Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №6**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «Классификация»

**Выполнил:**

Студент 3 курса

Группы ИИ-21

Пучинский А.А.

**Проверил:**

Туз И. С.

Брест 2023

**Цель:** классифицировать данные с помощью random forest и персептрона с одним скрытым слоем. Сравнить скорость, точность обучения моделей.

**Ход работы**

**Вариант 7**

Датасет: diabet

Код программы:

*from matplotlib import pyplot as plt*

*import numpy as np*

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

*from sklearn.metrics import accuracy\_score, classification\_report*

*from scipy.special import expit*

*import time*

*from math import isnan*

*E\_arr = []*

*class Perceptron:*

*def \_\_init\_\_(self, input\_size, hidden\_size, output\_size, learning\_rate=0.4):*

*self.input\_size = input\_size*

*self.hidden\_size = hidden\_size*

*self.output\_size = output\_size*

*self.learning\_rate = learning\_rate*

*self.start\_rate = learning\_rate*

*self.weights\_input\_hidden = np.random.randn(*

*self.input\_size, self.hidden\_size)*

*self.bias\_hidden = np.zeros((1, self.hidden\_size))*

*self.weights\_hidden\_output = np.random.randn(*

*self.hidden\_size, self.output\_size)*

*self.bias\_output = np.zeros((1, self.output\_size))*

*# print(self.weights\_input\_hidden)*

*# print(self.weights\_hidden\_output)*

*def sigmoid(self, x):*

*return 1/(1+np.exp(-x))*

*def sigmoid\_derivative(self, y):*

*return y\*(1-y)*

*def forward(self, inputs):*

*self.hidden\_input = np.dot(*

*inputs, self.weights\_input\_hidden) + self.bias\_hidden*

*self.hidden\_output = self.sigmoid(self.hidden\_input)*

*self.output = np.dot(self.hidden\_output,*

*self.weights\_hidden\_output) + self.bias\_output*

*return self.sigmoid(self.output)*

*def backward(self, inputs, target, output, isAdapt: bool = False):*

*error = target - output*

*deltaOutput = error \* self.sigmoid\_derivative(output)*

*errorHiddenLayer = deltaOutput.dot(self.weights\_hidden\_output.T)*

*deltaHiddenLayer = errorHiddenLayer \* \*

*self.sigmoid\_derivative(self.hidden\_output)*

*if isAdapt:*

*self.learning\_rate = (4\*np.sum(deltaOutput\*\*2 \* output \* (1-output)))/(*

*(1+np.sum(output\*\*2)) \* np.sum(deltaOutput\*\*2 \* output\*\*2 \* (1-output)\*\*2))/10 % 10*

*if isnan(self.learning\_rate):*

*self.learning\_rate = 0.4*

*# print(self.learning\_rate)*

*self.weights\_hidden\_output += self.hidden\_output.T.dot(*

*deltaOutput)\*self.learning\_rate*

*self.bias\_output += np.sum(deltaOutput, axis=0) \* self.learning\_rate*

*if isAdapt:*

*self.learning\_rate = (4\*np.sum(deltaHiddenLayer\*\*2 \* self.hidden\_output \* (1-self.hidden\_output)))/((1+np.sum(*

*self.hidden\_output\*\*2)) \* np.sum(deltaHiddenLayer\*\*2 \* self.hidden\_output\*\*2 \* (1-self.hidden\_output)\*\*2))/10 % 10*

*if isnan(self.learning\_rate):*

*self.learning\_rate = 0.4*

*self.weights\_input\_hidden += inputs.T.dot(*

*deltaHiddenLayer)\*self.learning\_rate*

*self.bias\_hidden += np.sum(deltaHiddenLayer,*

*axis=0) \* self.learning\_rate*

*def train(self, inputs, targets, epochs: int, isAdapt: bool = False):*

*global E\_arr*

*for epoch in range(epochs):*

*e\_arr = []*

*for i in range(len(inputs)):*

*input\_data = np.array([inputs[i]])*

*target\_data = np.array([targets[i]])*

*output = self.forward(input\_data)*

*e\_arr.append(target\_data - output)*

*self.backward(input\_data, target\_data, output, isAdapt)*

*E2 = np.sum(np.array(e\_arr)\*\*2)/2*

*E\_arr.append(E2)*

*def backwardBatch(self, inputs, targets, outputs, isAdapt: bool = False):*

*for i in range(len(outputs)):*

*error = targets[i] - outputs[i]*

*deltaOutput = np.array(error \* self.sigmoid\_derivative(outputs[i]))*

*errorHiddenLayer = deltaOutput.dot(self.weights\_hidden\_output.T)*

*deltaHiddenLayer = errorHiddenLayer \* \*

*self.sigmoid\_derivative(self.hidden\_output)*

*if isAdapt:*

*self.learning\_rate = (4\*np.sum(deltaOutput\*\*2 \* output \* (1-output)))/(*

*(1+np.sum(output\*\*2)) \* np.sum(deltaOutput\*\*2 \* output\*\*2 \* (1-output)\*\*2))/10 % 10*

*if isnan(self.learning\_rate):*

*self.learning\_rate = 0.4*

*s = self.hidden\_output.T.dot(*

*deltaOutput)\*self.learning\_rate*

*s = np.array([[s[0]], [s[1]]])*

*self.weights\_hidden\_output += s*

*self.bias\_output += np.sum(deltaOutput,*

*axis=0) \* self.learning\_rate*

*if isAdapt:*

*self.learning\_rate = (4\*np.sum(deltaHiddenLayer\*\*2 \* self.hidden\_output \* (1-self.hidden\_output)))/((1+np.sum(*

*self.hidden\_output\*\*2)) \* np.sum(deltaHiddenLayer\*\*2 \* self.hidden\_output\*\*2 \* (1-self.hidden\_output)\*\*2))/10 % 10*

*if isnan(self.learning\_rate):*

*self.learning\_rate = 0.4*

*self.weights\_input\_hidden += inputs[i].reshape(-1, 1).dot(*

*deltaHiddenLayer)\*self.learning\_rate*

*self.bias\_hidden += np.sum(deltaHiddenLayer,*

*axis=0) \* self.learning\_rate*

*def trainBatch(self, inputs, targets, epochs: int, batchsize: int, isAdapt: bool = False):*

*global E\_arr*

*if (len(inputs) % batchsize != 0):*

*print("Плохое значение пакета")*

*return ValueError*

*inputspack = [inputs[i-batchsize:i]*

*for i in range(batchsize, len(inputs), batchsize)]*

*targetspack = [targets[i-batchsize:i]*

*for i in range(batchsize, len(targets), batchsize)]*

*for epoch in range(epochs):*

*e\_arr = []*

*for i in range(len(inputspack)):*

*outputs = [self.forward(batchElem).item()*

*for batchElem in inputspack[i]]*

*for j in range(len(targetspack[i])):*

*e\_arr.append(targetspack[i][j]-outputs[j])*

*self.backwardBatch(inputspack[i], targetspack[i], outputs)*

*E2 = np.sum(np.array(e\_arr)\*\*2)/2*

*E\_arr.append(E2)*

*# print(f"Batch: Epoch: {epoch} MSE: {E2} LR: {self.learning\_rate}")*

*def predict(self, inputs):*

*output = self.forward(inputs)*

*return output*

*if \_\_name\_\_ == "\_\_main\_\_":*

*import csv*

*epochs = 100*

*X, Y = [], []*

*with open(r"C:\Users\mrlon\Рабочий стол\MRZIS\LAB6\diabetes.csv") as file:*

*reader = csv.reader(file)*

*for i, row in enumerate(reader):*

*X.append([float(value) for value in row[:-1]])*

*Y.append([float(row[-1])])*

*X = np.array(X)*

*Y = np.array(Y)*

*X /= 1000*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(*

*X, Y, test\_size=0.2, random\_state=42)*

*perceptron = Perceptron(8, 2, 1)*

*start = time.time()*

*perceptron.train(X\_train, y\_train, epochs)*

*end = time.time()*

*temp = perceptron.predict(X\_test)*

*print(temp)*

*y\_pred = (temp > 0.5).astype(int)*

*accuracy = accuracy\_score(y\_test, y\_pred)*

*print(f"Time taken: {(end-start):.03f}s")*

*print(f"Точность модели : {accuracy:.2f}")*

*print(f"Точность модели (MSE) : {min(E\_arr):.10f}")*

*plt.plot(range(epochs), E\_arr)*

*plt.show()*

*E\_arr.clear()*

*print(classification\_report(y\_test, y\_pred, zero\_division=1))*

*perceptron = Perceptron(8, 2, 1)*

*start = time.time()*

*perceptron.train(X\_train, y\_train, epochs, True)*

*end = time.time()*

*temp = perceptron.predict(X\_test)*

*y\_pred = (temp > 0.5).astype(int)*

*accuracy = accuracy\_score(y\_test, y\_pred)*

*print(f"Time taken: {(end-start):.03f}s")*

*print(f"Точность модели: {accuracy:.2f}")*

*print(f"Точность модели (MSE) : {min(E\_arr):.10f}")*

*plt.plot(range(epochs), E\_arr)*

*plt.show()*

*E\_arr.clear()*

*print(classification\_report(y\_test, y\_pred, zero\_division=1))*

*perceptron = Perceptron(8, 2, 1)*

*start = time.time()*

*perceptron.trainBatch(X\_train, y\_train, epochs, 2, False)*

*end = time.time()*

*temp = perceptron.predict(X\_test)*

*y\_pred = (temp > 0.5).astype(int)*

*accuracy = accuracy\_score(y\_test, y\_pred)*

*print(f"Time taken: {(end-start):.03f}s")*

*print(f"Точность модели: {accuracy:.2f}")*

*print(f"Точность модели (MSE) : {min(E\_arr):.10f}")*

*plt.plot(range(epochs), E\_arr)*

*plt.show()*

*E\_arr.clear()*

*print(classification\_report(y\_test, y\_pred, zero\_division=1))*

*perceptron = Perceptron(8, 2, 1)*

*start = time.time()*

*perceptron.trainBatch(X\_train, y\_train, epochs, 2, True)*

*end = time.time()*

*temp = perceptron.predict(X\_test)*

*y\_pred = (temp > 0.5).astype(int)*

*accuracy = accuracy\_score(y\_test, y\_pred)*

*print(f"Time taken: {(end-start):.03f}s")*

*print(f"Точность модели: {accuracy:.2f}")*

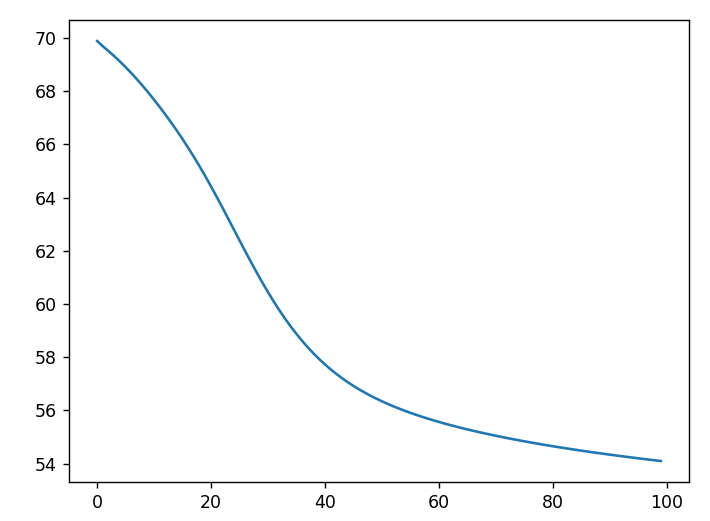
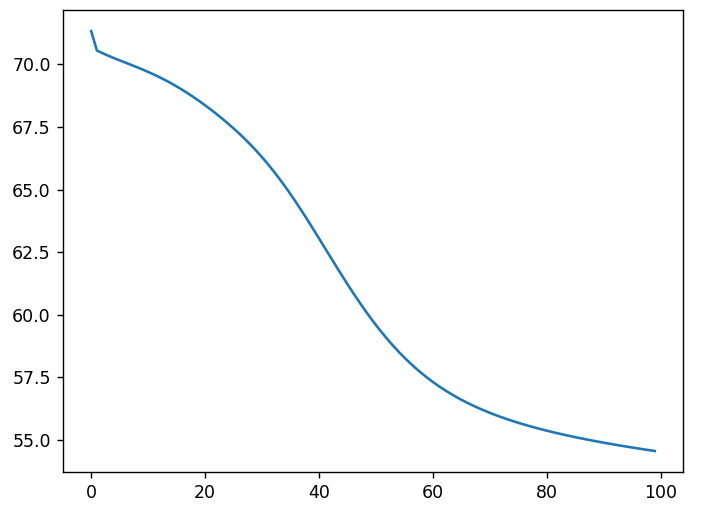
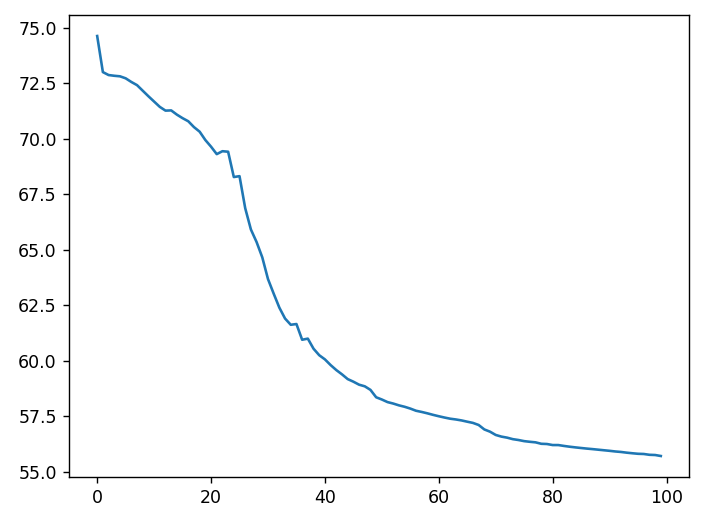
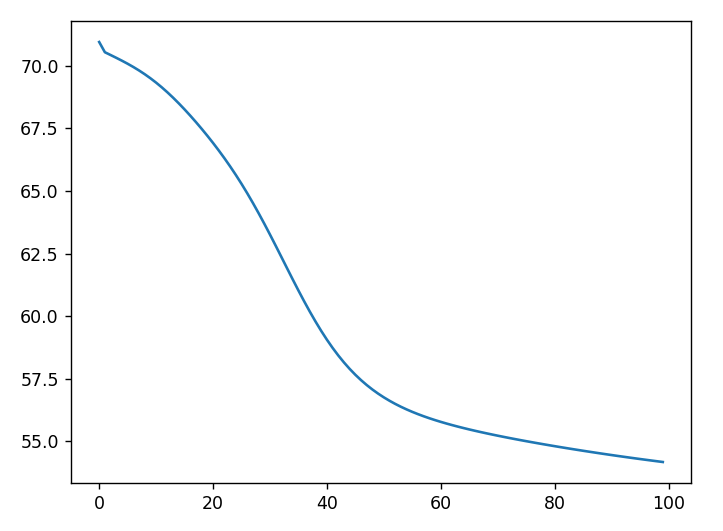
*print(f"Точность модели (MSE) : {min(E\_arr):.10f}")*

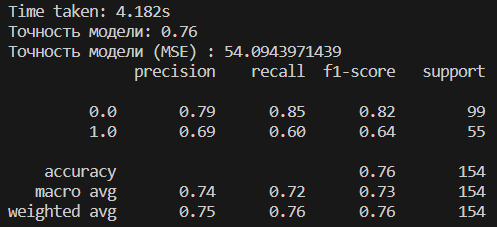
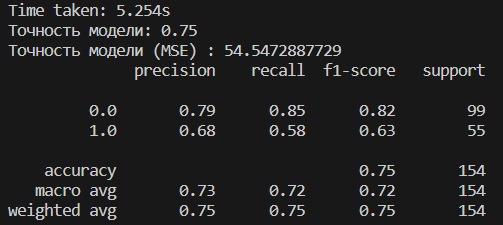
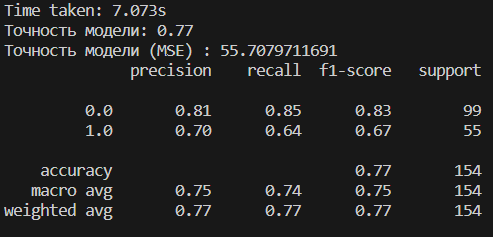
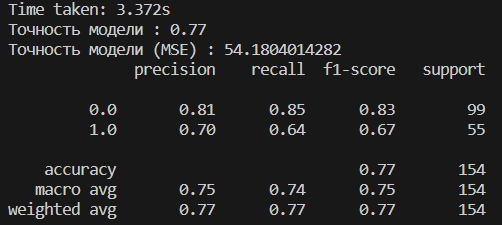
*plt.plot(range(epochs), E\_arr)*

*plt.show()*

*print(classification\_report(y\_test, y\_pred, zero\_division=1))*

Вывод программы:





**Вывод:** в ходе лабораторной работы я научился классифицировать данные с помощью алгоритма персептрона.