Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №3

По дисциплине «МРЗИС»

Тема: “Классификация”

Выполнил:

Студент 3 курса

Группы ИИ-21

Романко Н.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.

Код персептрона:

import pandas as pd

import numpy as np

from sklearn.metrics import classification\_report

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

class Layer\_Dense:

    def \_\_init\_\_(self, num\_inputs, num\_neurons):

        self.weight = 0.01 \* np.random.randn(num\_inputs, num\_neurons)

        self.biases = np.zeros((1, num\_neurons))

    def forward(self, inputs):

        self.inputs = inputs

        self.output = np.dot(inputs, self.weight) + self.biases

    def backward(self, dvalues):

        self.dweights = np.dot(self.inputs.T, dvalues)

        self.dbiases = np.sum(dvalues, axis=0, keepdims=True)

        self.dinputs = np.dot(dvalues, self.weight.T)

class Softmax:

    def forward(self, inputs):

        self.inputs = inputs

        exp\_values = np.exp(inputs - np.max(inputs, axis=1, keepdims=True))

        probabilities = exp\_values / np.sum(exp\_values, axis=1, keepdims=True)

        self.output = probabilities

    def backward(self, dvalues):

        self.dinputs = np.empty\_like(dvalues)

        for index, (single\_out, single\_dvalues) in enumerate(zip(self.output, dvalues)):

            single\_out = single\_out.reshape(-1, 1)

            jacobian\_matrix = np.diagflat(single\_out) - np.dot(single\_out, single\_out.T)

            self.dinputs[index] = np.dot(jacobian\_matrix, single\_dvalues)

class Optimazer\_Adam():

    def \_\_init\_\_(self, learning\_rate=0.001, decay = 0., epsilon = 1e-7, beta\_1 = 0.9, beta\_2 = 0.999):

        self.learning\_rate = learning\_rate

        self.current\_learning\_rate = learning\_rate

        self.decay = decay

        self.iterations = 0

        self.epsilon = epsilon

        self.beta\_1 = beta\_1

        self.beta\_2 = beta\_2

    def pre\_update\_params(self):

        if self.decay:

            self.current\_learning\_rate =self.learning\_rate \* (1. / (1. + self.decay \* self.iterations))

    def update\_params(self, layer):

        if not hasattr(layer, 'weight\_cache'):

            layer.weight\_momentums = np.zeros\_like(layer.weight)

            layer.weight\_cache = np.zeros\_like(layer.weight)

            layer.bias\_momentums = np.zeros\_like(layer.biases)

            layer.bias\_cache = np.zeros\_like(layer.biases)

        layer.weight\_momentums = self.beta\_1 \* layer.weight\_momentums + (1 - self.beta\_1) \* layer.dweights

        layer.bias\_momentums = self.beta\_1 \* layer.bias\_momentums + (1 - self.beta\_1) \* layer.dbiases

        weight\_momentums\_corrected = layer.weight\_momentums / (1 - self.beta\_1 \*\* (self.iterations + 1))

        bias\_momentums\_corrected = layer.bias\_momentums / (1 - self.beta\_1 \*\* (self.iterations + 1))

        layer.weight\_cache = self.beta\_2 \* layer.weight\_cache + (1 - self.beta\_2) \* layer.dweights \*\* 2

        layer.bias\_cache = self.beta\_2 \* layer.bias\_cache + (1 - self.beta\_2) \* layer.dbiases \*\* 2

        weight\_cache\_corrected = layer.weight\_cache / (1 - self.beta\_2 \*\* (self.iterations + 1))

        bias\_cache\_corrected = layer.bias\_cache / (1 - self.beta\_2 \*\* (self.iterations + 1))

        layer.weight += -self.current\_learning\_rate \* weight\_momentums\_corrected / (np.sqrt(weight\_cache\_corrected) + self.epsilon)

        layer.biases += -self.current\_learning\_rate \* bias\_momentums\_corrected / (np.sqrt(bias\_cache\_corrected) + self.epsilon)

    def post\_update\_params(self):

        self.iterations += 1

class Loss:

    def calc(self, output, y):

        sample\_losses = self.forward(output, y)

        return np.mean(sample\_losses)

class Loss\_CCE(Loss):

    def forward(self, y\_pred, y\_real):

        samples = len(y\_pred)

        y\_pred\_clipped = np.clip(y\_pred, 1e-7, 1-1e-7)

        if len(y\_real.shape) == 1:

            correct\_conf = y\_pred\_clipped[range(samples), y\_real]

        elif len(y\_real.shape) == 2:

            correct\_conf = np.sum(y\_pred\_clipped\*y\_real, axis=1)

        neg\_log\_likehoods = -np.log(correct\_conf)

        return neg\_log\_likehoods

    def backward(self, dvalues, y\_real):

        samples = len(dvalues)

        labels = len(dvalues[0])

        if len(y\_real.shape) == 1:

            y\_real = np.eye(labels)[y\_real]

        self.dinputs = -y\_real / dvalues

        self.dinputs = self.dinputs / samples

class Loss\_CCE\_and\_Softmax:

    def \_\_init\_\_(self):

        self.activation = Softmax()

        self.loss = Loss\_CCE()

    def forward(self, inputs, y\_true):

        self.activation.forward(inputs)

        self.output = self.activation.output

        return self.loss.calc(self.output, y\_true)

    def backward(self, dvalues, y\_true):

        samples = len(dvalues)

        if len(y\_true.shape) == 2:

            y\_true = np.argmax(y\_true, axis=1)

        self.dinputs = dvalues.copy()

        self.dinputs[range(samples), y\_true] -= 1

        self.dinputs = self.dinputs / samples

file = pd.read\_csv('MRZIS\\lab3\\WineQT.csv')

file = file[file['volatile acidity'] < 1]

file = file[file['sulphates'] < 1]

file = file[file['chlorides'] < 0.14]

file = file[file['free sulfur dioxide'] < 35]

file = file[(file['quality'] != 3) & (file['quality'] != 4) & (file['quality'] != 8)]

col = ['fixed acidity', 'volatile acidity', 'citric acid', 'chlorides', 'total sulfur dioxide', 'pH','sulphates', 'alcohol']

X = pd.DataFrame()

for i in col:

    X[i] = file[i]

Y = file['quality'].apply(lambda x: x - 5 if x>=5 else x)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2)

activation = Softmax()

#print(Y)

input\_layer = Layer\_Dense(len(col), 8)

hidden\_layer = Layer\_Dense(8, 8)

output\_layer = Layer\_Dense(8, 3)

loss\_func = Loss\_CCE\_and\_Softmax()

optimazer = Optimazer\_Adam(learning\_rate=0.12, decay=3e-5)

for epoch in range(1000):

    input\_layer.forward(X\_train)

    activation.forward(input\_layer.output)

    hidden\_layer.forward(activation.output)

    activation.forward(hidden\_layer.output)

    output\_layer.forward(activation.output)

    loss = loss\_func.forward(output\_layer.output, y\_train)

    predictions = np.argmax(loss\_func.output, axis=1)

    if len(y\_train.shape) == 2:

        y\_train = np.argmax(y\_train, axis=1)

    accuracy = np.mean(predictions==y\_train)

# print("Accuracy: ", accuracy)

    if not epoch % 100:

        print(f'epoch: {epoch}, ' + f'acc: {accuracy:.3f}, ' + f'loss: {loss:.3f}, ' + f'lr: {optimazer.current\_learning\_rate}')

    loss\_func.backward(loss\_func.output, y\_train)

    output\_layer.backward(loss\_func.dinputs)

    activation.backward(output\_layer.dinputs)

    hidden\_layer.backward(activation.dinputs)

    activation.backward(hidden\_layer.dinputs)

    input\_layer.backward(activation.dinputs)

    optimazer.pre\_update\_params()

    optimazer.update\_params(input\_layer)

    optimazer.update\_params(hidden\_layer)

    optimazer.update\_params(output\_layer)

    optimazer.post\_update\_params()

#validation

input\_layer.forward(X\_test)

activation.forward(input\_layer.output)

hidden\_layer.forward(activation.output)

activation.forward(hidden\_layer.output)

output\_layer.forward(activation.output)

loss = loss\_func.forward(output\_layer.output, y\_test)

predictions = np.argmax(loss\_func.output, axis=1)

if len(y\_test.shape) == 2:

    y\_test = np.argmax(y\_test, axis=1)

accuracy = np.mean(predictions==y\_test)

print(classification\_report(y\_test, predictions))

print(f'final acc: {accuracy:.3f}, ' + f'loss: {loss:.3f}')

Обучение и вывод метрик точности персептрона с одним скрытым слоем:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Код random forest:

import pandas as pd

from sklearn.metrics import classification\_report

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

from sklearn.ensemble import RandomForestClassifier

file = pd.read\_csv('MRZIS\\lab3\\WineQT.csv')

file = file[file['volatile acidity'] < 1]

file = file[file['sulphates'] < 1]

file = file[file['chlorides'] < 0.14]

file = file[file['free sulfur dioxide'] < 35]

file = file[(file['quality'] != 3) & (file['quality'] != 4) & (file['quality'] != 8)]

#print(file.describe())

col = ['fixed acidity', 'volatile acidity', 'citric acid', 'chlorides', 'total sulfur dioxide', 'pH','sulphates', 'alcohol']

X = pd.DataFrame()

for i in col:

    X[i] = file[i]

Y = file['quality']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2)

model = RandomForestClassifier(n\_estimators=100)

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

print(classification\_report(y\_test, model.predict(X\_test)))

Вывод метрик точности random forest:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Вывод: научился классифицировать данные при помощи персептрона со скрытым слоем и при помощи random forest.