Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ульяновский государственный технический университет»

Кафедра: «Вычислительная техника»

Дисциплина: «Системы искусственного интеллекта»

Лабораторная работа №4

Основы нейронных сетей

Вариант 20

Выполнил

Студент группы ИВТАСбд-32

Хасанов И. Н.

Проверил:

преподаватель кафедры «ВТ»

Хайруллин И. Д.

Ульяновск 2024

**Оглавление**

[Постановка задачи 3](#_Toc167463914)

[Реализация 4](#_Toc167463915)

[Вывод 8](#_Toc167463916)

[Приложение. Source.cpp 9](#_Toc167463917)

# Постановка задачи

1. Написать программу, которая разделяет исходную выборку на обучающую и тестовую (training set, validation set, test set), если такое разделение не предусмотрено предложенным набором данных.
2. Произвести масштабирование признаков (scaling).
3. С использованием библиотеки scikit-learn обучить 2 модели нейронной сети (Perceptron и MLPClassifier) по обучающей выборке. Перед обучением необходимо осуществить масштабирование признаков. Пример MLPClassifier Пример и описание Perceptron
4. Проверить точность модели по тестовой выборке.
5. Провести эксперименты и определить наилучшие параметры коэффициента обучения, параметра регуляризации, функции оптимизации. Данные экспериментов необходимо представить в отчете (графики, ход проведения эксперимента, выводы).

Вариант: Pen-Based Recognition of Handwritten Digits.

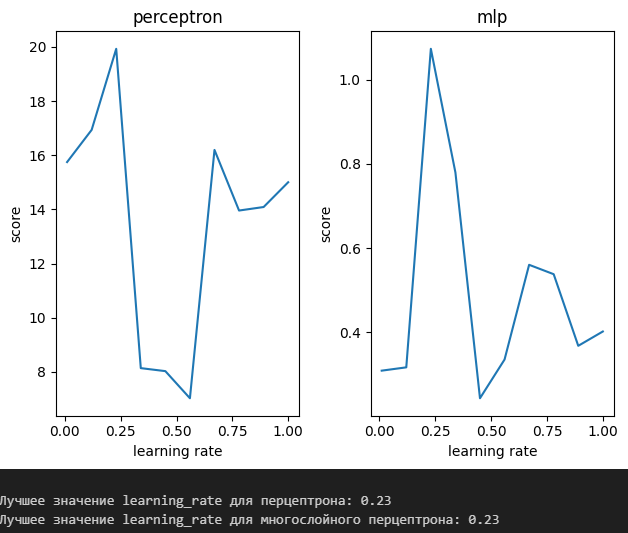
# Реализация

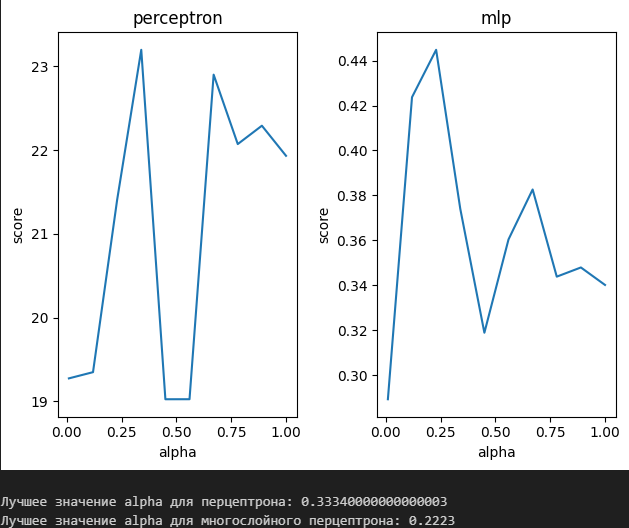
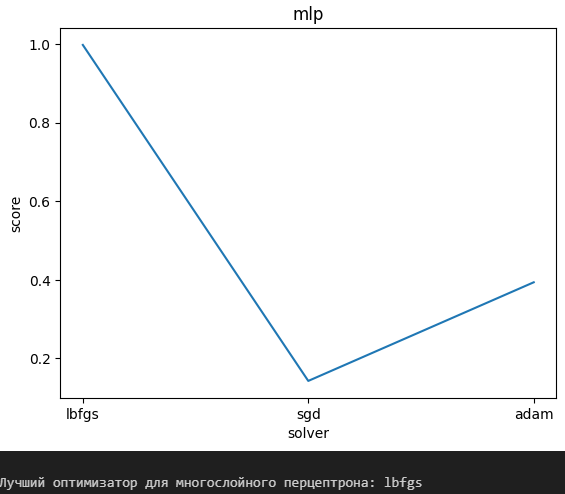
Код написан на языке Python. Вначале считываются данные из файла и записываются в датафрейм. После производится масштабирование признаков с помощью preprocessing.StandardScaler(). Масштабирование производится с помощью стандартизации - математическое ожидание каждой колонки становится равным 0, а его стандратное отклонение становится равным 1.

С использованием библиотеки scikit-learn обучаются 2 модели нейронной сети (Perceptron и MLPClassifier) по обучающей выборке. Perceptron и MLPClassifier обучаются с использованием одноименных функций. Точность моделей находится при помощи accuracy\_score().

Эффективность работы моделей вычисляется как: точность модели / время обучения.

Далее строятся находятся лучшие значения шага обучения, параметра регуляризации моделей и лучший оптимизатор для MLP.



# Вывод

Проделав данную лабораторную работу, были изучены методы построения моделей нейронных сетей.

# Приложение. Source.cpp

|  |
| --- |
| import pandas as pd  import numpy as np  from matplotlib import pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn import preprocessing  from sklearn.linear\_model import Perceptron  from sklearn.metrics import accuracy\_score  from sklearn.neural\_network import MLPClassifier  from sklearn.linear\_model import SGDClassifier  import time  from mpl\_toolkits.mplot3d import Axes3D df\_test = pd.read\_csv('pendigits.tes', header=None)  df\_train = pd.read\_csv('pendigits.tra', header=None)  df = pd.concat([df\_train, df\_test]) x\_train = df\_train.iloc[:, :-1]  y\_train = df\_train.drop(df.iloc[:, :-1], axis=1)  x\_test = df\_test.iloc[:, :-1]  y\_test = df\_test.drop(df.iloc[:, :-1], axis=1)  x\_data = df.iloc[:, :-1]  y\_data = df.drop(df.iloc[:, :-1], axis=1)  scaler\_x = preprocessing.StandardScaler()  x\_data = pd.DataFrame(scaler\_x.fit\_transform(x\_data), columns=x\_data.columns)  x\_train = pd.DataFrame(scaler\_x.fit\_transform(x\_train), columns=x\_train.columns)  x\_test = pd.DataFrame(scaler\_x.fit\_transform(x\_test), columns=x\_test.columns) print('\nМатематическое ожидание для каждой из колонок x\_data:')  print(x\_data.mean())  print('\nСтандартное отклонение для каждой из колонок x\_data:')  print(x\_data.std())x\_train = np.array(x\_train)  y\_train = np.array(y\_train)  x\_test = np.array(x\_test)  y\_test = np.array(y\_test) perceptron = Perceptron(tol=1e-3, random\_state=0, verbose=True)  perceptron.fit(x\_train, y\_train.ravel())  perceptron\_y\_pedict = perceptron.predict(x\_test) mlp = MLPClassifier(random\_state=1, verbose=True)  mlp.fit(x\_train, y\_train.ravel())  mlp\_y\_pedict = mlp.predict(x\_test) print(f'Точность перцептрона: {accuracy\_score(y\_test, perceptron\_y\_pedict)}')  print(f'Точность многослойного перцептрона: {accuracy\_score(y\_test, mlp\_y\_pedict)}') # Эффективность работы перцептрона  def get\_p\_score(learning\_rate, alpha):  start\_time = time.time()  perceptron = SGDClassifier(loss='perceptron', learning\_rate='constant', eta0=learning\_rate, alpha=alpha, max\_iter=1000, tol=1e-3) # Тоже самое, что и перцептрон  perceptron.fit(x\_train, y\_train.ravel())  end\_time = time.time()  predict = perceptron.predict(x\_test)  elapsed\_time = end\_time - start\_time  return accuracy\_score(y\_test, predict) / elapsed\_time  # Эффектиность работы многослойного перцептрона  def get\_mlp\_score(learning\_rate, alpha, optimizer):  start\_time = time.time()  mlp = MLPClassifier(learning\_rate='constant', learning\_rate\_init=learning\_rate, alpha=alpha, solver=optimizer, max\_iter=1000, tol=1e-3) # Тоже самое, что и перцептрон  mlp.fit(x\_train, y\_train.ravel())  end\_time = time.time()  predict = mlp.predict(x\_test)  elapsed\_time = end\_time - start\_time  return accuracy\_score(y\_test, predict) / elapsed\_time  def to\_test(learning\_rate\_list, alpha\_list, optimizer):  perceptron\_score = []  mlp\_score = []  for id in range(len(learning\_rate\_list)):  perceptron\_score.append(get\_p\_score(learning\_rate\_list[id], alpha\_list[id]))  mlp\_score.append(get\_mlp\_score(learning\_rate\_list[id], alpha\_list[id], optimizer[id]))    perceptron\_score = np.array(perceptron\_score)  mlp\_score = np.array(mlp\_score)    return perceptron\_score, mlp\_score  def draw\_test(x, perceptron\_score, mlp\_score, x\_label):  fig, axs = plt.subplots(1, 2)  axs[0].plot(x, perceptron\_score)  axs[0].set\_title('perceptron')  axs[0].set(xlabel=x\_label, ylabel='score')  axs[1].plot(x, mlp\_score)  axs[1].set\_title('mlp')  axs[1].set(xlabel=x\_label, ylabel='score')  fig.tight\_layout()  plt.show()  count\_tests = 10  learning\_rate\_list = np.linspace(0.01, 1, count\_tests)  perceptron\_score, mlp\_score = to\_test(learning\_rate\_list, np.full(count\_tests, 0.0001), np.full(count\_tests, 'adam'))  draw\_test(learning\_rate\_list, perceptron\_score, mlp\_score, 'learning rate')  print(f'Лучшее значение learning\_rate для перцептрона: {learning\_rate\_list[np.argmax(perceptron\_score)]}')  print(f'Лучшее значение learning\_rate для многослойного перцептрона: {learning\_rate\_list[np.argmax(mlp\_score)]}') alpha\_list = np.linspace(0.0001, 1, count\_tests)  perceptron\_score, mlp\_score = to\_test(np.full(count\_tests, 0.001), alpha\_list, np.full(count\_tests, 'adam'))  draw\_test(learning\_rate\_list, perceptron\_score, mlp\_score, 'alpha')  print(f'Лучшее значение alpha для перцептрона: {alpha\_list[np.argmax(perceptron\_score)]}')  print(f'Лучшее значение alpha для многослойного перцептрона: {alpha\_list[np.argmax(mlp\_score)]}') optimizers\_list = np.array(['lbfgs', 'sgd', 'adam'])  perceptron\_score, mlp\_score = to\_test(np.full(len(optimizers\_list), 0.001), np.full(len(optimizers\_list), 0.0001), optimizers\_list)  plt.plot(optimizers\_list, mlp\_score)  plt.title('mlp')  plt.xlabel('solver')  plt.ylabel('score')  plt.show()  print(f'Лучший оптимизатор для многослойного перцептрона: {optimizers\_list[np.argmax(mlp\_score)]}') |