

---

# UMB W08: klasyfikacja

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

import umb_tools as umb
```

```
# konfiguracja
plt.rcParams["figure.figsize"] = [5, 4]
pd.set_option("display.float_format", lambda x: "%.4f" % x)
```

---

## 1. Zbiór danych

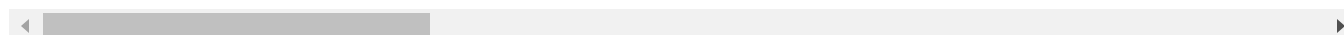
---

### Wczytanie i normalizacja danych

```
# odczyt pliku TSV (zwracane są: zbiór danych w postaci DataFrame biblioteki Pandas oraz lista kolumn)
(df, c_names) = umb.read_data("data/BreastCancer.txt")
df
```

#BreastCancer	labels	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cor
842302	1	17.9900	10.3800	122.8000	1001.0000	0.1184	
874858	1	14.2200	23.1200	94.3700	609.9000	0.1075	
875263	1	12.3400	26.8600	81.1500	477.4000	0.1034	
87556202	1	14.8600	23.2100	100.4000	671.4000	0.1044	
875938	1	13.7700	22.2900	90.6300	588.9000	0.1200	
...	...	...	...	...	...	...	...
8910720	2	10.7100	20.3900	69.5000	344.9000	0.1082	
8910506	2	12.8700	16.2100	82.3800	512.2000	0.0943	
8910499	2	13.5900	21.8400	87.1600	561.0000	0.0796	
8912055	2	11.7400	14.0200	74.2400	427.3000	0.0781	
92751	2	7.7600	24.5400	47.9200	181.0000	0.0526	

569 rows × 31 columns



```
# pobranie etykiet klas (0 lub 1)
labels = df.iloc[:, 0].to_numpy() - 1

# pobranie macierzy danych
data = df.iloc[:, 1:].to_numpy()

# normalizacja
data = StandardScaler().fit_transform(data)
```

## Informacje o klasach

```
# pobranie indeksów próbek z klas
(c_labels, c_index) = umb.class_info(labels)

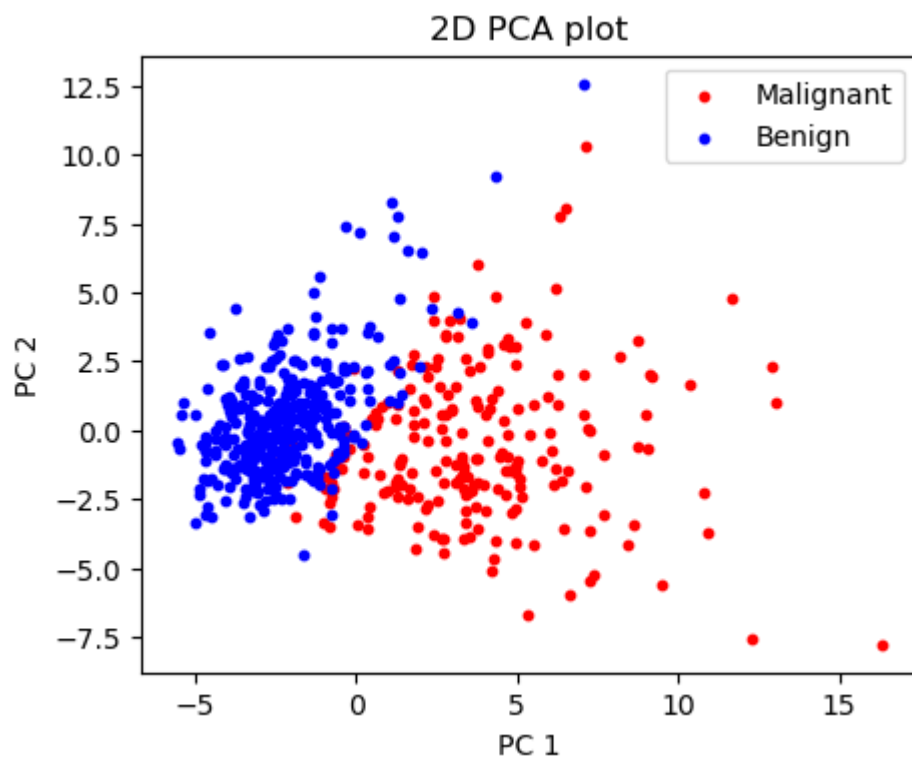
# liczba klas
c_n = len(c_labels)

# informacje o klasach
print(f"\nClasses: {c_n}")
for i in range(c_n):
    print(f"  name = {c_names[i]}, label = {c_labels[i]}, samples = {len(c_index[i])}")
```

```
Classes: 2
  name = Malignant, label = 0, samples = 212
  name = Benign, label = 1, samples = 357
```

## Analiza składowych głównych

```
# wykres PCA  
umb.pca_plot(data, labels, c_names)
```



---

## 2. Przykładowe klasyfikatory

---

### Gaussowskie klasyfikatory Bayesa (QDA i LDA)

[https://scikit-](https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html)

[learn.org/stable/modules/generated/sklearn.discriminant\\_analysis.QuadraticDiscriminantAnalysis.html](https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html)

[https://scikit-](https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html)

[learn.org/stable/modules/generated/sklearn.discriminant\\_analysis.LinearDiscriminantAnalysis.html](https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html)

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis, LinearDiscriminantAnalysis
```

```
# utworzenie obiektu klasyfikatora QDA
c_model = QuadraticDiscriminantAnalysis()

# budowanie klasyfikatora na podstawie danych i etykiet klas
c_model.fit(data, labels)

# uzyskanie wyniku klasyfikacji (przewidywanych etykiet klas)
y = c_model.predict(data)
print(f"\nPredicted class labels:\n {y}")

# uzyskanie prawdopodobieństw a posteriori przynależności do klas
p = c_model.predict_proba(data)
print(f"\nPredicted probabilities:\n {p}")

# błęd
e = sum(labels != y) / len(labels)
print(f"\nerror = {e:.4f}")

# wykres PCA
umb.classif_plot(data, c_model, labels, c_names)
```

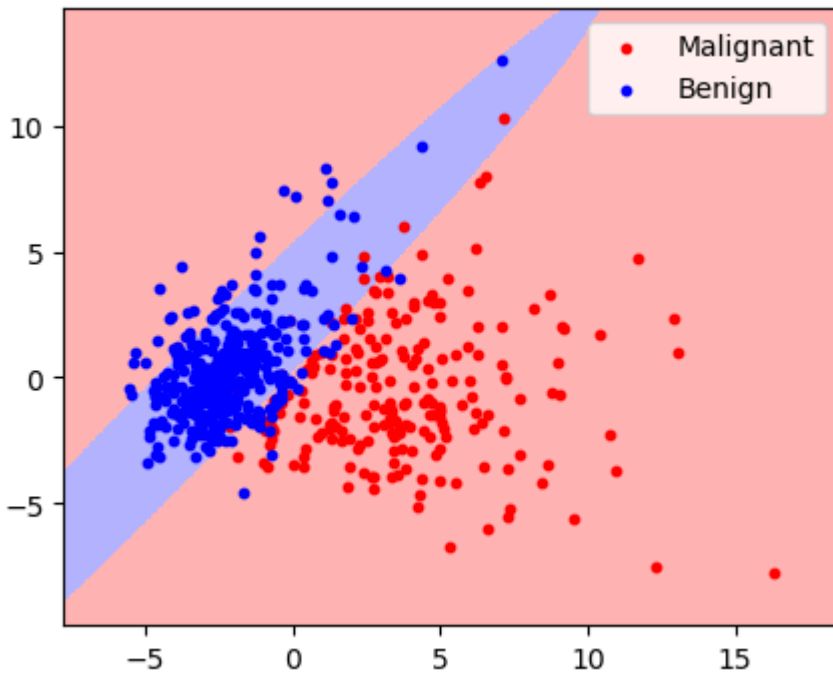
Predicted class labels:

[illegible]

Predicted probabilities:

```
[1.00000000e+00 0.00000000e+00]
[1.00000000e+00 6.34142338e-67]
[1.00000000e+00 2.89200146e-14]
...
[5.83081679e-09 9.99999994e-01]
[9.96708246e-13 1.00000000e+00]
[1.43775343e-48 1.00000000e+00]
```

error = 0.0264



```
# utworzenie obiektu klasyfikatora LDA
c_model = LinearDiscriminantAnalysis()

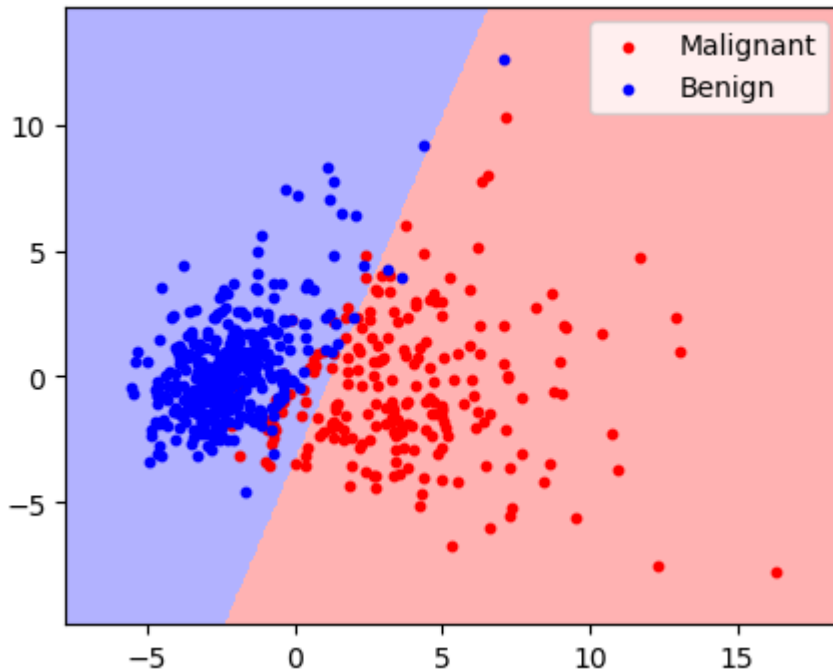
# budowanie klasyfikatora na podstawie danych i etykiet klas
c_model.fit(data, labels)
```

```
# uzyskanie wyniku klasyfikacji (przewidywanych etykiet klas)
y = c_model.predict(data)

# błęd
e = sum(labels != y) / len(labels)
print(f"\nerror = {e:.4f}")

# wykres PCA
umb.classif_plot(data, c_model, labels, c_names)
```

error = 0.0351



## Gaussowskie naiwne klasyfikatory Bayesa (DQDA i DLDA)

[https://scikit-learn.org/stable/modules/generated/sklearn.naive\\_bayes.GaussianNB.html](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html)

```
from sklearn.naive_bayes import GaussianNB
```

```
# utworzenie obiektu klasyfikatora
c_model = GaussianNB()

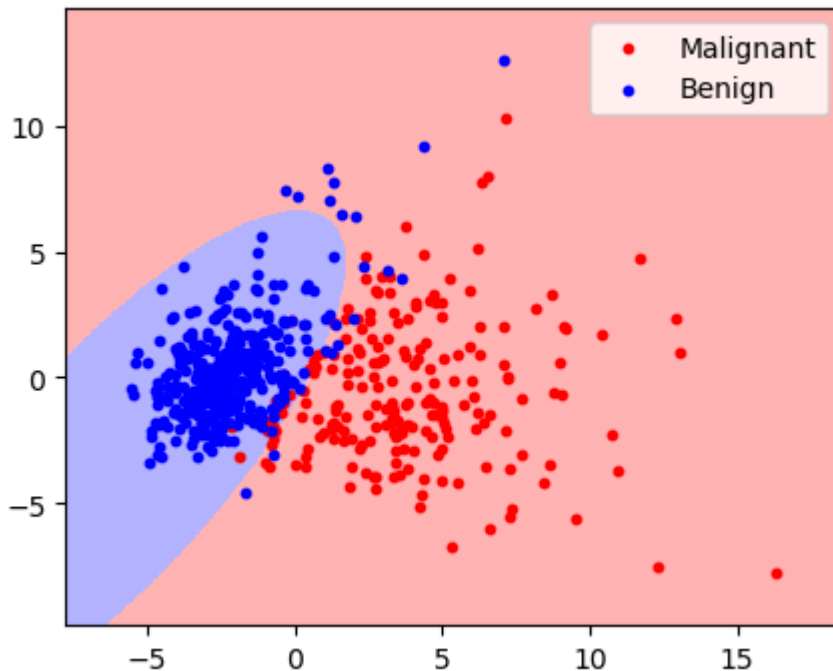
# trening klasyfikatora na podstawie danych i etykiet klas
c_model.fit(data, labels)

# uzyskanie wyniku klasyfikacji (przewidywanych etykiet klas)
y = c_model.predict(data)

# błęd
e = sum(labels != y) / len(labels)
print(f"\nerror = {e:.4f}")
```

```
# wykres PCA
umb.classif_plot(data, c_model, labels, c_names)
```

error = 0.0598



```
# utworzenie obiektu klasyfikatora
c_model = GaussianNB()

# trening klasyfikatora na podstawie danych i etykiet klas
c_model.fit(data, labels)

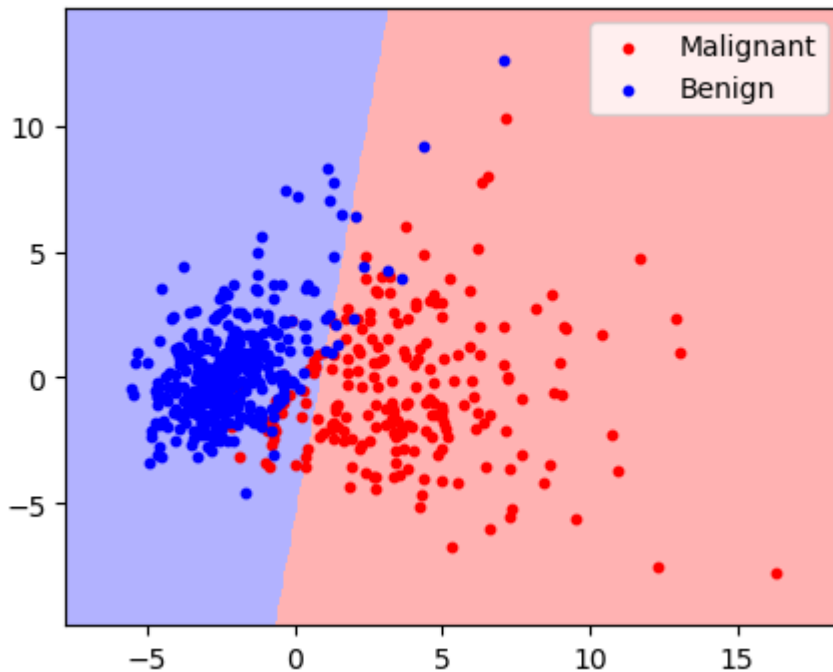
# narzucenie jednakowych wariancji atrybutów w każdej z klas
# (powstanie klasyfikator DLDA zamiast DQDA)
c_model.var_ = np.tile(np.var(data, axis=0), (c_n, 1))

# uzyskanie wyniku klasyfikacji (etykiet klas)
y = c_model.predict(data)

# błęd
e = sum(labels != y) / len(labels)
print(f"\nerror = {e:.4f}")

# wykres PCA
umb.classif_plot(data, c_model, labels, c_names)
```

error = 0.0703



## Klasyfikator K-NN

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

```
from sklearn.neighbors import KNeighborsClassifier
```

```
K = 3
```

```
# utworzenie obiektu klasyfikatora
```

```
c_model = KNeighborsClassifier(n_neighbors=K)
```

```
# trening klasyfikatora na podstawie danych i etykiet klas
```

```
c_model.fit(data, labels)
```

```
# uzyskanie wyniku klasyfikacji (przewidywanych etykiet klas)
```

```
y = c_model.predict(data)
```

```
# błęd
```

```
e = sum(labels != y) / len(labels)
```

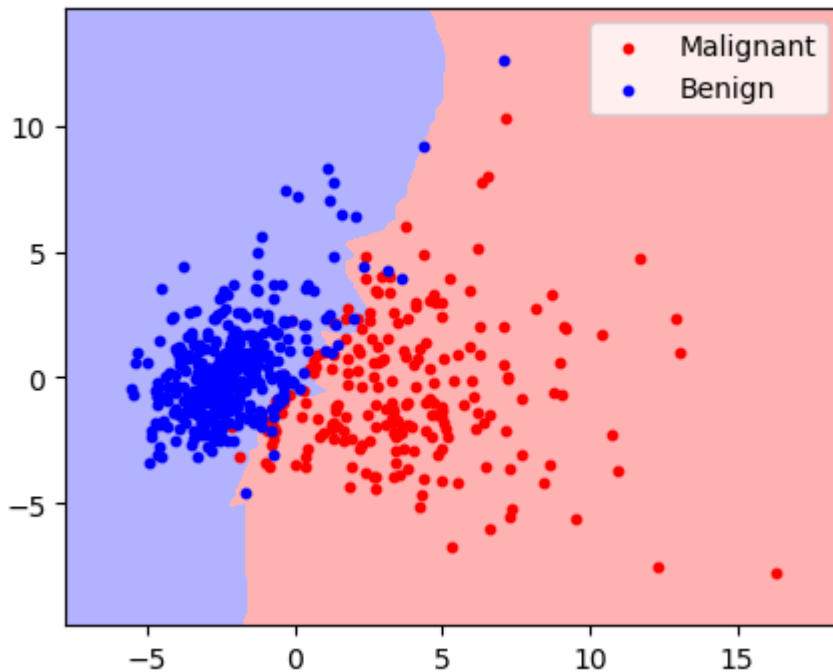
```
print(f"\nerror = {e:.4f}")
```

```
# wykres PCA
```

```
umb.classif_plot(data, c_model, labels, c_names)
```

```
error = 0.0193
```





---

## Klasyfikator SVM

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

```
from sklearn.svm import SVC
```

```
# utworzenie obiektu klasyfikatora (Liniowy SVM)
c_model = SVC(kernel="linear", C=1.0)

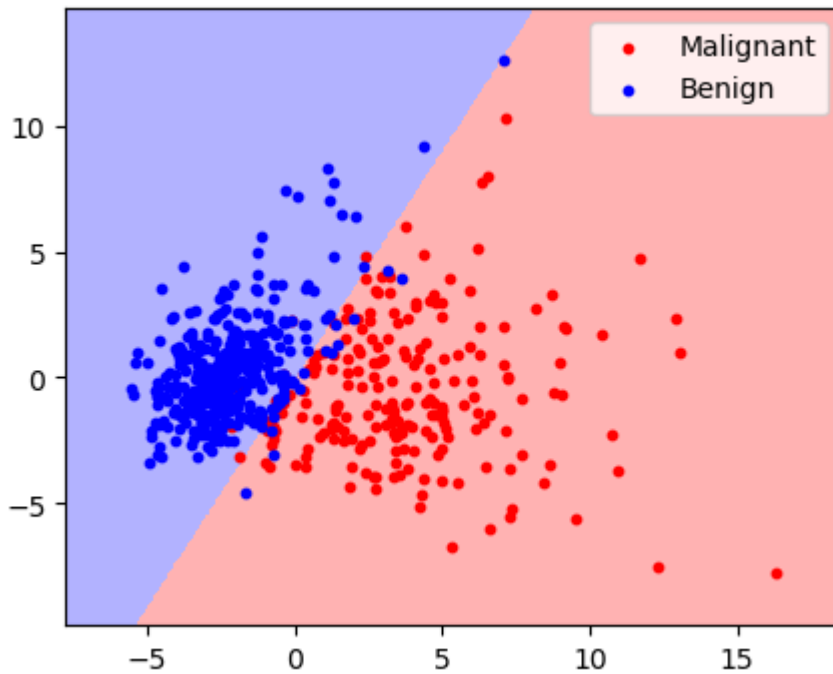
# trening klasyfikatora na podstawie danych i etykiet klas
c_model.fit(data, labels)

# uzyskanie wyniku klasyfikacji (przewidywanych etykiet klas)
y = c_model.predict(data)

# błąd
e = sum(labels != y) / len(labels)
print(f"\nerror = {e:.4f}")

# wykres PCA
umb.classif_plot(data, c_model, labels, c_names)
```

error = 0.0123



```
# utworzenie obiektu klasyfikatora (SVM z jądrem gaussowskim)
c_model = SVC(kernel="rbf", C=1.0, gamma=0.1)

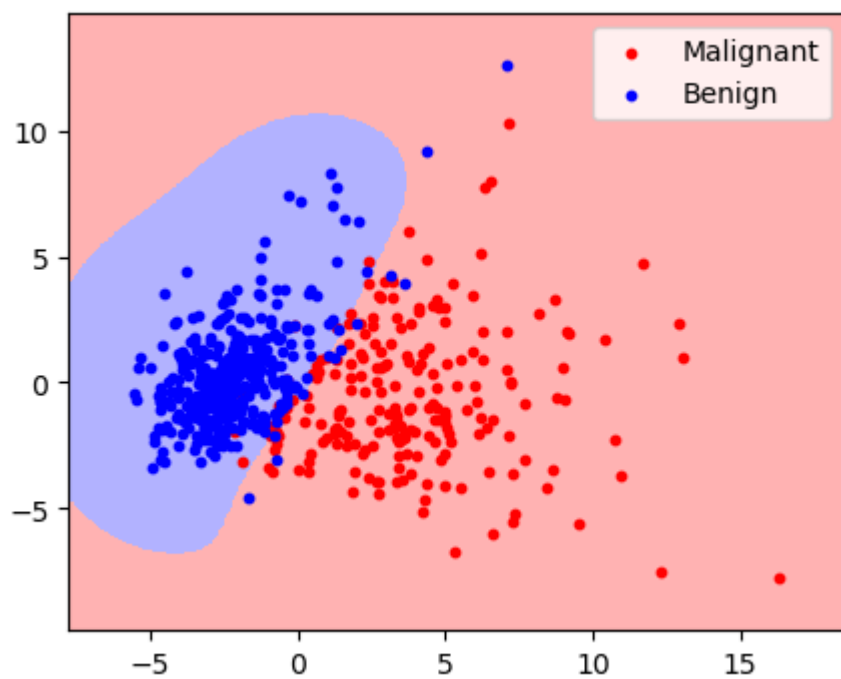
# trening klasyfikatora na podstawie danych i etykiet klas
c_model.fit(data, labels)

# uzyskanie wyniku klasyfikacji (przewidyanych etykiet klas)
y = c_model.predict(data)

# błąd
e = sum(labels != y) / len(labels)
print(f"\nerror = {e:.4f}")

# wykres PCA
umb.classif_plot(data, c_model, labels, c_names)
```

error = 0.0088



---

### 3. Przykładowa implementacja klasyfikatora

---

#### Klasyfikator minimalnej odległości Euklidesa

```
import scipy.spatial.distance as dist

class MinEuclidean:

    def __init__(self):
        # środki klas (wektory wartości średnich)
        self.theta_ = None

        # liczba atrybutów
        self.n_features_in_ = 0

        # liczba klas
        self.class_count_ = 0

    def fit(self, X, y):
        # pobranie informacji o klasach na podstawie etykiet
        c_labels, c_index = umb.class_info(y)

        # liczba klas i atrybutów
        self.class_count_ = len(c_labels)
        self.n_features_in_ = X.shape[1]

        # wyznaczenie wektorów wartości średnich w klasach
        self.theta_ = np.zeros([self.class_count_, self.n_features_in_])
        for i in range(self.class_count_):
            self.theta_[i, :] = np.mean(X[c_index[i], :], axis=0)

        return self

    def predict(self, X):
        y = np.zeros(X.shape[0], dtype="int")

        for i in range(X.shape[0]):
            x = X[i, :].reshape(1, X.shape[1])

            # liczenie odległości wektorów do środków klas
            d = dist.cdist(x, self.theta_)

            # znalezienie klasy o minimalnej odległości
            y[i] = np.argmin(d)

        return y
```

```
# utworzenie obiektu klasyfikatora
c_model = MinEuclidean()

# trening klasyfikatora na podstawie danych i etykiet klas
```

```

c_model.fit(data, labels)

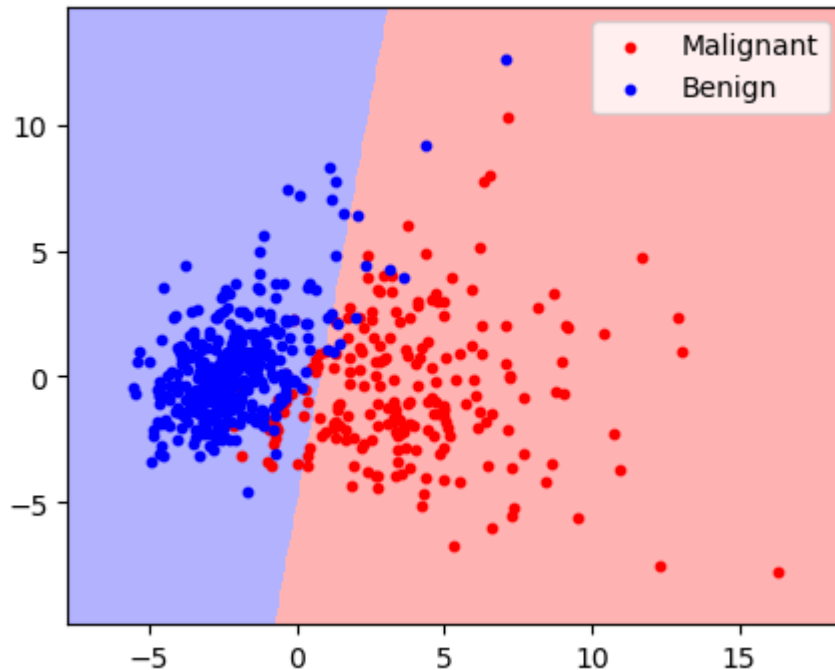
# uzyskanie wyniku klasyfikacji (przewidywanych etykiet klas)
y = c_model.predict(data)

# błęd
e = sum(labels != y) / len(labels)
print(f"\nerror = {e:.4f}")

# wykres PCA
umb.classif_plot(data, c_model, labels, c_names)

```

error = 0.0685



## Porównanie działania z implementacją w *scikit-learn*

```

# utworzenie obiektu klasyfikatora
c_model = GaussianNB()

# trening klasyfikatora na podstawie danych i etykiet klas
c_model.fit(data, labels)

# wariancje atrybutów równe 1 i równoliczne klas: przy takich założeniach
# wynik będzie taki sam, jak dla klasyfikatora min. odległości
c_n = len(np.unique(labels))
c_model.var_ = np.ones([c_n, data.shape[1]])
c_model.class_prior_ = np.ones(c_n) / c_n

# uzyskanie wyniku klasyfikacji (przewidywanych etykiet klas)
y = c_model.predict(data)

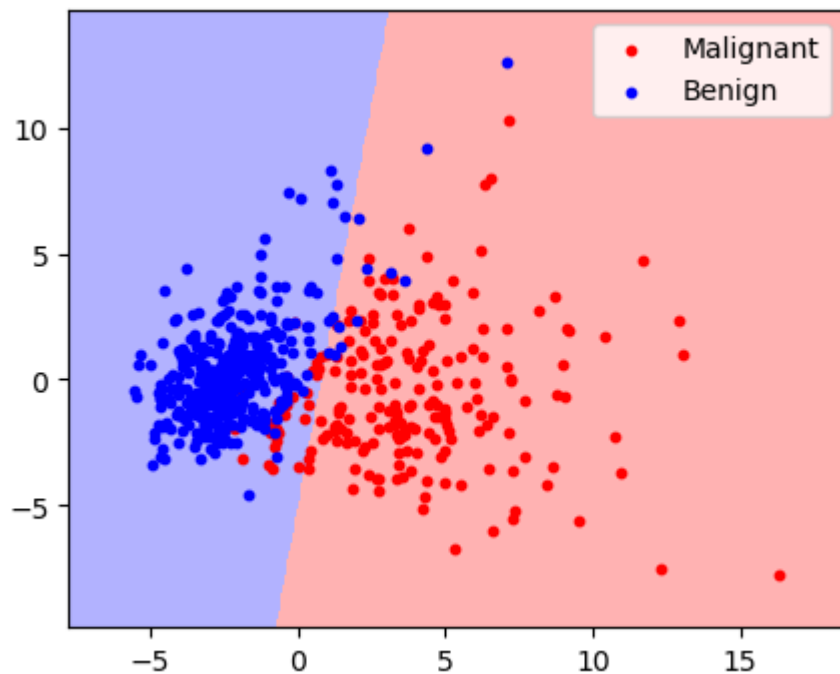
# błęd

```

```
e = sum(labels != y) / len(labels)
print(f"\nerror = {e:.4f}")

# wykres PCA
umb.classif_plot(data, c_model, labels, c_names)
```

error = 0.0685



---

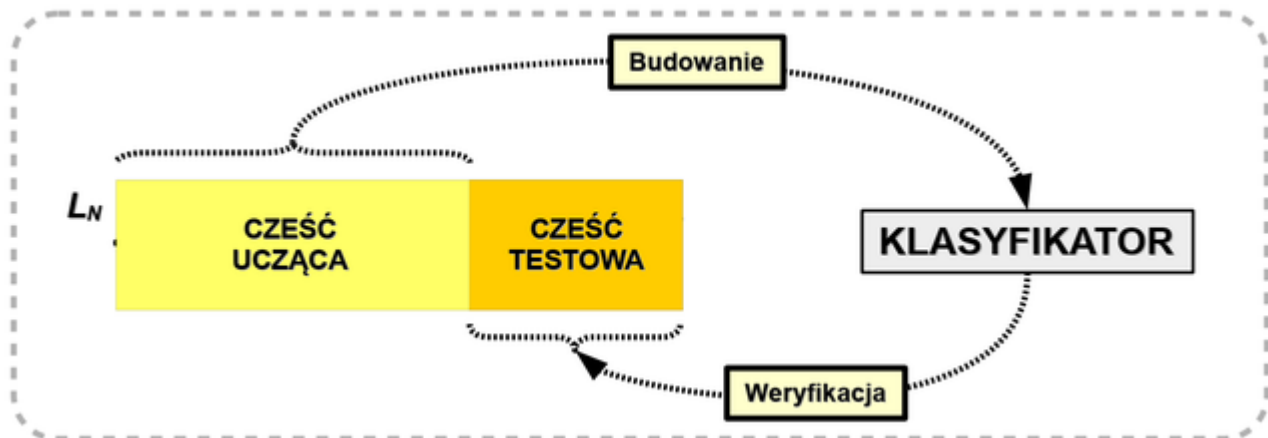
## 4. Weryfikacja skuteczności klasyfikatora

[https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html)

[https://scikit-learn.org/stable/modules/classes.html#module-sklearn.model\\_selection](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection)

---

### Podział danych na zbiór uczący i treningowy (*hold-out*)



```
from sklearn.model_selection import train_test_split
```

```
# podział zbioru danych na dwie części (70% próbek uczących, 30% testowych)
data_train, data_test, labels_train, labels_test = train_test_split(data, labels, test_size=0.3)

# trening klasyfikatora 3NN na danych uczących
c_model = KNeighborsClassifier(n_neighbors=3)
c_model.fit(data_train, labels_train)

# wynik klasyfikacji dla zbioru testowego
y = c_model.predict(data_test)

e = sum(labels_test != y) / len(labels_test)
print(f"\ntest set error = {e:.4f}")

# wynik klasyfikacji dla zbioru uczącego
y = c_model.predict(data_train)

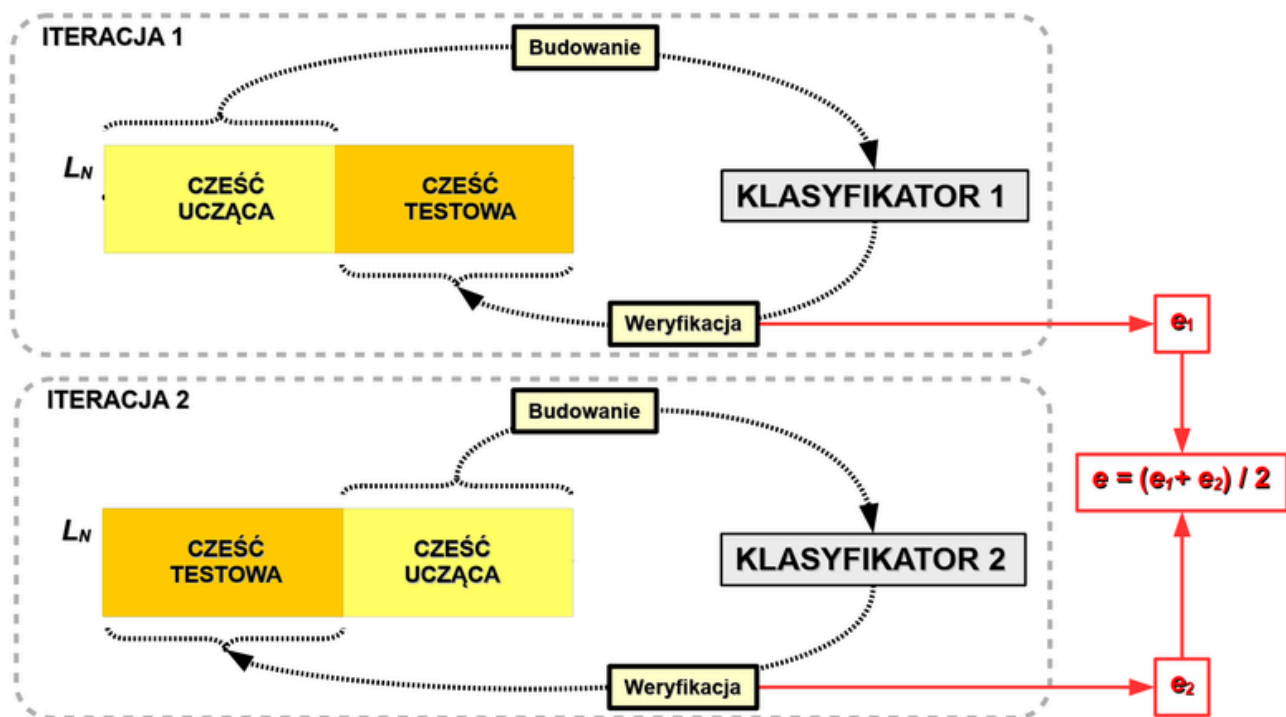
e = sum(labels_train != y) / len(labels_train)
print(f"\ntrain set error = {e:.4f}")
```

```
test set error = 0.0292
```

```
train set error = 0.0176
```

---

### K-krotna walidacja krzyżowa (*K-Folds cross-validation*)



```
from sklearn.model_selection import cross_val_score
```

```
# 5-krotna walidacja krzyżowa klasyfikatora 3NN
```

```
c_model = KNeighborsClassifier(n_neighbors=3)
```

```
acc = cross_val_score(c_model, data, labels, cv=5)
```

```
print(f"\nCV accuracy = {acc}")
```

```
print(f"\nmean accuracy {acc.mean():.4f} with standard deviation of {acc.std():.4f}")
```

```
CV accuracy = [0.97368421 0.96491228 0.98245614 0.95614035 0.98230088]
```

```
mean accuracy 0.9719 with standard deviation of 0.0102
```



---

## 3.4 Miary jakości klasyfikatora

[https://scikit-learn.org/stable/modules/model\\_evaluation.html#classification-metrics](https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics)

```
# podział zbioru danych na dwie części (70% próbek uczących, 30% testowych)
data_train, data_test, labels_train, labels_test = train_test_split(data, labels, test_size=0.3)

# trening klasyfikatora 3NN na danych uczących
c_model = KNeighborsClassifier(n_neighbors=3)
c_model.fit(data_train, labels_train)

# wynik klasyfikacji dla zbioru testowego
y = c_model.predict(data_test)
```

---

### Macierz pomyłek (*confusion matrix*)

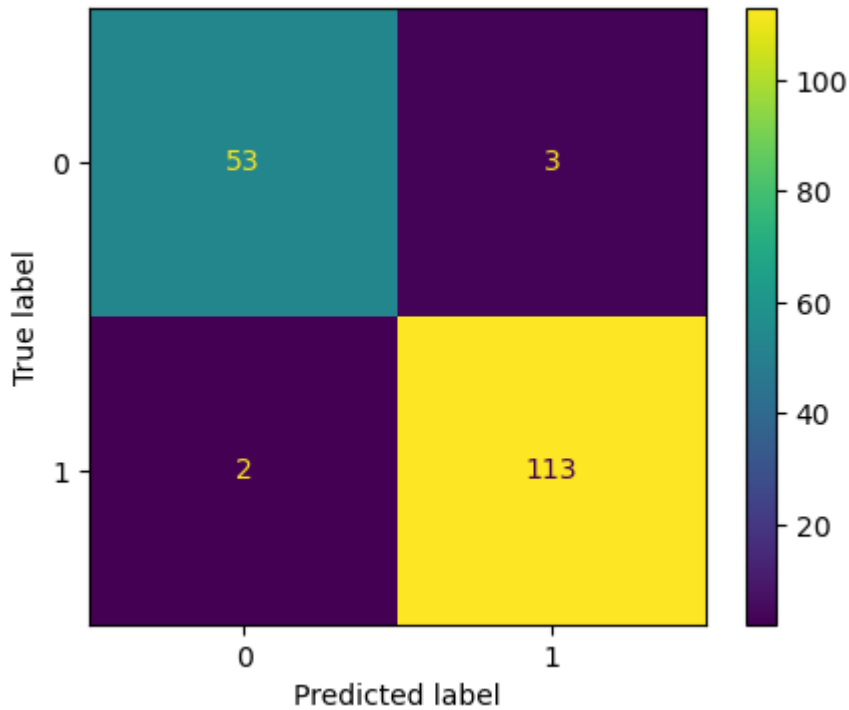
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
cm = confusion_matrix(labels_test, y)

print(f"\n{cm}")
```

```
[[ 53   3]
 [  2 113]]
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```



---

### Dokładność (*accuracy*)

```
from sklearn.metrics import accuracy_score
```

```
# przy użyciu funkcji
acc = accuracy_score(labels_test, y)
print(f"\nacc = {acc:.4f}")

# na podstawie macierzy pomyłek
acc = (cm[0, 0] + cm[1, 1]) / np.sum(cm)
print(f"\nacc = {acc:.4f}")
```

```
acc = 0.9708
```

```
acc = 0.9708
```

---

### Czułość (*sensitivity, recall, true positive rate, TPR*)

```
from sklearn.metrics import recall_score
```

```
# przy użyciu funkcji
se = recall_score(labels_test, y)
print(f"\nse = {se:.4f}")

# na podstawie macierzy pomyłek
se = cm[0, 1] / np.sum(cm[0, :])
print(f"\nse = {se:.4f}")
```

```
se = 0.9826
```

```
se = 0.9464
```

---

### Swoistość (*specificity, true negative rate, TNR*)

```
# na podstawie macierzy pomyłek  
sp = cm[1, 1] / np.sum(cm[1, :])  
print(f"\nsp = {sp:.4f}")
```

```
sp = 0.9826
```

---

### Precyzja, wartość predykcyjna dodatnia (*precision, positive predictive value, PPV*)

```
from sklearn.metrics import precision_score
```

```
# przy użyciu funkcji  
ppv = precision_score(labels_test, y)  
print(f"\nppv = {ppv:.4f}")
```

```
# na podstawie macierzy pomyłek  
ppv = cm[0, 0] / np.sum(cm[:, 0])  
print(f"\nppv = {ppv:.4f}")
```

```
ppv = 0.9741
```

```
ppv = 0.9636
```

---

### Wartość predykcyjna ujemna (*negative predictive value, NPV*)

```
# na podstawie macierzy pomyłek  
npv = cm[1, 1] / np.sum(cm[:, 1])  
print(f"\nnpv = {npv:.4f}")
```

```
npv = 0.9741
```

---

### Miara F1 (*F1 score*)

```
from sklearn.metrics import f1_score
```

```
# przy użyciu funkcji  
f1 = f1_score(labels_test, y)  
print(f"\nf1 = {f1:.4f}")
```

```
# na podstawie macierzy pomyłek
f1 = 2 * (ppv * se) / (ppv + se)
print(f"\nf1 = {ppv:.4f}")
```

f1 = 0.9636

f1 = 0.9636

---

## Wykres ROC (Receiver Operating Characteristic)

```
from sklearn.metrics import roc_curve, auc
```

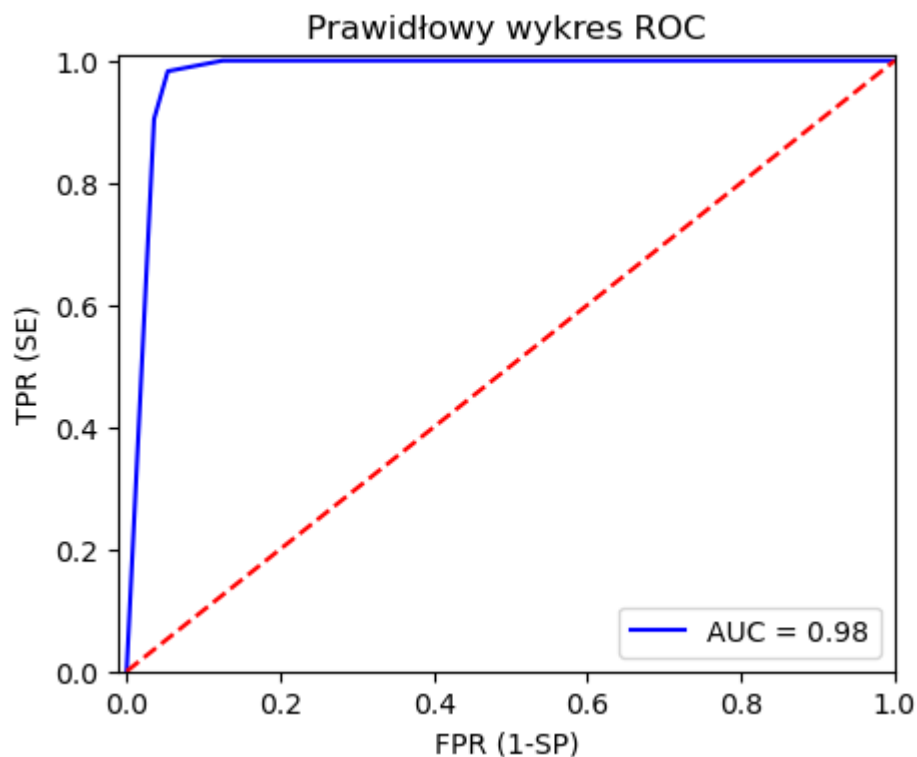
```
# wartości decyzyjne klasyfikatora (lub prawdopodobieństwo) dla zbioru testowego
score = c_model.predict_proba(data_test)
```

```
fpr, tpr, thresholds = roc_curve(labels_test, score[:, 1])
```

```
# pole powierzchni pod krzywą ROC
roc_auc = auc(fpr, tpr)
print(f"\nauc = {roc_auc:.4f}")
```

```
# wykres
umb.roc_plot(fpr, tpr, title="Prawidłowy wykres ROC")
```

auc = 0.9788



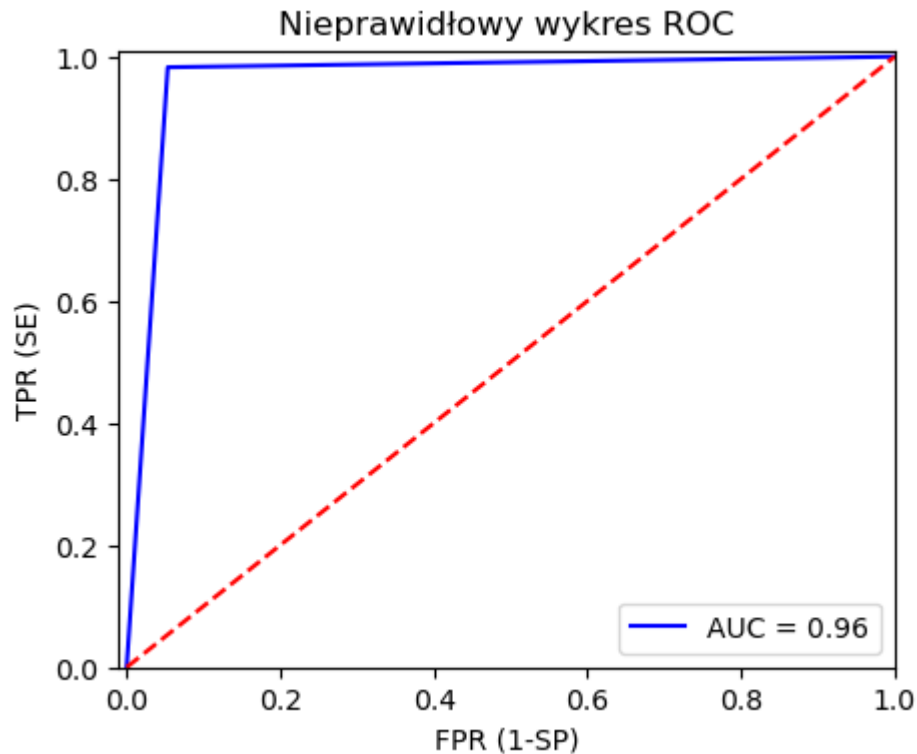
```
# etykiety klas dla zbioru testowego
y = c_model.predict(data_test)
```

```
fpr, tpr, thresholds = roc_curve(labels_test, y)

# pole powierzchni pod krzywą ROC
roc_auc = auc(fpr, tpr)
print(f"\nauc = {roc_auc:.4f}")

# wykres
umb.roc_plot(fpr, tpr, title="Nieprawidłowy wykres ROC")
```

auc = 0.9645




---

## Miary jakości w walidacji krzyżowej

```
from sklearn.model_selection import cross_validate
```

```
# 5-krotna walidacja krzyżowa klasyfikatora 3NN
clf_model = KNeighborsClassifier(n_neighbors=3)

scores = cross_validate(clf_model, data, labels, cv=5, scoring = ["accuracy", "f1", "roc_auc"])
print(f"\nacc = {scores['test_accuracy']}")
print(f"\nf1 = {scores['test_f1']}")
print(f"\nauc = {scores['test_roc_auc']}")
```

acc = [0.97368421 0.96491228 0.98245614 0.95614035 0.98230088]

f1 = [0.97931034 0.97222222 0.98630137 0.96551724 0.98611111]

auc = [0.98689813 0.96953816 0.98677249 0.9823082 0.99832327]

---

