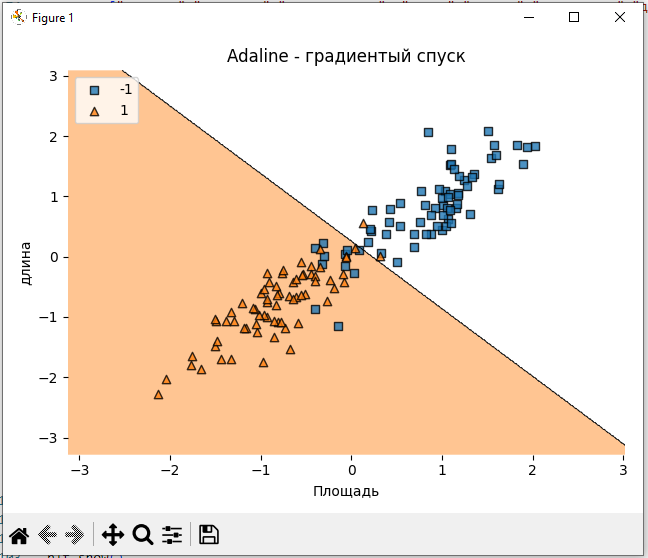
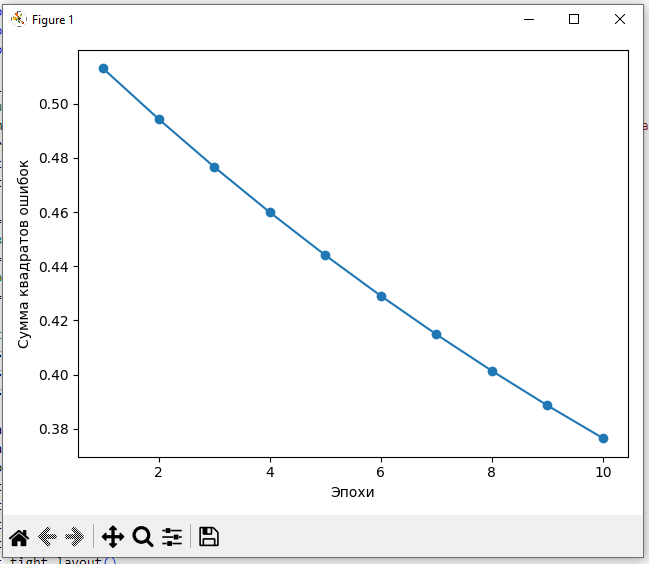


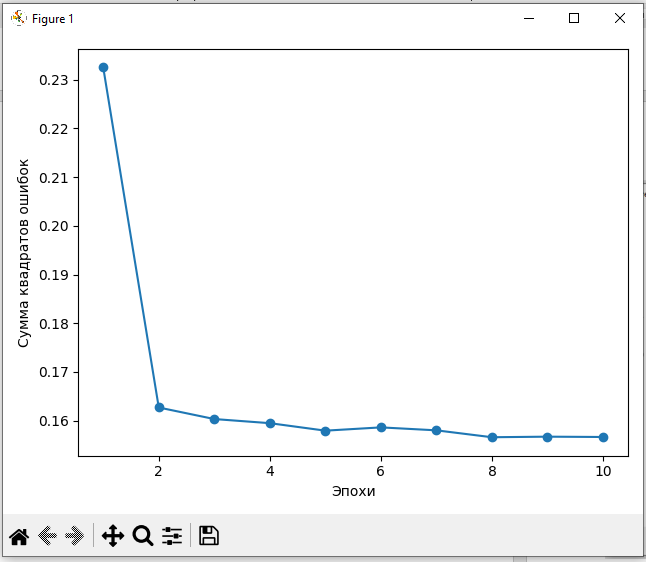
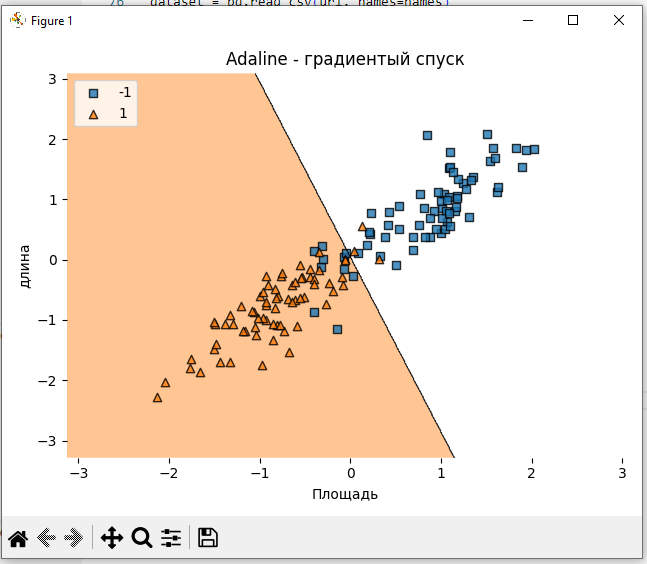
|  |  |
| --- | --- |
| **Скорость 0.01 итераций 11** | **Скорость 0.0001 итераций 20** |
|  |  |

На графиках выше видно, как зависит сумма квадратичных ошибок от количества итераций и скорости обучения

****

****

****

****

По предыдущим графикам видно, как зависят графики от скорости обучения и количества итераций.

В первом случае практически линейный спад суммы ошибок.

Во втором случае рузский спад и далее идет постепенное снижение.

**Код программы**

# Создание основного объекта персептрона.

class Perceptron(object):

    #Начальная скорость обучения и количество итераций.

    def \_\_init\_\_(self, Learn\_Rate=0.5, Iterations=10):

        self.learn\_rate = Learn\_Rate

        self.Iterations = Iterations

        self.errors = []

        self.weights = np.zeros(1 + X.shape[1])

    # Определение метода подгонки для обучения модели.

    def fit(self, X, y):

        self.weights = np.zeros(1 + X.shape[1])

        for i in range(self.Iterations):

            error = 0

            for Xi, target in zip(X, y):

                update = self.learn\_rate \* (target - self.predict(Xi))

                self.weights[1:] += update\*Xi

                self.weights[0] += update

                error += int(update != 0)

            self.errors.append(error)

        return self

    # Метод Net Input для суммирования заданных входных данных матрицы и их соответствующих весов.

    def net\_input(self, X):

        return np.dot(X, self.weights[1:]) + self.weights[0]

    # Метод Predict для прогнозирования классификации входных данных.

    def predict(self, X):

        return np.where(self.net\_input(X) >= 0.0, 1, -1)

import pandas as pd

import numpy as np

from matplotlib import pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from mlxtend.plotting import plot\_decision\_regions

url = "C:/Users/fox/Desktop/ii/data/seeds\_dataset.data"

# url = "F:/ii ЛАБЫ/data/seeds\_dataset.data"

names = ["площадь","периметр","компактность", "длина","ширина","асимметрия","длина канавки ядра","сорт"] #название атрибутов

target\_names = { 1:"Кама" , 2:"Роза" , 3:"Канадка" }

dataset = pd.read\_csv(url, names=names)

dataset.tail()

y = dataset.iloc[0:140,7].values

#тзменение классов для работы

y = np.where(y== 2, -1, 1)

# print(y)

X = dataset.iloc[0:140, [0, 3]].values

# print(X)

plt.scatter(X[: 50, 0], X[: 50, 1], color='red' , marker='o' , label= 'Кама' )

plt.scatter(X[70 :140 , 0] , X[70 :140 , 1], color='blue' , marker= 'X' , label= 'Роза' )

plt.xlabel ( 'Площадь' )

plt.ylabel('длина')

plt.legend(loc='upper left')

plt.show()

#

    # print(X[: 50, 0])  #Площадь первые 50 первого типа сорта КАМА

    # print(X[: 50, 1])  #длина первые 50 первого типа сорта КАМА

    # print(X[70 :120 , 0]) #Площадь с 70 по 100 строку, тут описывается РОЗА

    # print(X[70 :120 , 1]) #длина с 70 по 100 строку, тут описывается РОЗ

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.3,random\_state=0)

perceptron = Perceptron(Learn\_Rate=0.0001, Iterations=20)

perceptron.fit(X\_train, y\_train)

y\_predict = perceptron.predict(X\_test)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_predict))

plt.plot(range(1, len(perceptron.errors) + 1), perceptron.errors, marker='o')

plt.xlabel('Эпохи')

plt.ylabel('Количество ошибочных классификаций')

plt.show()

class AdalineSGD(object):

    def \_\_init\_\_(self, eta=0.01, n\_iter=10, shuffle=True, random\_state=None):

        self.eta = eta

        self.n\_iter = n\_iter

        self.w\_initialized = False

        self.shuffle = shuffle

        self.random\_state = random\_state

    def fit(self, X, y):

        self.\_initialize\_weights(X.shape[1])

        self.cost\_ = []

        for i in range(self.n\_iter):

            if self.shuffle:

                X, y = self.\_shuffle(X, y)

            cost = []

            for xi, target in zip(X, y):

                cost.append(self.\_update\_weights(xi, target))

            avg\_cost = sum(cost) / len(y)

            self.cost\_.append(avg\_cost)

        return self

    def partial\_fit(self, X, y):

        """Fit training data without reinitializing the weights"""

        if not self.w\_initialized:

            self.\_initialize\_weights(X.shape[1])

        if y.ravel().shape[0] > 1:

            for xi, target in zip(X, y):

                self.\_update\_weights(xi, target)

        else:

            self.\_update\_weights(X, y)

        return self

    def \_shuffle(self, X, y):

        """Shuffle training data"""

        r = self.rgen.permutation(len(y))

        return X[r], y[r]

    def \_initialize\_weights(self, m):

        """Initialize weights to small random numbers"""

        self.rgen = np.random.RandomState(self.random\_state)

        self.w\_ = self.rgen.normal(loc=0.0, scale=0.01, size=1 + m)

        self.w\_initialized = True

    def \_update\_weights(self, xi, target):

        """Apply Adaline learning rule to update the weights"""

        output = self.activation(self.net\_input(xi))

        error = (target - output)

        self.w\_[1:] += self.eta \* xi.dot(error)

        self.w\_[0] += self.eta \* error

        cost = 0.5 \* error\*\*2

        return cost

    def net\_input(self, X):

        """Calculate net input"""

        return np.dot(X, self.w\_[1:]) + self.w\_[0]

    def activation(self, X):

        """Compute linear activation"""

        return X

    def predict(self, X):

        """Return class label after unit step"""

        return np.where(self.activation(self.net\_input(X)) >= 0.0, 1, -1)

import pandas as pd

import numpy as np

from matplotlib import pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from mlxtend.plotting import plot\_decision\_regions

url = "C:/Users/fox/Desktop/ii/data/seeds\_dataset.data"

# url = "F:/ii ЛАБЫ/data/seeds\_dataset.data"

names = ["площадь","периметр","компактность", "длина","ширина","асимметрия","длина канавки ядра","сорт"] #название атрибутов

target\_names = { 1:"Кама" , 2:"Роза" , 3:"Канадка" }

dataset = pd.read\_csv(url, names=names)

dataset.tail()

y = dataset.iloc[0:140,7].values

#тзменение классов для работы

y = np.where(y== 2, -1, 1)

# print(y)

X = dataset.iloc[0:140, [0, 3]].values

# стандартизация

X\_std = np.copy(X)

X\_std[:, 0] = (X[:, 0] - X[:, 0].mean()) / X[:, 0].std()

X\_std[:, 1] = (X[:, 1] - X[:, 1].mean()) / X[:, 1].std()

ada = AdalineSGD(n\_iter=10, eta=0.0001)

ada.fit(X\_std, y)

plot\_decision\_regions(X\_std, y, clf = ada)

plt.title('Adaline - градиентый спуск')

plt.xlabel('Площадь')

plt.ylabel('длина')

plt.legend(loc='upper left')

plt.tight\_layout()

plt.show()

plt.plot(range(1, len(ada.cost\_) + 1), ada.cost\_, marker='o')

plt.xlabel('Эпохи')

plt.ylabel('Сумма квадратов ошибок')

plt.tight\_layout()

plt.show()