# Perceptron oraz Adaline

Aleksander Kłak, prowadzący Dr Marek Bazan

18 czerwca 2023

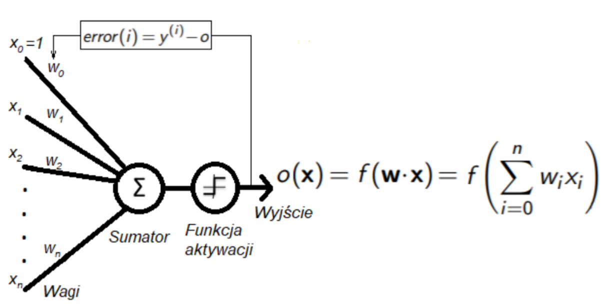
## Spis treści



# 1 Cel ćwiczenia

# 2 Przebieg zadania

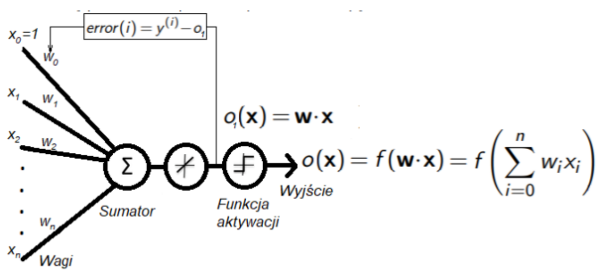
## 2.1 Perceptron




Rysunek 1: Schemat perceptronu

Dla zbioru $Z = (x^{(}1), y^{(}1)), ..., (x^{(}N), y^{(}N))$
od ustalonej liczby n¡sub¿epoch¡/sub¿ nalezy iterować po zbiorze Z dla i=1,...,N nalezy obliczyć wagi, $error(i) = y^{(}i) - o(x^{(}i))$
$\Delta w = \eta * error(i) * x^{(}i)$
$w = w + \Delta w$ $\eta$ to współczynnik uczenia z przedziału (0,1). Zbiór Z to 80

## 2.2 Adaline

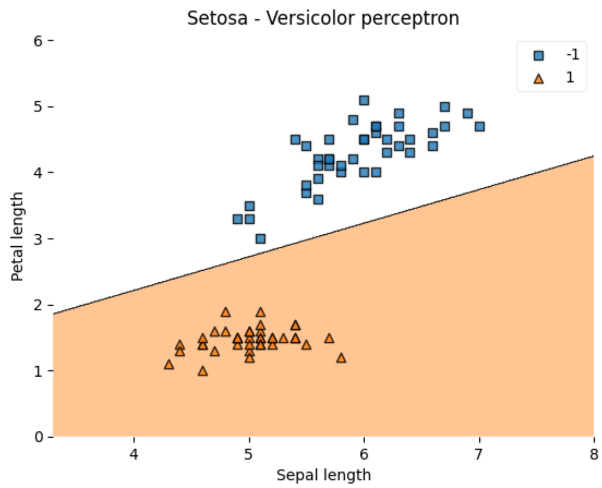



Rysunek 2: Schemat Adaline

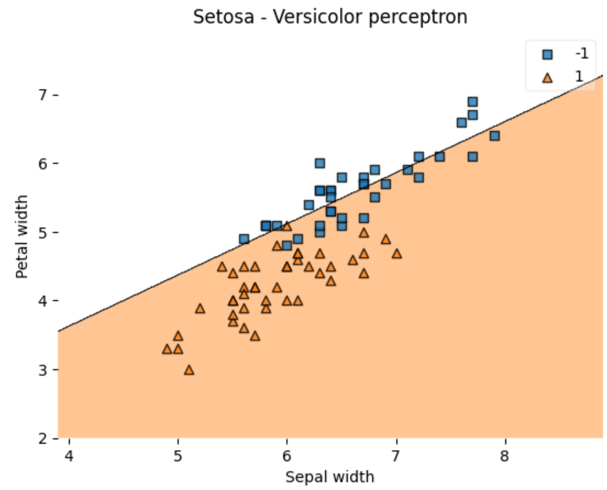
Dla zbioru $Z = (x^{(}1), y^{(}1)), ..., (x^{(}N), y^{(}N))$
od ustalonej liczby n¡sub¿epoch¡/sub¿ nalezy iterować po zbiorze Z
dla i=1,...,N nalezy obliczyć wagi,
$error(i) = y^{(}i) - o_1(x^{(}i))$
$\Delta w = \eta * error(i) * x^{(}i)$
$w = w + \Delta w$
$\eta$ to współczynnik uczenia z przedziału (0,1). Zbiór Z to 80

# 3 Wyniki

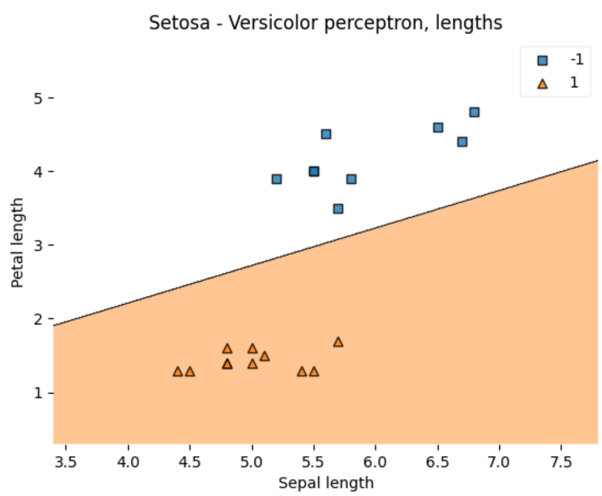
## 3.1 Perceptron, 2 klasy



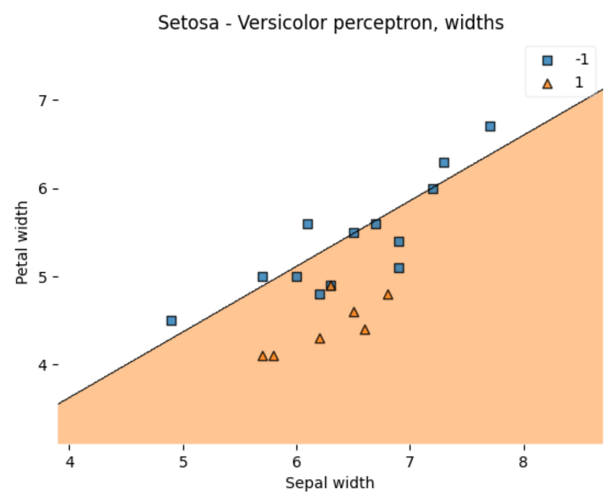

Rysunek 3: Trening na długościach




Rysunek 4: Trening na szerokościach

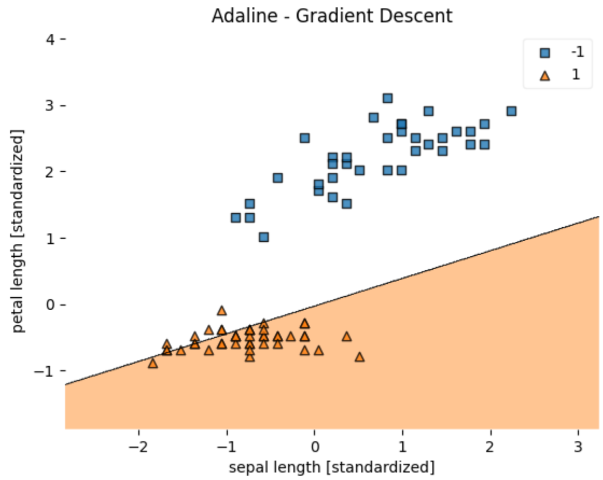



Rysunek 5: Test na długościach

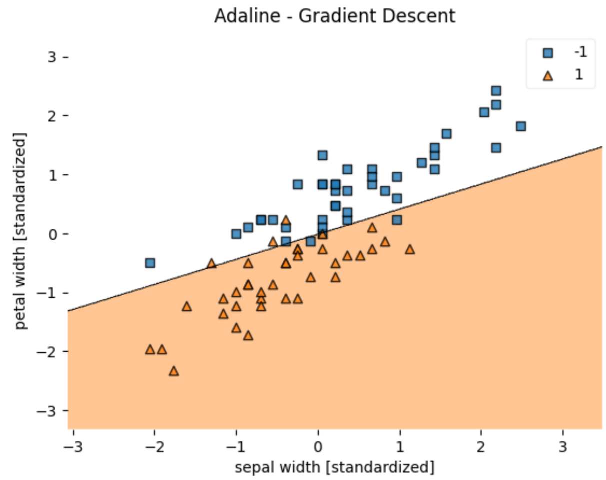



Rysunek 6: Test na szerokościach

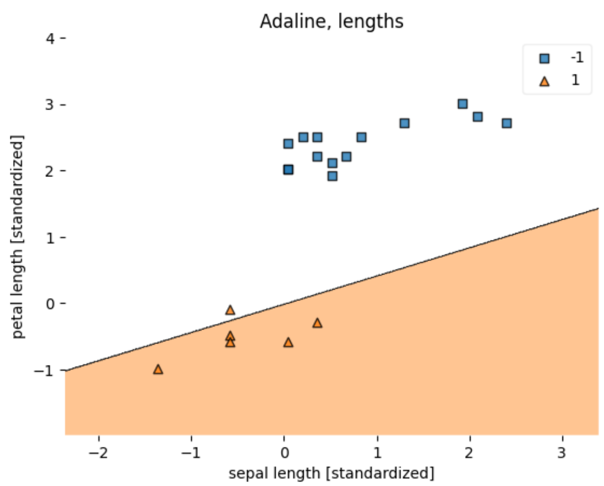
## 3.2 Adaline, 2 klasy



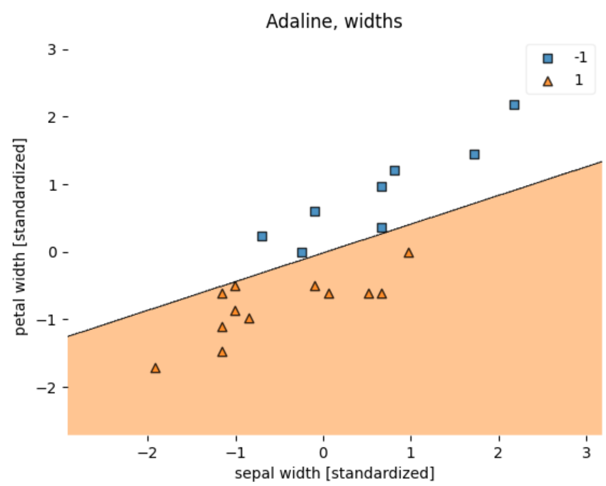

Rysunek 7: Trening na długościach




Rysunek 8: Trening na szerokościach



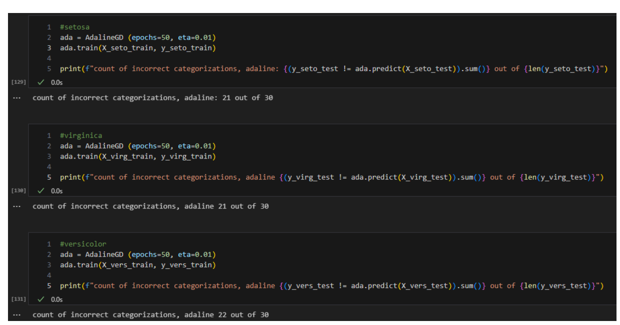

Rysunek 9: Test na długościach




Rysunek 10: Test na szerokościach

## 3.3 3 klasy


Rysunek 11: Liczba błędnych klasyfikacji Perceptron




Rysunek 12: Liczba błędnych klasyfikacji Adaline

## 3.4 Tworzenie klasyfikatora dla różnych kombinacji dwóch parametrów oraz porównanie tego z 4 parametrami

Wpierw $\eta = 0.01$, nastepnie $\eta = 0.5$
W kolejności setosa, virginica, versicolor

```
Perceptron count of incorrect categorizations for Sepal length and sepal width: 1
Perceptron count of incorrect categorizations for petal length and petal width: 0
Perceptron count of incorrect categorizations for Sepal length and petal length: 0
Perceptron count of incorrect categorizations for Sepal length and petal width: 0
Perceptron count of incorrect categorizations for sepal width and petal length: 0
Perceptron count of incorrect categorizations for 4DIM: 0
Average accuracy Perceptron: 0.9944444444444445

Perceptron count of incorrect categorizations for Sepal length and sepal width: 15
Perceptron count of incorrect categorizations for petal length and petal width: 1
Perceptron count of incorrect categorizations for Sepal length and petal length: 1
Perceptron count of incorrect categorizations for Sepal length and petal width: 3
Perceptron count of incorrect categorizations for sepal width and petal length: 7
Perceptron count of incorrect categorizations for 4DIM: 4
Average accuracy Perceptron: 0.8277777777777778

Perceptron count of incorrect categorizations for Sepal length and sepal width: 17
Perceptron count of incorrect categorizations for petal length and petal width: 11
Perceptron count of incorrect categorizations for Sepal length and petal length: 9
Perceptron count of incorrect categorizations for Sepal length and petal width: 16
Perceptron count of incorrect categorizations for sepal width and petal length: 20
Perceptron count of incorrect categorizations for 4DIM: 12
Average accuracy Perceptron: 0.5277777777777778

Perceptron count of incorrect categorizations for Sepal length and sepal width: 0
Perceptron count of incorrect categorizations for petal length and petal width: 0
Perceptron count of incorrect categorizations for Sepal length and petal length: 0
Perceptron count of incorrect categorizations for Sepal length and petal width: 0
Perceptron count of incorrect categorizations for sepal width and petal length: 0
Perceptron count of incorrect categorizations for 4DIM: 0
Average accuracy Perceptron: 1.0

Perceptron count of incorrect categorizations for Sepal length and sepal width: 12
Perceptron count of incorrect categorizations for petal length and petal width: 6
Perceptron count of incorrect categorizations for Sepal length and petal length: 1
Perceptron count of incorrect categorizations for Sepal length and petal width: 2
Perceptron count of incorrect categorizations for sepal width and petal length: 1
Perceptron count of incorrect categorizations for 4DIM: 1
Average accuracy Perceptron: 0.8722222222222222

Perceptron count of incorrect categorizations for Sepal length and sepal width: 12
Perceptron count of incorrect categorizations for petal length and petal width: 8
Perceptron count of incorrect categorizations for Sepal length and petal length: 17
Perceptron count of incorrect categorizations for Sepal length and petal width: 10
Perceptron count of incorrect categorizations for sepal width and petal length: 9
Perceptron count of incorrect categorizations for 4DIM: 21
Average accuracy Perceptron: 0.5722222222222223
```

Rysunek 13: Liczba błędnych klasyfikacji, perceptron

```
Adaline count of incorrect categorizations for Sepal length and sepal width: 0
Adaline count of incorrect categorizations for petal length and petal width: 27
Adaline count of incorrect categorizations for Sepal length and petal length: 28
Adaline count of incorrect categorizations for Sepal length and petal width: 27
Adaline count of incorrect categorizations for sepal width and petal length: 0
Adaline count of incorrect categorizations for 4DIM: 18
Average accuracy Adaline: 0.4444444444444444

Adaline count of incorrect categorizations for Sepal length and sepal width: 4
Adaline count of incorrect categorizations for petal length and petal width: 24
Adaline count of incorrect categorizations for Sepal length and petal length: 21
Adaline count of incorrect categorizations for Sepal length and petal width: 26
Adaline count of incorrect categorizations for sepal width and petal length: 2
Adaline count of incorrect categorizations for 4DIM: 17
Average accuracy Adaline: 0.4777777777777777

Adaline count of incorrect categorizations for Sepal length and sepal width: 7
Adaline count of incorrect categorizations for petal length and petal width: 17
Adaline count of incorrect categorizations for Sepal length and petal length: 20
Adaline count of incorrect categorizations for Sepal length and petal width: 16
Adaline count of incorrect categorizations for sepal width and petal length: 5
Adaline count of incorrect categorizations for 4DIM: 21
Average accuracy Adaline: 0.5222222222222223

Adaline count of incorrect categorizations for Sepal length and sepal width: 27
Adaline count of incorrect categorizations for petal length and petal width: 29
Adaline count of incorrect categorizations for Sepal length and petal length: 26
Adaline count of incorrect categorizations for Sepal length and petal width: 23
Adaline count of incorrect categorizations for sepal width and petal length: 28
Adaline count of incorrect categorizations for 4DIM: 23
Average accuracy Adaline: 0.1333333333333333

Adaline count of incorrect categorizations for Sepal length and sepal width: 22
Adaline count of incorrect categorizations for petal length and petal width: 20
Adaline count of incorrect categorizations for Sepal length and petal length: 26
Adaline count of incorrect categorizations for Sepal length and petal width: 23
Adaline count of incorrect categorizations for sepal width and petal length: 23
Adaline count of incorrect categorizations for 4DIM: 20
Average accuracy Adaline: 0.25555555555555554

Adaline count of incorrect categorizations for Sepal length and sepal width: 17
Adaline count of incorrect categorizations for petal length and petal width: 17
Adaline count of incorrect categorizations for Sepal length and petal length: 18
Adaline count of incorrect categorizations for Sepal length and petal width: 11
Adaline count of incorrect categorizations for sepal width and petal length: 21
Adaline count of incorrect categorizations for 4DIM: 19
Average accuracy Adaline: 0.4277777777777778
```

Rysunek 14: Liczba błędnych klasyfikacji, adaline

## 3.5 Budowa klasyfikatorów dla każdej z klas

```
Przypisanie dla danych testowych:
Dane: [4.4 2.9 1.4 0.2] -> Przewidywana klasa: Iris-setosa     (Oryginalna klasa: Iris-setosa    ), Wyniki: [0.64, -0.23, 0.25]
Dane: [6.7 3.  5.2 2.3] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-virginica ), Wyniki: [0.46, -0.66, 1.43]
Dane: [7.7 2.8 6.7 2. ] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-virginica ), Wyniki: [0.11, -0.09, 1.16]
Dane: [5.8 2.7 4.1 1. ] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-versicolor), Wyniki: [0.3, -0.05, 0.59]
Dane: [6.  3.  4.8 1.8] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-virginica ), Wyniki: [0.36, -0.45, 1.05]
Dane: [5.5 2.5 4.  1.3] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-versicolor), Wyniki: [0.35, -0.19, 0.76]
Dane: [4.5 2.3 1.3 0.3] -> Przewidywana klasa: Iris-setosa     (Oryginalna klasa: Iris-setosa    ), Wyniki: [0.67, -0.15, 0.36]
Dane: [7.3 2.9 6.3 1.8] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-virginica ), Wyniki: [0.12, -0.09, 1.03]
Dane: [4.7 3.2 1.3 0.2] -> Przewidywana klasa: Iris-setosa     (Oryginalna klasa: Iris-setosa    ), Wyniki: [0.73, -0.35, 0.31]
Dane: [6.3 2.5 4.9 1.5] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-versicolor), Wyniki: [0.27, -0.11, 0.89]
Dane: [6.6 2.9 4.6 1.3] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-versicolor), Wyniki: [0.36, -0.18, 0.84]
Dane: [6.9 3.2 5.7 2.3] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-virginica ), Wyniki: [0.37, -0.61, 1.38]
Dane: [4.3 3.  1.1 0.1] -> Przewidywana klasa: Iris-setosa     (Oryginalna klasa: Iris-setosa    ), Wyniki: [0.69, -0.27, 0.21]
Dane: [6.1 2.6 5.6 1.4] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-virginica ), Wyniki: [0.04, 0.08, 0.68]
Dane: [6.3 2.9 5.6 1.8] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-virginica ), Wyniki: [0.18, -0.24, 0.97]
Dane: [4.9 3.1 1.5 0.1] -> Przewidywana klasa: Iris-setosa     (Oryginalna klasa: Iris-setosa    ), Wyniki: [0.67, -0.21, 0.24]
Dane: [5.  3.2 1.2 0.2] -> Przewidywana klasa: Iris-setosa     (Oryginalna klasa: Iris-setosa    ), Wyniki: [0.79, -0.37, 0.37]
Dane: [6.  2.9 4.5 1.5] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-versicolor), Wyniki: [0.36, -0.31, 0.89]
Dane: [6.8 3.2 5.9 2.3] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-virginica ), Wyniki: [0.31, -0.56, 1.34]
Dane: [5.9 3.  4.2 1.5] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-versicolor), Wyniki: [0.43, -0.41, 0.92]
Dane: [7.7 2.6 6.9 2.3] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-virginica ), Wyniki: [0.12, -0.17, 1.34]
Dane: [5.6 2.9 3.6 1.3] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-versicolor), Wyniki: [0.49, -0.39, 0.84]
Dane: [5.  3.5 1.3 0.3] -> Przewidywana klasa: Iris-setosa     (Oryginalna klasa: Iris-setosa    ), Wyniki: [0.8, -0.49, 0.42]
Dane: [5.4 3.9 1.3 0.4] -> Przewidywana klasa: Iris-setosa     (Oryginalna klasa: Iris-setosa    ), Wyniki: [0.9, -0.67, 0.55]
Dane: [6.1 2.8 4.7 1.2] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-versicolor), Wyniki: [0.24, -0.06, 0.68]
Dane: [7.7 3.8 6.7 2.2] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-virginica ), Wyniki: [0.22, -0.5, 1.28]
Dane: [6.7 3.3 5.7 2.5] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-virginica ), Wyniki: [0.4, -0.75, 1.49]
Dane: [5.  3.6 1.4 0.2] -> Przewidywana klasa: Iris-setosa     (Oryginalna klasa: Iris-setosa    ), Wyniki: [0.76, -0.44, 0.34]
Dane: [6.4 3.1 5.5 1.8] -> Przewidywana klasa: Iris-virginica  (Oryginalna klasa: Iris-virginica ), Wyniki: [0.23, -0.32, 1.0]
Dane: [5.1 3.5 1.4 0.2] -> Przewidywana klasa: Iris-setosa     (Oryginalna klasa: Iris-setosa    ), Wyniki: [0.77, -0.41, 0.35]
Liczba nieporozumień: 8
```

Rysunek 15: Badanie dla jakiej danej jest jaka klasa

# 4 Wnioski

Współczynnik uczenia nie ma większego wpływu na klasyfikację trzech klas metodą perceptronu. Adaline sprawdza się lepiej przy dwóch klasach. Perceptron sprawdza się lepiej przy trzech klasach

# 5 Kod

## 5.1 Implementacja Perceptronu

Listing 1: RBF

```
class Perceptron(object):

    def ˙˙init˙˙(self, eta=0.01, epochs=50):
        self.eta = eta
        self.epochs = epochs

    def train(self, X, y):

        self.w˙ = np.zeros(1 + X.shape[1])
        self.errors˙ = []

        for ˙ in range(self.epochs):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w˙[1:] += update * xi
                self.w˙[0] += update
                errors += int(update != 0.0)
            self.errors˙.append(errors)
        return self

    def net˙input(self, X):
        return np.dot(X, self.w˙[1:]) + self.w˙[0]

    def predict(self, X):
        return np.where(self.net˙input(X) ¿= 0.0, 1, -1)
```

## 5.2 Implementacja Adaline

Listing 2: Adaline

```
class AdalineGD(object):

    def ˙˙init˙˙(self, eta=0.01, epochs=50):
        self.eta = eta
        self.epochs = epochs

    def train(self, X, y):

        self.w˙ = np.zeros(1 + X.shape[1])
        self.cost˙ = []

        for i in range(self.epochs):
            output = self.net˙input(X)
            errors = (y - output)
            self.w˙[1:] += self.eta * X.T.dot(errors)
            self.w˙[0] += self.eta * errors.sum()
```

```
            cost = (errors**2).sum() / 2.0
            self.cost˙.append(cost)
        return self

    def net˙input(self, X):
        return np.dot(X, self.w˙[1:]) + self.w˙[0]

    def activation(self, X):
        return self.net˙input(X)

    def predict(self, X):
        return np.where(self.activation(X) ¿= 0.0, 1, -1)
```

## 5.3 Skrypt

Listing 3: Skrypt

```
y˙data = iris.iloc[0:150, 4].values
y˙seto = np.where(y˙data == 'Iris-setosa', 1, -1)
y˙virg = np.where(y˙data == 'Iris-virginica', 1, -1)
y˙vers = np.where(y˙data == 'Iris-versicolor', 1, -1)
# sepal length and petal length
X˙data = iris.iloc[0:150, [0, 1, 2, 3]].values
X˙seto˙train, X˙seto˙test, y˙seto˙train, y˙seto˙test = train˙test˙split(X˙data, y˙seto,
    test˙size=0.20)
X˙virg˙train, X˙virg˙test, y˙virg˙train, y˙virg˙test = train˙test˙split(X˙data, y˙virg,
    test˙size=0.20)
X˙vers˙train, X˙vers˙test, y˙vers˙train, y˙vers˙test = train˙test˙split(X˙data, y˙vers,
    test˙size=0.20) #dla klasy Setosa

# dla klasy Setosa

ppn˙seto = Perceptron(epochs=50, eta=0.001)
ppn˙seto.train(X˙seto˙train, y˙seto˙train)

print(f"count of incorrect categorizations for 3 classes, lengths: –(y˙seto˙test != ppn˙seto.
    predict(X˙seto˙test)).sum()˝ out of –len(y˙seto˙test)˝")

# dla klasy Virginica
ppn˙virg = Perceptron(epochs=50, eta=0.001)
ppn˙virg.train(X˙virg˙train, y˙virg˙train)


print(f"count of incorrect categorizations for 3 classes, lengths: –(y˙virg˙test != ppn˙virg.
    predict(X˙virg˙test)).sum()˝ out of –len(y˙virg˙test)˝")

# dla klasy Versicolor
ppn˙vers = Perceptron(epochs=50, eta=0.01)
ppn˙vers.train(X˙vers˙train, y˙vers˙train)

print(f"count of incorrect categorizations for 3 classes, lengths: –(y˙vers˙test != ppn˙vers.
    predict(X˙vers˙test)).sum()˝ out of –len(y˙vers˙test)˝")

#3klasy dla Adaline
# sepal length and petal length
X˙data =  iris.iloc[0:150, [0, 1, 2, 3]].values
X˙std = np.copy(X˙data)
X˙std[:,0] = (X˙data[:,0] - X˙data[:,0].mean()) / X˙data[:,0].std()
X˙std[:,1] = (X˙data[:,1] - X˙data[:,1].mean()) / X˙data[:,1].std()
y˙seto = np.where(y˙data == 'Iris-setosa', 1, -1)
```

```
y˙virg= np.where(y˙data == 'Iris-virginica', 1, -1)
y˙vers = np.where(y˙data == 'Iris-versicolor', 1, -1)
X˙seto˙train, X˙seto˙test, y˙seto˙train, y˙seto˙test = train˙test˙split(X˙std, y˙seto, test˙size=0.20, shuffle = 4)
X˙virg˙train, X˙virg˙test, y˙virg˙train, y˙virg˙test = train˙test˙split(X˙std, y˙virg, test˙size=0.20, shuffle = 4)
X˙vers˙train, X˙vers˙test, y˙vers˙train, y˙vers˙test = train˙test˙split(X˙std, y˙vers, test˙size=0.20, shuffle = 4)

#setosa
ada = AdalineGD (epochs=50, eta=0.01)
ada.train(X˙seto˙train, y˙seto˙train)

print(f"count of incorrect categorizations, adaline: –(y˙seto˙test != ada.predict(X˙seto˙test)).sum()˝ out of –len(y˙seto˙test)˝")

#virginica
ada = AdalineGD (epochs=50, eta=0.01)
ada.train(X˙virg˙train, y˙virg˙train)

print(f"count of incorrect categorizations, adaline –(y˙virg˙test != ada.predict(X˙virg˙test)).sum()˝ out of –len(y˙virg˙test)˝")

#versicolor
ada = AdalineGD (epochs=50, eta=0.01)
ada.train(X˙vers˙train, y˙vers˙train)

print(f"count of incorrect categorizations, adaline –(y˙vers˙test != ada.predict(X˙vers˙test)).sum()˝ out of –len(y˙vers˙test)˝")

y˙data = iris.iloc[0:150, 4].values
y˙seto = np.where(y˙data == 'Iris-setosa', 1, -1)
y˙virg = np.where(y˙data == 'Iris-virginica', 1, -1)
y˙vers = np.where(y˙data == 'Iris-versicolor', 1, -1)
ppn˙acc1 = –"seto": 0, "virg": 0, "vers": 0˝
ppn˙acc2 = –"seto": 0, "virg": 0, "vers": 0˝
ada˙acc1 = –"seto": 0, "virg": 0, "vers": 0˝
ada˙acc2 = –"seto": 0, "virg": 0, "vers": 0˝


# all possible data combinations
perm = [["Sepal length and sepal width",[0,1]],["petal length and petal width",[2,3]],["Sepal length and petal length",[0,2]], ["Sepal length and petal width",[0,3]], ["sepal width and petal length",[1,2]],["4DIM",[0,1,2,3]]]

#Perceptron
def ppn˙run(y, eta = 0.01):
    accuracy = 0
    for each in perm:
        # sepal length and petal length
        X˙data = iris.iloc[0:150, each[1]].values
        # Data for perceptron
        X˙train, X˙test , y˙train, y˙test = train˙test˙split(X˙data, y, test˙size=0.20)
        ppn = Perceptron(epochs=50, eta=eta)
        ppn.train(X˙train, y˙train)
        print(f"Perceptron count of incorrect categorizations for –each[0]˝: –(y˙test != ppn.predict(X˙test)).sum()˝")
        accuracy +=(y˙test != ppn.predict(X˙test)).sum()/len(y˙test)
    print(f"Average accuracy Perceptron: –(len(perm)-accuracy)/len (perm)˝")
    print("")
    return (len(perm)-accuracy)/len(perm)

```

```python
ppn_acc1["seto"] = ppn_run(y_seto)
ppn_acc1["virg"] = ppn_run(y_virg)
ppn_acc1["vers"] = ppn_run(y_vers)

ppn_acc2["seto"] = ppn_run(y_seto, eta = 0.5)
ppn_acc2["virg"] = ppn_run(y_virg, eta = 0.5)
ppn_acc2["vers"] = ppn_run(y_vers, eta = 0.5)


#Adaline
def ada_run(y, eta = 0.01):
    accuracy = 0
    for each in perm:
        # sepal length and petal length
        X_data = iris.iloc[0:150, each[1]].values
        # standardized data for Adaline
        X_std= np.copy(X_data)

        X_std[:,0] = (X_data[:,0] - X_data[:,0].mean()) / X_data[:,0].std()
        X_std[:,1] = (X_data[:,1] - X_data[:,1].mean()) / X_data[:,1].std()
        X_std_train, X_std_test, y_std_train, y_std_test = train_test_split(X_std, y, test_size=0.20, shuffle=4)
        ada = AdalineGD (epochs=50, eta=eta)
        ada.train(X_std_train, y_std_train)
        print(f"Adaline count of incorrect categorizations for {each[0]}: {(y_std_test != ada.predict(X_std_test)).sum() }")
        accuracy +=(y_std_test != ada.predict(X_std_test)).sum()/len(y_std_test)
    print (f"Average accuracy Adaline: {(len(perm)-accuracy)/len(perm)}")
    print("")
    return (len(perm)-accuracy)/len (perm)

ada_acc1["seto"] = ada_run(y_seto)
ada_acc1["virg"] = ada_run(y_virg)
ada_acc1["vers"] = ada_run(y_vers)

ada_acc2["seto"] = ada_run(y_seto, eta = 0.5)
ada_acc2["virg"] = ada_run(y_virg, eta = 0.5)
ada_acc2["vers"] = ada_run(y_vers, eta = 0.5)

from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import numpy as np

# Przygotowanie danych
iris = pd.read_csv('iris.data', header=None)
X = iris.iloc[:, [0,1,2,3]].values #cechy - dĆugoŻ i szerokoŻ dziaĆki kielicha
y = iris.iloc[:, 4].values #TARGET - nazwa gatunku (Iris-setosa, Iris-versicolor, Iris-virginica)


# PodziaĆ danych na zbiory treningowy i testowy
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Klasyfikator 1: Iris-setosa vs. Iris-versicolor + Iris-virginica
y_train_setosa = np.where(y_train == 'Iris-setosa', 1, 0)
y_test_setosa = np.where(y_test == 'Iris-setosa', 1, 0)
perceptron_setosa = Perceptron(epochs=50, eta=0.01)
perceptron_setosa.train(X_train, y_train_setosa)
pred_setosa = perceptron_setosa.net_input(X_test)

# Klasyfikator 2: Iris-versicolor vs. Iris-setosa + Iris-virginica
y_train_versicolor = np.where(y_train == 'Iris-versicolor', 1, 0)
```

```
y˙test˙versicolor = np.where(y˙test == 'Iris-versicolor', 1, 0)
perceptron˙versicolor = Perceptron(epochs=50, eta=0.01)
perceptron˙versicolor.train(X˙train, y˙train˙versicolor)
pred˙versicolor = perceptron˙versicolor.net˙input(X˙test)

# Klasyfikator 3: Iris-virginica vs. Iris-setosa + Iris-versicolor
y˙train˙virginica = np.where(y˙train == 'Iris-virginica', 1, 0)
y˙test˙virginica = np.where(y˙test == 'Iris-virginica', 1, 0)
perceptron˙virginica = Perceptron(epochs=50, eta=0.01)
perceptron˙virginica.train(X˙train, y˙train˙virginica)
pred˙virginica = perceptron˙virginica.net˙input(X˙test)

# Klasyfikacja danych z trzech klas
final˙pred = []
misclassifications = 0  # Licznik nieporozumie
scores˙list = []
for i in range(len(X˙test)):
    scores = [pred˙setosa[i], pred˙versicolor[i], pred˙virginica[i]]
    scores˙list.append(scores)
    class˙index = np.argmax(scores)
    if class˙index == 0:
        final˙pred.append('Iris-setosa')
    elif class˙index == 1:
        final˙pred.append('Iris-versicolor')
    else:
        final˙pred.append('Iris-virginica')


    # Zliczanie nieporozumie
    if final˙pred[i] != y˙test[i]:
        misclassifications += 1

# WyŻwietlenie przypisa  dla danych testowych
print("Przypisanie dla danych testowych:")
for i, data in enumerate(X˙test):
    print(f"Dane: –data˝ -¿ Przewidywana klasa: –final˙pred[i]+(len('Iris-versicolor')-len(final˙pred[i]))*' '˝ (Oryginalna klasa: –y˙test[i]+(len('Iris-versicolor')-len(y˙test[i]))*' '˝), Wyniki: –list(map(lambda x: round(x, 2), scores˙list[i]))˝")

# WyŻwietlenie liczby nieporozumie
print(f"Liczba nieporozumie : –misclassifications˝")
```