Tomasz MARKIEWICZ Nr albumu: 136917

Lab2 – **Sprawozdanie z analizy klasyfikacji raka piersi.**

1. Wstęp

Celem tego zadania było zastosowanie klasyfikatora ***Support Vector Machine – SVM*** z wykorzystaniem kernela *linear* i radialnej funkcji bazowej (*RBF*) do problemu klasyfikacji na przykładzie zbioru danych diagnostyczny dotyczących raka piersi w stanie Wisconsin. Baza danych składa się z podziału na łagodnego (nie rozprzestrzenia się na resztę ciała) lub złośliwego (rozprzestrzenia się na resztę ciała). W ramach zadania zostały zbadane różne kombinacie parametrów *kernel* oraz wartości parametru *C*. Do tego celu wykorzystano bibliotekę `scikit-learn`; z pakietu *sklearn.datasets* wczytana została funkcja *breast_cancer*. Przedstawiony został (Rysunek 1.) zbiór z podziałem danych na zbiór uczący 70% i zbiór testowy 30% za pomocą funkcji *train_test_split* (Kod 1.)

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split as tts
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from tabulate import tabulate

'''Wczytanie zbioru danych breast cancer'''
data = datasets.load_breast_cancer()
X, y = data.data, data.target

'''Podział danych na zbiór uczący i testujący'''
X_train, X_test, y_train, y_test = tts(X, y, test_size=0.3, random_state=42)
print(f"Wartośc zbioru testującego: „, len(X_test), „obietków", „\nWartość zbiór
uczącego: „, len(X_train), „obiektów")
```

Kod 1. Fragment kodu przy zachowaniu stałości danych wejściowych.

```
Wartośc zbioru testującego:  171 obietków
Wartość zbiór uczącego:  398 obiektów
```

Rysunek 1. Podział próbek.

2. Klasyfikator SVC i pętla w celu znalezienia optymalnych parametrów

Polska nazwa *maszyna wektorów nośnych* średnio do mnie przemawia więc zastanę przy angielskim skrócie. SVM to nadzorowany algorytm uczenia maszynowego stosowany do problemów klasyfikacji liniowej i nieliniowej, regresji, a nawet wykrywania wartości odstających. Głównym celem jest znalezienie optymalnej hiperpłaszczyzny w przestrzeni N-wymiarowej, która może oddzielić punkty danych w różnych klasach przestrzeni cech. Wykorzystywany jest do takich zadań jak klasyfikacja tekstu, obrazu, wykrywanie spamu, identyfikacja pisma odręcznego, analiza ekspresji genów, wykrywanie twarzy oraz wykrywanie anomalii. SVM potrafi

zarządzać danymi wielowymiarowymi i relacjami nieliniowymi. W przypadku klasyfikacji, SVM szuka hiperpłaszczyzny, która najlepiej separuje różne klasy danych. Kernel SVM to rozszerzenie SVM, które umożliwia przekształcanie danych do przestrzeni o wyższych wymiarach, użyte w niniejszym zadaniu to:

- kernel='linear' – prosta hiperpłaszczyzna separująca dane.
- kernel='rbf' – bardzo skomplikowane funkcje przekształcające.

Funkcja `train_and_evaluate_classifier` pełni rolę nauczania klasyfikatora SVM na danych uczących, ocenę jego skuteczności na danych testowych, po czym generuje macierzy konfuzji oraz wizualizację wyników, badając optymalny parametru *C* i wpływ argumentu *kernel.* Dla każdej kombinacji kernela i parametru *C* są tworzone dwa klasyfikatory: jeden z *kernel='linear'* i drugi z *kernel='rbf'*(Kod 2.) Dla każdej kombinacji parametrów przeprowadzona jest analiza dokładności klasyfikacji oraz utworzona zostaje macierz konfuzji. Klasyfikatory są uczone na danych uczących za pomocą klasy SVC, a dane są przeskalowywane przy użyciu StandardScaler. (Rysunek 2.)

```
'''Przypisanie etykiet klas raka złośliwego i łagodnego'''
malignant_label = data.target_names[1]
benign_label = data.target_names[0]

'''Nauczenie klasyfikatora SVC na danych uczących'''
def train_and_evaluate_classifier(kernel_type, C_value, print_results=True):
    clf = make_pipeline(StandardScaler(), SVC(kernel=kernel_type, C=C_value))
    clf.fit(X_train, y_train)

    '''Obliczenie dokładności klasyfikacji na zbiorze danych testowych'''
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    '''Stworzenie macierzy konfuzji'''
    cm = confusion_matrix(y_test, y_pred)

    '''Wykonanie wizualizacji z nazwami, siatką i legendą'''
    plt.figure(figsize=(10, 5))

    '''Wykres punktowy dla rzeczywistego podziału klas'''
    plt.subplot(1, 2, 1)
    plt.scatter(X_test[y_test == 1, 0], X_test[y_test == 1, 1], c='red', marker='+',
                label=f"{malignant_label} (Rak złośliwy)")
    plt.scatter(X_test[y_test == 0, 0], X_test[y_test == 0, 1], c='green', marker='_',
                label=f"{benign_label} (Rak łagodny)")
    plt.title("Rzeczywisty")
    plt.xlabel("Cecha 1")
    plt.ylabel("Cecha 2")
    plt.grid(color='grey', linestyle="--", linewidth=1.15, alpha=0.5)
    plt.legend()

    '''Wykres punktowy dla predykcji klasyfikatora'''
    plt.subplot(1, 2, 2)
    plt.scatter(X_test[y_test == 1, 0], X_test[y_test == 1, 1], c='red', marker='+',
                label=f"{malignant_label} (Rak złośliwy)")
    plt.scatter(X_test[y_test == 0, 0], X_test[y_test == 0, 1], c='green', marker='_',
                label=f"{benign_label} (Rak łagodny)")
    plt.title("Przewidywany")
```

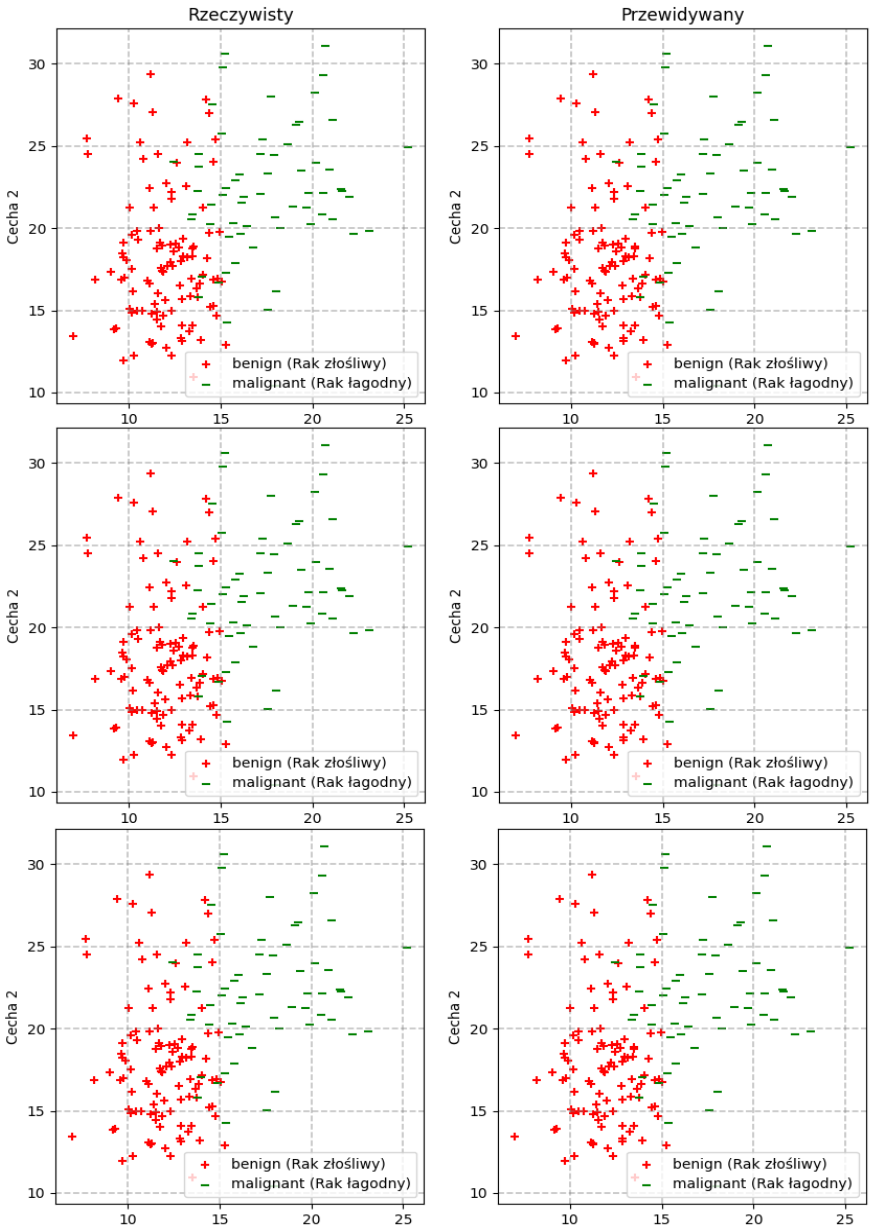
```
    plt.xlabel("Cecha 1")
    plt.ylabel("Cecha 2")
    plt.grid(color='grey', linestyle="--", linewidth=1.15, alpha=0.5)
    plt.legend()

    plt.show()

    '''Umieszczenie wyników w tabeli przy użyciu tabulate'''
    if print_results:
        headers = ["Kernel", "C", "Accuracy", "Confusion Matrix"]
        table_data = [[kernel_type, C_value, accuracy, cm]]
        table = tabulate(table_data, headers, tablefmt='pretty', colalign=("center",
"center", "center", "center"))
        print(f"\nWynik dla kernel={kernel_type}, przy C={C_value}:\n{table}")

'''Pętla w celu znalezienia optymalnych parametrów C i badania wpływu argumentu
"kernel"'''
kernel_types = ['linear', 'rbf']
C_values = [0.1, 1, 10]

for kernel_type in kernel_types:
    for C_value in C_values:
        train_and_evaluate_classifier(kernel_type, C_value)
```

Kod 2. Dalsza część kodu, funkcja `train_and_evaluate_classifier` i badanie parametrów.

```
Wynik dla kernel=linear, przy C=0.1:
+--------+-----+--------------------+------------------+
| Kernel |  C  |      Accuracy      | Confusion Matrix |
+--------+-----+--------------------+------------------+
| linear | 0.1 | 0.9824561403508771 |    [[ 61   2]    |
|        |     |                    |    [  1 107]]    |
+--------+-----+--------------------+------------------+
```

```
Wynik dla kernel=linear, przy C=1:
+--------+---+--------------------+------------------+
| Kernel | C |      Accuracy      | Confusion Matrix |
+--------+---+--------------------+------------------+
| linear | 1 | 0.9766081871345029 |    [[ 61   2]    |
|        |   |                    |    [  2 106]]    |
+--------+---+--------------------+------------------+
```

```
Wynik dla kernel=linear, przy C=10:
+--------+----+--------------------+------------------+
| Kernel | C  |      Accuracy      | Confusion Matrix |
+--------+----+--------------------+------------------+
| linear | 10 | 0.9649122807017544 |    [[ 61   2]    |
|        |    |                    |    [  4 104]]    |
+--------+----+--------------------+------------------+
```



Wynik dla kernel=rbf, przy C=0.1:

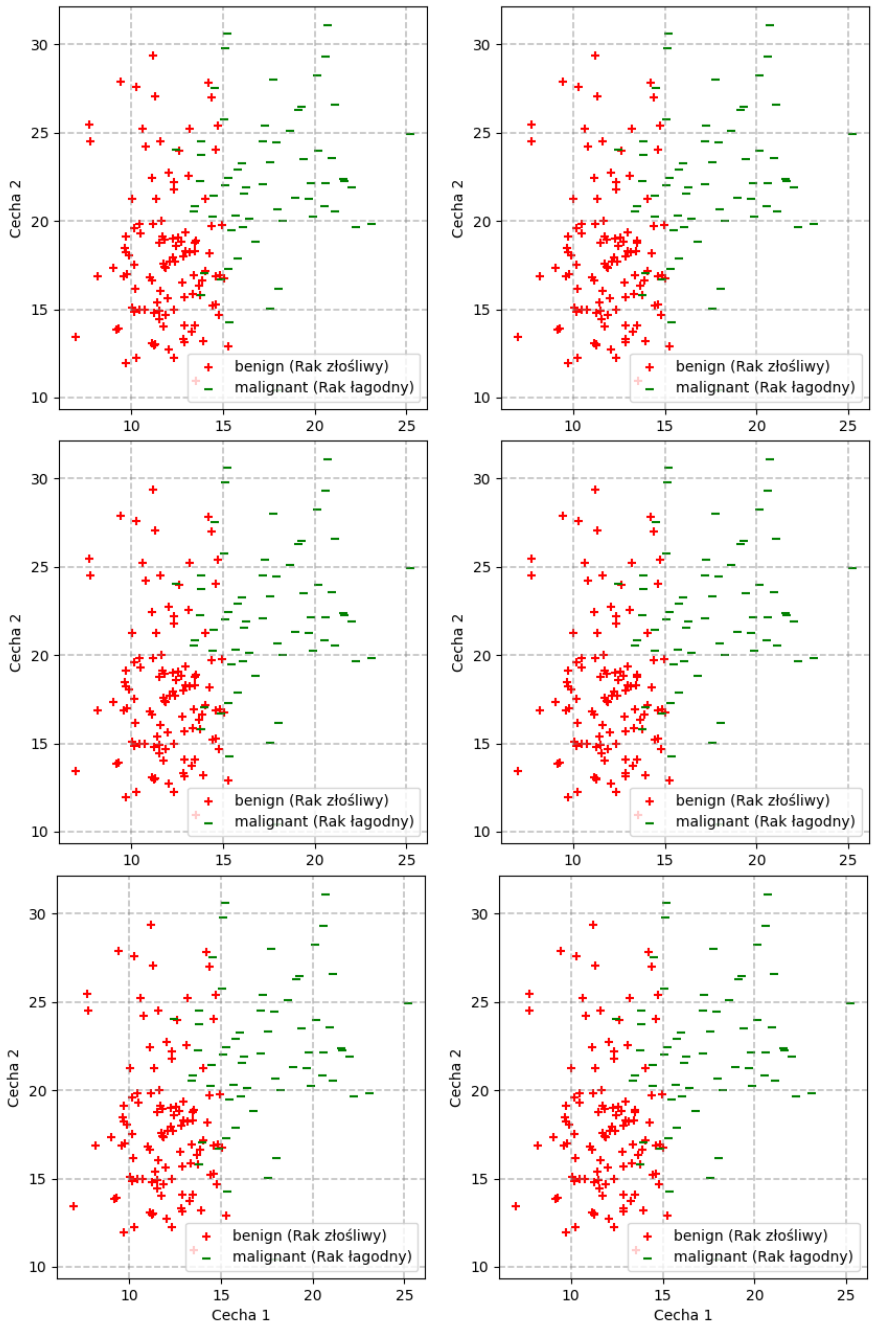
Kernel	C	Accuracy	Confusion Matrix
rbf	0.1	0.935672514619883	[[56 7] [4 104]]

Wynik dla kernel=rbf, przy C=1:

Kernel	C	Accuracy	Confusion Matrix
rbf	1	0.9766081871345029	[[61 2] [2 106]]

Wynik dla kernel=rbf, przy C=10:

Kernel	C	Accuracy	Confusion Matrix
rbf	10	0.9766081871345029	[[61 2] [2 106]]



Rysunek 2. Wyniki w formie graficznej, macierzą konfuzji przy różnych wariantach *C* oraz *kernel*.