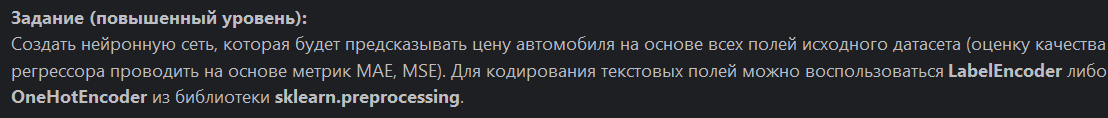
# Группа 5 Панфилов Валерий Александрович Лабораторная работа №3

«Создание нейронной сети, выполняющей предсказание непрерывной величины»

Задание: 

Код:

import numpy as np  
import pandas as pd  
import requests  
  
# url = "https://www.dropbox.com/s/bbm6rxqb4bsfl2d/training\_data.xlsx?dl=1"  
# r = requests.get(url, allow\_redirects=True)  
# open('training\_data.xlsx', 'wb').write(r.content)  
#  
#  
# url = "https://www.dropbox.com/s/gjhur7eyzcv265y/test\_data.xlsx?dl=1"  
# r = requests.get(url, allow\_redirects=True)  
# open('test\_data.xlsx', 'wb').write(r.content)  
  
  
# Чтение данных из файлов Excel  
training\_data = pd.read\_excel('D:/Valerian/Documents/OneDrive/Python/ДопОбр Анализ данных/НС лабы/Lab3/training\_data.xlsx',usecols=lambda x: 'Unnamed' not in x, na\_values=[''], keep\_default\_na=False)  
test\_data = pd.read\_excel('D:/Valerian/Documents/OneDrive/Python/ДопОбр Анализ данных/НС лабы/Lab3/test\_data.xlsx',usecols=lambda x: 'Unnamed' not in x, na\_values=[''], keep\_default\_na=False)  
# Заменяем NaN на 'None'  
training\_data = training\_data.fillna("0")  
test\_data = test\_data.fillna("0")  
  
# Определение категориальных и числовых признаков  
categorical\_features = [ 5, 6, 7, 8]  
numerical\_features = [1, 2, 3, 4, 9, 10, 11,12]  
  
def data\_preparation(data):  
 # Кодирование категориальных признаков  
 def one\_hot\_encode(data, categorical\_features):  
 categories = [list(set([row[i] for row in data])) for i in categorical\_features]  
 encoded\_data = []  
 for row in data:  
 encoded\_row = []  
 for i, feature in enumerate(row):  
 if i in categorical\_features:  
 one\_hot = [0] \* len(categories[categorical\_features.index(i)])  
 one\_hot[categories[categorical\_features.index(i)].index(feature)] = 1  
 encoded\_row.extend(one\_hot)  
 else:  
 encoded\_row.append(feature)  
 encoded\_data.append(encoded\_row)  
 return np.array(encoded\_data)  
  
 encoded\_data = one\_hot\_encode(data, categorical\_features)  
  
 # Разделение данных на признаки (X) и целевую переменную (y)  
 X = np.array(encoded\_data[:, 1:],dtype=float)  
 Y = np.array(encoded\_data[:, 0],dtype=float)  
  
  
 # Нормализация числовых признаков  
 for feature in numerical\_features:  
 idx = feature - 2 # сдвиг из-за удаления 'price' в X  
 mean = np.mean(X[:, idx].astype(float))  
 std = np.std(X[:, idx].astype(float))  
 X[:, idx] = (X[:, idx].astype(float) - mean) / std  
  
 # Разделение данных на тренировочную и тестовую выборки  
 def train\_test\_split(X, y, test\_size=0.2):  
 indices = np.random.permutation(X.shape[0])  
 test\_size = int(X.shape[0] \* test\_size)  
 test\_indices = indices[:test\_size]  
 train\_indices = indices[test\_size:]  
 return X[train\_indices], X[test\_indices], y[train\_indices], y[test\_indices]  
 return X,Y  
  
X\_train, Y\_train = data\_preparation(training\_data.values.tolist())  
print(X\_train[0])  
X\_test, Y\_test = data\_preparation(test\_data.values.tolist())  
# Добавление нового столбца к X\_test  
# Например, новый столбец с постоянным значением 1  
new\_column = np.ones((X\_test.shape[0], 1))  
  
# Добавление нового столбца к X\_test  
X\_test = np.hstack((X\_test, new\_column))  
print(X\_test[0])  
  
  
# Определение модели нейронной сети  
class NeuralNetwork:  
 def \_\_init\_\_(self, input\_size, hidden\_layer\_sizes, output\_size):  
 self.weights = []  
 self.biases = []  
 layer\_sizes = [input\_size] + hidden\_layer\_sizes + [output\_size]  
 for i in range(len(layer\_sizes) - 1):  
 self.weights.append(np.random.randn(layer\_sizes[i], layer\_sizes[i + 1]))  
 self.biases.append(np.zeros((1, layer\_sizes[i + 1])))  
  
 def relu(self, x):  
 return np.maximum(0, x)  
  
 def relu\_derivative(self, x):  
 return np.where(x > 0, 1, 0)  
  
 def forward(self, X):  
 activations = [X]  
 for i in range(len(self.weights)):  
 z = activations[-1].dot(self.weights[i]) + self.biases[i]  
 a = self.relu(z) if i < len(self.weights) - 1 else z # Linear activation in output layer  
 activations.append(a)  
 print(activations)  
 return activations  
  
 def backward(self, activations, y, learning\_rate):  
 m = y.shape[0]  
 deltas = [activations[-1] - y.reshape(-1, 1)]  
 for i in range(len(self.weights) - 1, 0, -1):  
 deltas.append(deltas[-1].dot(self.weights[i].T) \* self.relu\_derivative(activations[i]))  
 deltas.reverse()  
 for i in range(len(self.weights)):  
 self.weights[i] -= learning\_rate \* activations[i].T.dot(deltas[i]) / m  
 self.biases[i] -= learning\_rate \* np.sum(deltas[i], axis=0, keepdims=True) / m  
  
 def train(self, X, y, epochs, learning\_rate):  
 for epoch in range(epochs):  
 activations = self.forward(X)  
 self.backward(activations, y, learning\_rate)  
 if epoch % 100 == 0:  
 loss = np.mean((activations[-1] - y.reshape(-1, 1)) \*\* 2)  
 print(f'Epoch {epoch}, Loss: {loss}')  
  
 def predict(self, X):  
 return self.forward(X)[-1]  
  
# Инициализация и обучение модели  
input\_size = X\_train.shape[1]  
hidden\_layer\_sizes = [64, 32]  
output\_size = 1  
  
nn = NeuralNetwork(input\_size, hidden\_layer\_sizes, output\_size)  
nn.train(X\_train, Y\_train, epochs=1000, learning\_rate=0.01)  
  
# Оценка модели на тестовых данных  
Y\_pred = nn.predict(X\_test)  
  
mae = np.mean(np.abs(Y\_pred - Y\_test.reshape(-1, 1)))  
mse = np.mean((Y\_pred - Y\_test.reshape(-1, 1)) \*\* 2)  
  
print(f'Test MAE: {mae}')  
print(f'Test MSE: {mse}')

Результат выполнения:

