
1      0  5   0
2      0  0  15
```



Zadanie Czy wyniki dla zbioru `dane2` są lepsze czy gorsze niz dla `dane1` ? Zastanów się, dzlaczego ?

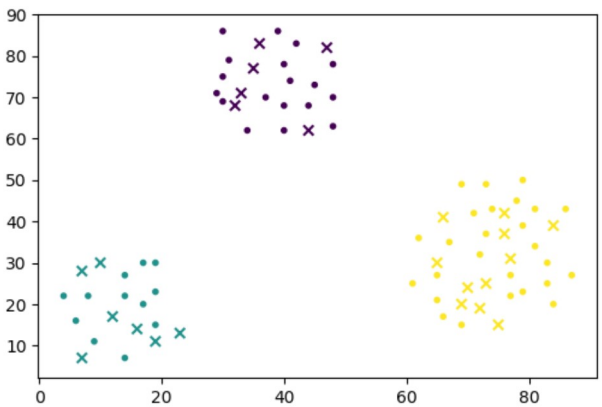
In [11]:
```
# miejsce na kod
```

Zadanie Poeksperymentuj w analogiczny sposób z pozostałymi macierzami `dane` , w tym także ze zbiorem `iris` . Ocen przydatność klasyfikatora dla każdego zbioru danych.

In [12]:
```
# miejsce na kod
```

Procedura `granice` umożliwia także wyświetlenie wykresów punktowych danych z podziałem na zbiór testowy i uczący bez wyświetlania granic decyzyjnych.

In [13]:
```
nazwa_pliku = 'dane1.csv'
df = pd.read_csv(nazwa_pliku)
d = podziel(df,0.3)
granice(0,d,0,1,"",0)
```



Zadanie Wykonaj powyższy kod kilkukrotnie. Czy widzisz jakieś różnice między wynikami kolejnych wywołań ? Dlaczego ?

2. Klasyfikator k-najbliższych sąsiadów (k-NN)

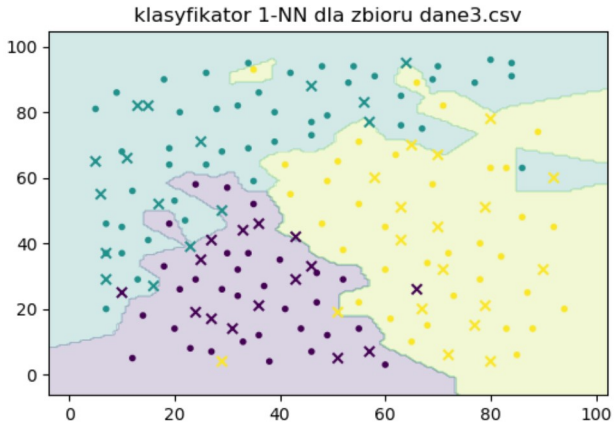
Klasyfikator k-najbliższych sąsiadów jest uoglnieniem klasyfikatora najbliższego sąsiada. W jego przypadku, na podstawie znanych klasy do której należy ustalona liczba k najbliższych sąsiadów określana jest przynależność klasyfikowanego obiektu do klasy. Klasa wynikowa odpowiada klasie dominującej w zbiorze k-najbliższych sąsiadów.

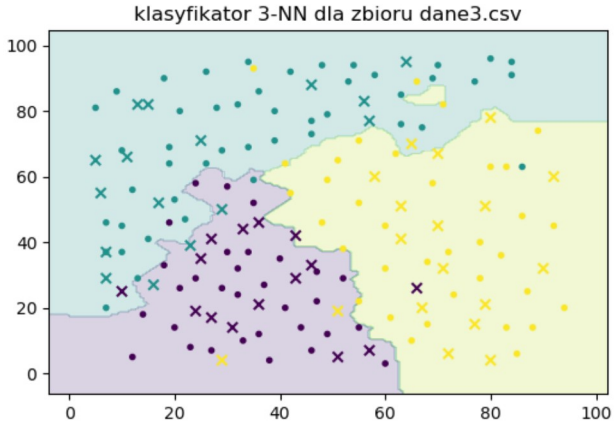
Zadanie Jak zwiększenie liczby sąsiadów wpłynie na wynik klasyfikacji zbioru `dane2` ? Dlaczego ?

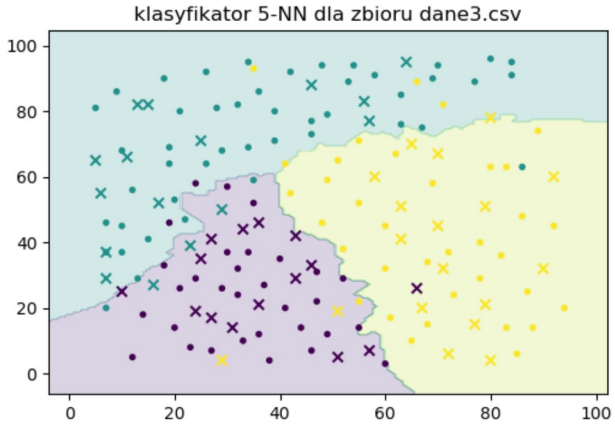
In [14]:
```
# miejsce na kod
```

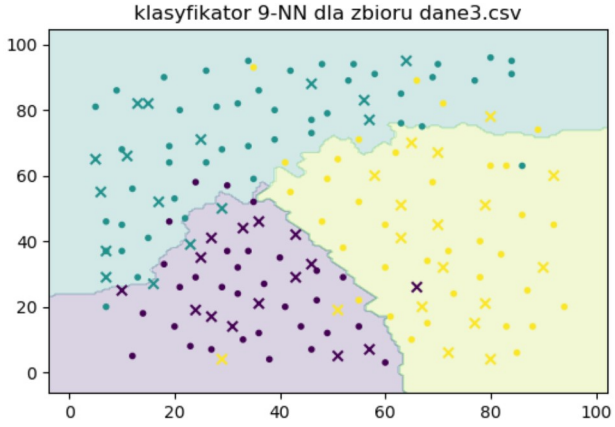
Analiza wpływu liczby k na wynik klasyfikacji na przykładzie zbioru `dane3` .

In [15]:
```
nazwa_pliku = 'dane3.csv'
df = pd.read_csv(nazwa_pliku)
d = podziel(df,0.3)
for k in [1,3,5,9]:
    model_knn = KNeighborsClassifier(n_neighbors=k)
    granice(model_knn,d,0,1,"klasyfikator " + str(k)+ "-NN dla zbioru " + nazwa_pliku)
```









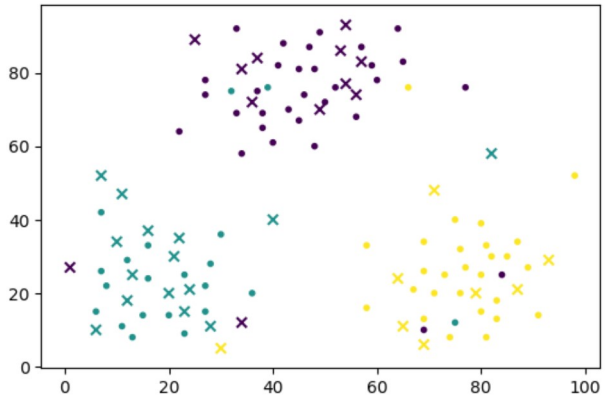
Zadanie Poeksperymentuj w analogiczny sposób z pozostałymi macierzami danych. Ocen przydatnosc klasyfikatora dla kazdego zbioru danych.

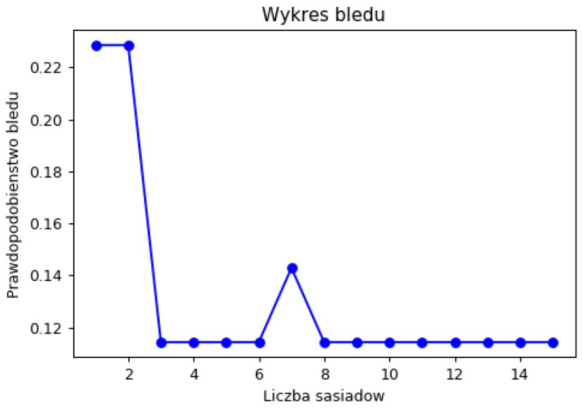
In [16]:
```
# miejsce na kod
```

W celu doboru właściwej (na ogół nieparzystej) ilości sąsiadów należy wykonać analizę błędu klasyfikacji dla różnych wartości sąsiadów.

In [17]:
```
nazwa_pliku = 'dane2.csv'
df = pd.read_csv(nazwa_pliku)
d = podziel(df,0.3)
granice(model,d,0,1,"",0)
kvals = range(1, 16)
rss_all = np.zeros(15)
for k in kvals:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(d['opis_ucz'],d['dec_ucz'])
    predictions = model.predict(d['opis_test'])
    rss_all[k-1] = 1-model.score(d['opis_test'],d['dec_test'])
plt.figure(dpi=90)
plt.plot(kvals, rss_all,'bo-')
plt.title('Wykres bledu')
plt.xlabel('Liczba sasiadow')
plt.ylabel('Prawdopodobienstwo bledu')
```

Out[17]:
```
Text(0, 0.5, 'Prawdopodobienstwo bledu')
```





Zadanie Jak na podstawie powyższego wykresu określić optymalną liczbę sąsiadów ? Jak przebiega ten wykres dla różnych zbiorów danych (porównaj m.in. `dane2` i `dane3`) ? Skąd się biorą różnice w jego przebiegu ?

In [18]:
```
# miejsce na kod
```

Zadanie Dobierz optymalną liczbę k dla innych zbiorów

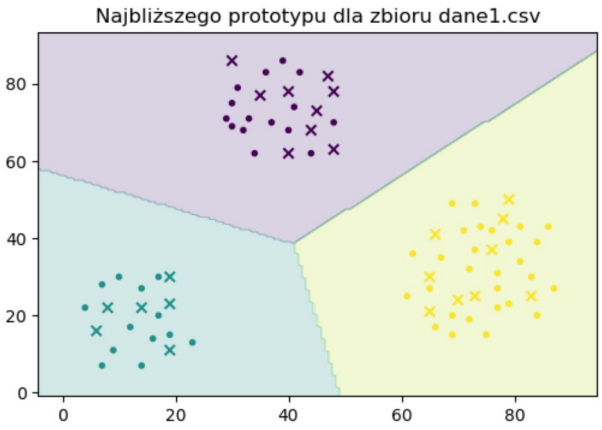
In [19]:
```
# miejsce na kod
```

3. Metoda najbliższych prototypów

Wadą wszystkich opisanych do tej pory klasyfikatorów najbliższych sąsiadów jest konieczność korzystania podczas procesu klasyfikacji z całego zbioru uczącego. Dla duzej liczby obiektów zbioru uczącego i dużej liczby cech proces klasyfikacji staje się czasochłonny, czesto zachodzi ponadto koniecznosc przechowywania całego tego zbioru w pamieci. Rozwiazaniem tego problemu jest redukcja zbioru uczacego do zbioru składajacego sie z obiektów reprezentatywnych dla kazdej z rozpatrywanych klas – prototypów klas. Wówczas, zamiast rozpatrywania całego zbioru uczacego, rozpatrywany jest jedynie zbiór prototypów. Typowym rozwiazaniem jest wybór po jednym prototypie na klasę, choc stosowane jest takze rozwiazanie polegajace na wyborze wiekszej liczby prototypów kazdej klasy. Prototyp jest charakteryzowany przez wartosci jego atrybutów. Wartosci te sa wyznaczane najczesciej jako miary tendencji centralnej wyznaczane dla wszystkich obiektów w danej klasie. Najczesciej stosowana miara jest tu średnia arytmetyczna. Podzbiory danych odpowiadające poszczególnym klasom są w tym przypadku zastępowane przez centroidy klas.

In [20]:
```
nazwa_pliku = 'dane1.csv'
# wczytanie badanego zbioru danych
df = pd.read_csv(nazwa_pliku)
# podział zbioru danych
d = podziel(df,0.3)
# zdefiniowanie modelu klasyfikatora
model = NearestCentroid()
# granice dycyzyjne
granice(model,d,0,1,"Najbliższego prototypu dla zbioru " + nazwa_pliku)
# weryfikacja
weryfikuj(model,d,[0,1])
```

```
Macierz pomyłek dla zbioru uczącego
col_0   0   1   2
row_0
0      15   0   0
1       0  13   0
2       0   0  28
Macierz pomyłek dla zbioru testowego
col_0  0  1  2
row_0
0      9  0  0
1      0  6  0
2      0  0  9
```



Zadanie Poeksperymentuj w analogiczny sposób z pozostałymi macierzami `dane` , w tym także ze zbiorem `iris` . Ocen przydatność klasyfikatora dla każdego zbioru danych.

In [21]:
```
# miejsce na kod
```

Pytania

1. Czy w przypadku klasyfikatora najblizszych prototypów zachodzi faza uczenia ? Jesli tak, to na czym polega ?
2. Jak zmierzyc stopien rozproszenia wartosci atrybutu wokół wartosci sredniej ?

Zadanie Wykonaj klasyfikacje klasyfikatorem k-NN, dla różnych wartości k oraz najblizszych prototypów. Wykonaj weryfikacje wyników klasyfikacji na zbiorze uczacym i testowym dla kazdego z trzech klasyfikatorów. Ocen wyniki. Zastanów sie jak zinterpretowac błedne wskazania klasyfikatora dla obiektów ze zbioru uczacego ? Dlaczego moze sie zdarzyc, ze nie wszystkie obiekty zbioru uczacego sa poprawnie klasyfikowane? Na wykresie punktowym wskaz takie obiekty. Czy – w tym konkretnym przypadku – niepoprawna klasyfikacja jest efektem pozytywnym czy negatywnym ? O jakiej własnosci klasyfikatora ona swiadczy ?

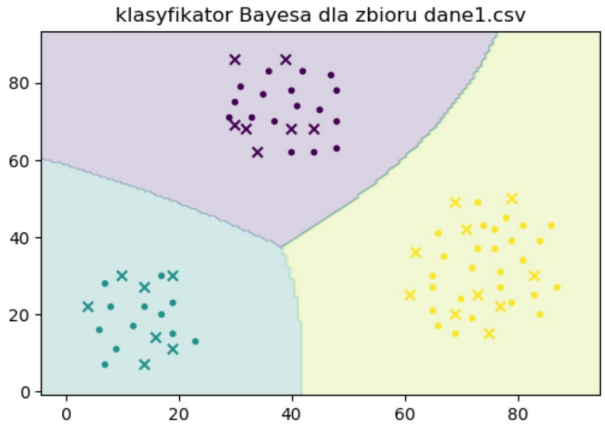
In [22]:
```
# miejsce na kod
```

4. Naiwny klasyfikator Bayesa

Podstawa klasyfikacji Bayesowskiej jest twierdzenie Bayesa, które dotyczy prawdopodobienstw warunkowych. W przypadku klasyfikacji, zdarzenia losowe, które sa brane pod uwage przy wyznaczaniu prawdopodobienstw dotycza dwóch faktów zwiazanych z rozpoznawanymi obiektami: posiadania przez obiekt konkretnego zbioru wartosci atrybutów opisujacych zapisanego zwykle w formie wektora wartosci atrybutów oraz przynaleznosci tego obiektu do poszczególnych klas. Przynaleznosc obiektu do poszczególnych klas jest okreslana przy pomocy funkcji dyskryminacyjnych. i-ta funkcja dyskryminacyjna dla obiektu o wektorze atrybutów opisujacych jest w tym przypadku tozsama prawdopodobienstwu warunkowemu przynależnosci obiektu do i-tej klasy pod warunkiem posiadania przez obiekt wektora atrybutów opisujacych. Wygodnym założeniem jest brak zaleznosci miedzy poszczególnymi atrybutami opisujacymi. Dzieki niemu mozna przyjać, ze zdarzenia losowe polegajace na posiadaniu przez obiekt konkretnych wartosci poszczególnych atrybutów sa od siebie niezalezne. Klasyfikatory spełniajace to założenie noszą nazwę naiwnych klasyfikatorów Bayesowskich. W przypadku atrybutów ilościowych niezbędne prawdopodobieństwa szacuje sie z wykorzystaniem typowych rozkładów zmiennych losowych.

In [23]:
```
nazwa_pliku = 'dane1.csv'
# wczytanie badanego zbioru danych
df = pd.read_csv(nazwa_pliku)
# podział zbioru danych
d = podziel(df,0.3)
# zdefiniowanie modelu klasyfikatora
model = GaussianNB()
# granice dycyzyjne
granice(model,d,0,1,"klasyfikator Bayesa dla zbioru " + nazwa_pliku)
# weryfikacja
weryfikuj(model,d,[0,1])
```

```
Macierz pomyłek dla zbioru uczącego
col_0   0   1   2
row_0
0      17   0   0
1       0  12   0
2       0   0  27
Macierz pomyłek dla zbioru testowego
col_0  0  1   2
row_0
0      7  0   0
1      0  7   0
2      0  0  10
```



Zadanie Wykonaj klasyfikacje klasyfikatorem Bayesa wszystkich zbiorów, dla których wyniki klasyfikacji najblizszego prototypu były niezadowalajace. Czy zastosowanie klasyfikatora Bayesa je polepsza ? Dlaczego ?

In [24]:
```
# kod zadania
```

Zadanie Znajdz zbiory danych, w przypadku których wyniki klasyfikacji Bayesowskiej są gorsze niz k-NN. Zastanów sie dlaczego tak sie dzieje. Dla jakich dystrybucji obiektów w przestrzeni atrybutów (położenia zbiorów punktów na wykresie punktowym) klasyfikator Bayesa daje dobre wyniki, a dla jakich gorsze ? Dlaczego ?

In [25]:
```
# kod zadania
```

Zadanie Poeksperymentuj z pozostałymi macierzami danych. Oceń przydatnosc klasyfikatora dla każdego zbioru danych.

In [26]:
```
# kod zadania
```

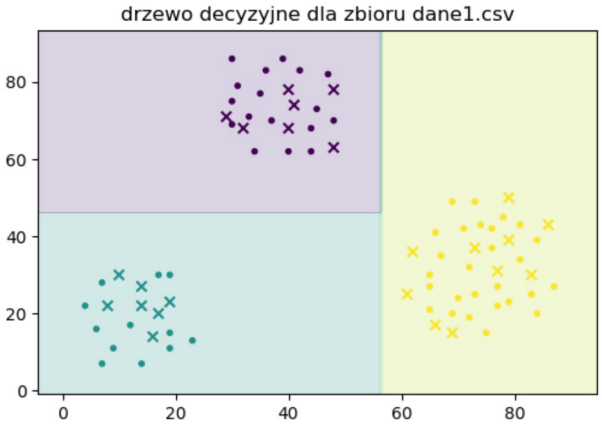
5. Drzewa decyzyjne

Drzewa decyzyjne sa struktura grafowa przedstawiajaca zaleznosci miedzy atrybutami obiektów. Drzewo decyzyjne opisuje w formie grafu zaleznosc wartosci atrybutu decyzyjnego od wartosci atrybutów opisujacych. Dzieki hierarchicznej reprezentacji tych zaleznosci drzewo nie tylko jest klasyfikatorem, ale takze umozliwia analize istotnosci poszczególnych atrybutów dla konkretnego procesu klasyfikacji. Czestym problemem przy klasyfikacji za pomoca drzew jest przetrenowanie. Nie zawsze stuprocentowo skuteczna klasyfikacja zbioru uczacego przekłada sie na dobre wyniki klasyfikacji zbioru testowego, poniewaz drzewo, w którym jest zbyt wiele zbyt szczegółowych testów traci zdolnosc generalizacji. Problem przetrenowania moze byc rozwiazany na dwa sposoby, poprzez:

- wstrzymanie budowy drzewa, zanim osiagnie maksymalne rozmiary (ograniczanie w trakcie rozrostu), lub
- przycinanie drzewa po jego wyznaczeniu (drzewa maksymalnego).

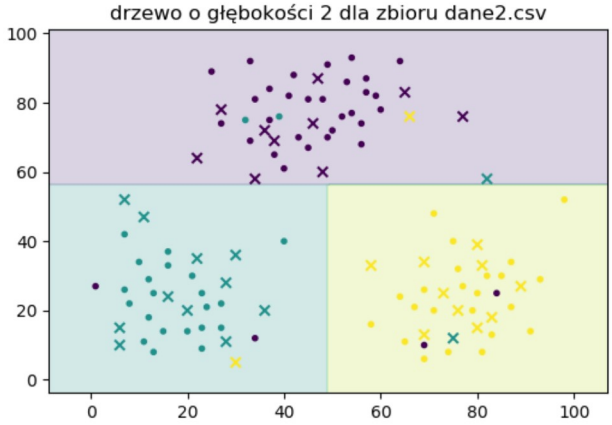
In [27]:
```
nazwa_pliku = 'dane1.csv'
# wczytanie badanego zbioru danych
df = pd.read_csv(nazwa_pliku)
# podział zbioru danych
d = podziel(df,0.3)
# zdefiniowanie modelu klasyfikatora
model = tree.DecisionTreeClassifier(max_depth=4)
# granice dycyzyjne
granice(model,d,0,1,"drzewo decyzyjne dla zbioru " + nazwa_pliku)
# weryfikacja
weryfikuj(model,d,[0,1])
```

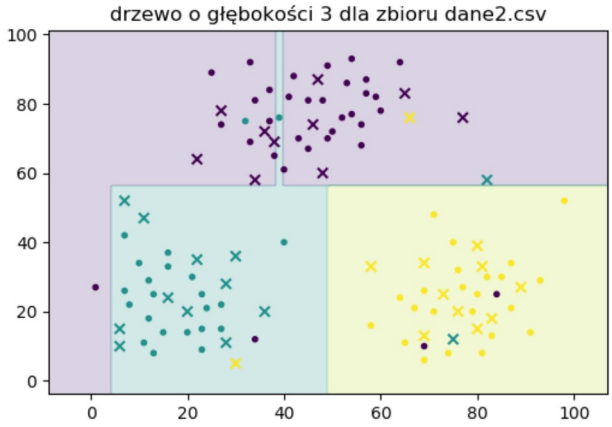
```
Macierz pomyłek dla zbioru uczącego
col_0   0   1   2
row_0
0      17   0   0
1       0  12   0
2       0   0  27
Macierz pomyłek dla zbioru testowego
col_0  0  1   2
row_0
0      7  0   0
1      0  7   0
2      0  0  10
```

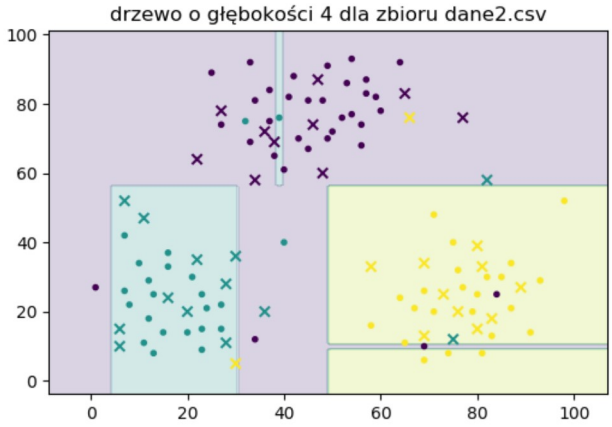


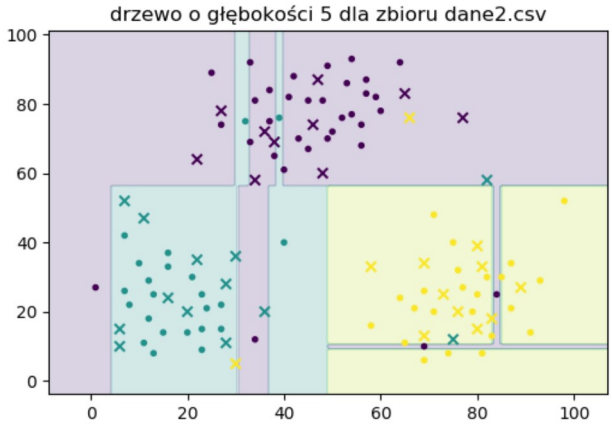
Drzewa decyzyjne o różnych głębokościach.

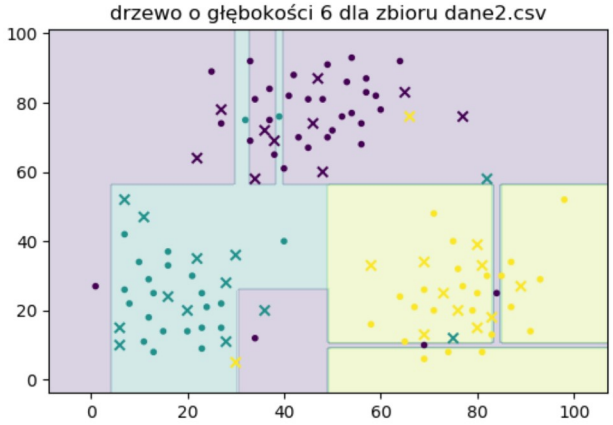
In [28]:
```
nazwa_pliku = 'dane2.csv'
# wczytanie badanego zbioru danych
df = pd.read_csv(nazwa_pliku)
# podział zbioru danych
d = podziel(df,0.3)
for g in [2,3,4,5,6]:
    drzewo = tree.DecisionTreeClassifier(max_depth=g)
    tekst = "drzewo o głębokości " + str(g) + " dla zbioru " + nazwa_pliku
    granice(drzewo ,d,0,1,tekst)
```


drzewo o głębokości 2 dla zbioru dane2.csv
0
20
40
60
80
100


drzewo o głębokości 3 dla zbioru dane2.csv
0
20
40
60
80
100


drzewo o głębokości 4 dla zbioru dane2.csv
0
20
40
60
80
100


drzewo o głębokości 5 dla zbioru dane2.csv
0
20
40
60
80
100



Zadanie Utwórz drzewa decyzyjne dla innych zbiorów danych. Ocen przydatnosc klasyfikatora dla kazdego zbioru danych. Dla jakich danych konstrukcja drzew jest prostsza, a dla jakich – bardziej skomplikowana ? Dlaczego ? Jak wielkosc drzewa wpływa na skutecznosc klasyfikacji ?

In [29]:
```
# kod zadania
```

Zadanie Poeksperymentuj z roznymi klasyfikatorami na zbiorze `iris` o czterech atrybutach decyzyjnych, wybierajac tylko dwa z nich. Zwróc uwage na to jak wybór dwóch z czterech atrybutów wpływa na wynik klasyfikacji. Wskaz najlepsza i najgorsza pare atrybutów z punktu widzenia poprawnosci klasyfikacji. Czy jestes w stanie wskazac na macierzy wykresów punktowych dla tego zbioru danych, cechy rozkładu punktów, które potwierdzaja ten wybór ? Porównaj wynik klasyfikacji dla najlepszej pary atrybutów z klasyfikacja z wykorzystaniem wszystkich czterech atrybutów.

In [30]:
```
# kod zadania
```

Zadanie Przeanalizuj zbiór `dane20.csv` . Wykonaj stosowną wyzializację danych. Określ, które atrybuty mają wpływ na przynależność obiektu do klasy, a które - nie. Przetestuj omówione klasyfikatory wykorzystując w klasyfikacji:

1. wszystkie atrybuty
2. atrybuty wpływające na klasę obiektu
3. atrybuty niewpływające na klasę
4. dowolną mieszankę obu rodzajów atrybutów

In [31]:
```
# kod zadania
```