Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

**Лабораторная работа №4**

Дисциплина: «Системы искусственного интеллекта»

«Основы нейронных сетей»

Выполнил студент

группы ИВТАПбд-41

Меховников Е.А.

Проверил:

ассистент кафедры «ВТ»

Хайруллин И. Д.

Ульяновск, 2024

**Цель работы**

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).

**Ход работы**

Данная нейронная сеть предназначена для распознавания букв на изображениях. Она обучается на наборе данных, содержащем изображения букв, и после обучения способна определить, какая буква изображена на новом изображении.

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.

В первую очередь были определены параметры, которые были использованы для обучения, из letter-recognition.data - ['letter', 'x-box', 'y-box', 'width', 'high', 'onpix', 'x-bar', 'y-bar', 'x2bar', 'y2bar', 'xybar', 'x2ybr', 'xy2br', 'x-ege', 'xegvy', 'y-ege', 'yegvx'].

Функция ‘train\_test\_split’ из библиотеки ‘sklearn’ используется для разделения данных на обучающую и тестовую выборки. В этой работе эта функция нужна для разделения данных на признаки и целевые переменные, тестовые и обучающие выборки (Листинг 1).

Листинг 1.

|  |
| --- |
| # Разделение на признаки (X) и целевую переменную (y)  X = data.drop('letter', axis=1)  # Признаки, все столбцы кроме 'letter'  y = data['letter']  # Целевая переменная, столбец 'letter'  # Разделение на обучающую и тестовую выборки  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # random\_state определяет начальное состояние генератора псевдослучайных чисел  # Проверим размерности полученных выборок  print("Размер обучающей выборки (X\_train):", X\_train.shape) #возвращает кортеж, содержащий количество строк и столбцов в DF  print("Размер тестовой выборки (X\_test):", X\_test.shape)  print("Размер обучающей выборки (y\_train):", y\_train.shape)  print("Размер тестовой выборки (y\_test):", y\_test.shape)  Размер обучающей выборки (X\_train): (16000, 16)  Размер тестовой выборки (X\_test): (4000, 16)  Размер обучающей выборки (y\_train): (16000,)  Размер тестовой выборки (y\_test): (4000,) |

2. Произвести масштабирование признаков (scaling).

Масштабирование признаков – это ключевой этап предобработки данных, который позволяет привести значения признаков в наборе данных к единому масштабу или распределению. (Листинг 2)

Листинг 2.

|  |
| --- |
| # Создаем объект StandardScaler  scaler = StandardScaler()  # Масштабируем признаки обучающей выборки  X\_train\_scaled = scaler.fit\_transform(X\_train)  # Применяем ту же самую стандартизацию к признакам тестовой выборки  X\_test\_scaled = scaler.transform(X\_test) |

3. С использованием библиотеки scikit-learn обучить 2 модели нейронной сети (Perceptron и MLPClassifier) по обучающей выборке. Перед обучением необходимо осуществить масштабирование признаков.

Модели нейронной сети задаются одинаково с помощью model.fit(), функция на вход получает обучающие данные (X) и метки (y). (Листинг 3).

Листинг 3.

|  |
| --- |
| # Обучение модели Perceptron  perceptron\_model = Perceptron(random\_state=42)  perceptron\_model.fit(X\_train\_scaled, y\_train)  # Обучение модели MLPClassifier  mlp\_model = MLPClassifier(random\_state=42)  mlp\_model.fit(X\_train\_scaled, y\_train)  # Предсказание на тестовых данных  y\_pred\_perceptron = perceptron\_model.predict(X\_test\_scaled)  y\_pred\_mlp = mlp\_model.predict(X\_test\_scaled)  # Оценка точности моделей  accuracy\_perceptron = accuracy\_score(y\_test, y\_pred\_perceptron)  accuracy\_mlp = accuracy\_score(y\_test, y\_pred\_mlp)  print("Точность модели Perceptron на тестовой выборке:", accuracy\_perceptron)  print("Точность модели MLPClassifier на тестовой выборке:", accuracy\_mlp) |

Perceptron – это простейший вид нейрона, базовый элемент в искусственных нейронных сетях.

Каждому входному значению в перцептроне присваивается вес на который умножается его значение, затем значения складываются и в зависимости от функции активации на выходе получается определённый результат (Рис. 1).

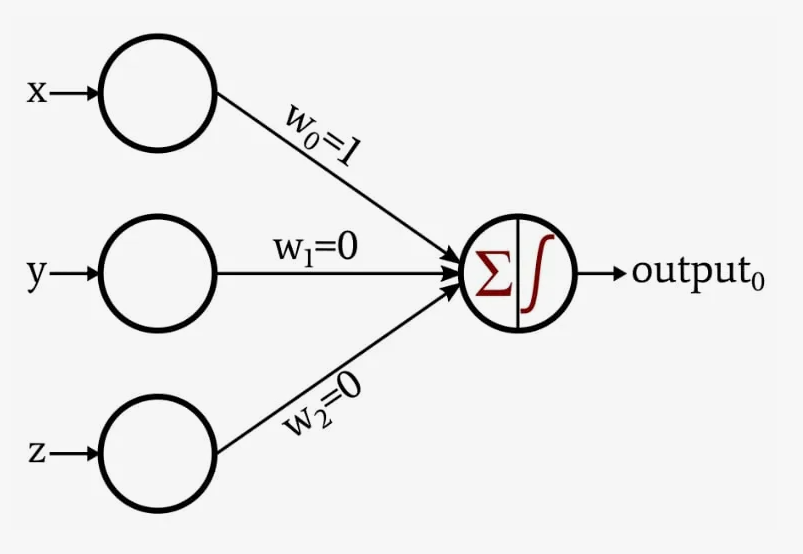


Рис. 1 – Perceptron

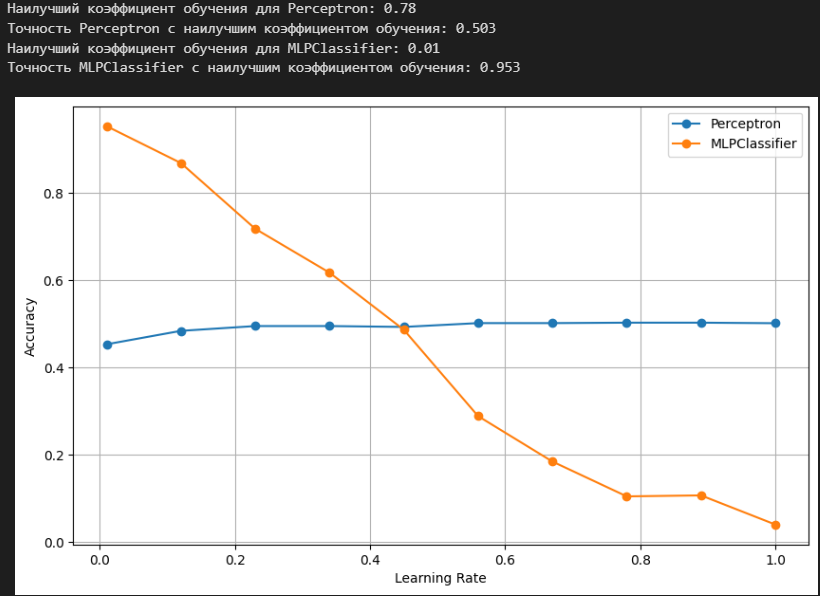
Многослойный персептрон (MLPClassifier) представляет собой тип искусственной нейронной сети прямого распространения, состоящий из нескольких слоёв нейронов. Он включает входной слой, один или несколько скрытых слоев и выходной слой, что позволяет моделировать сложные взаимосвязи между входными и выходными данными.

4. Проверить точность модели по тестовой выборке

Благодаря наличию нескольких слоев и нелинейных функций активации, MLPClassifier может создавать нелинейные зависимости между входными и выходными данными. Это позволяет моделировать более сложные взаимосвязи, что приводит к более точным результатам чем получились в Perceptron. (Рис. 2).

Рис. 2 – Результаты моделирования

. 5. Провести эксперименты и определить наилучшие параметры коэффициента обучения, параметра регуляризации, функции оптимизации. (Рис. 4)

Рис. 4 – Результаты эксперимента

Был проведён эксперимент по нахождению лучшего оптимизатора для данного набора данных, в зависимости от скорости обучения разные оптимизации выдавали неодинаковые результаты (Рис. 5).

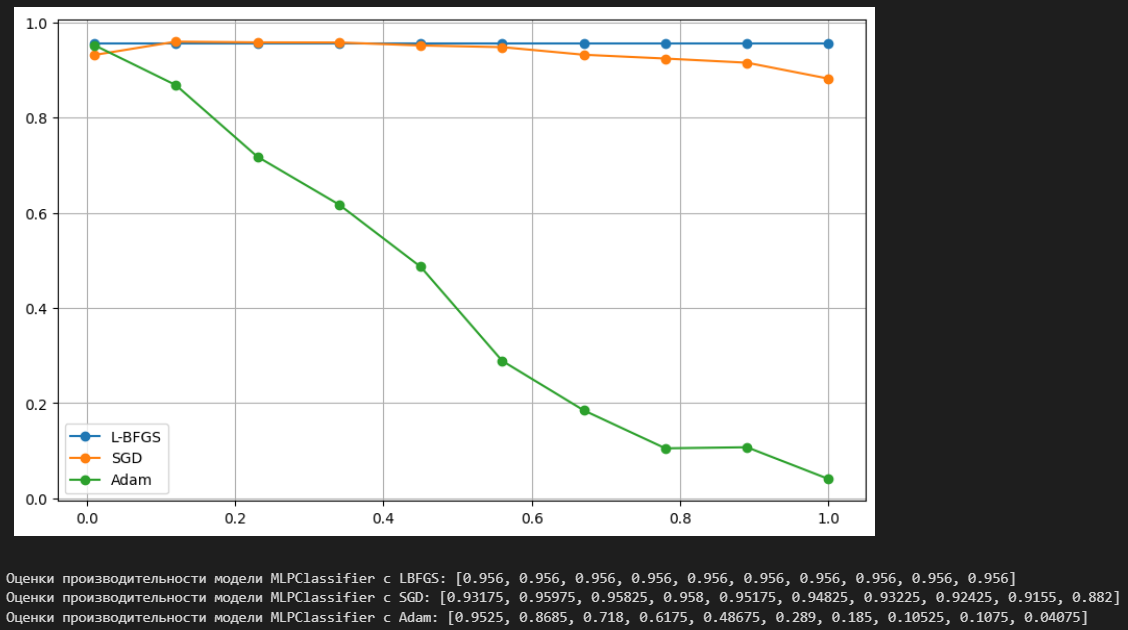


Рис. 4 – Результаты эксперимента по нахождению лучшего оптимизатора

LBFGS - это алгоритм оптимизации, который использует ограниченную память для хранения информации о предыдущих итерациях и их градиентах. Это делает его эффективным для работы с большими объемами данных или сложными моделями с большим количеством параметров.

SGD (Stochastic Gradient Descent) - простой и широко используемый алгоритм, который обновляет параметры модели на основе градиента, рассчитанного на случайном подмножестве данных. Этот метод хорошо подходит для работы с большими наборами данных и высокоразмерными моделями.

Adam - адаптивный алгоритм, который сочетает в себе идеи SGD Momentum с использованием моментов градиента для обновления параметров, а также скользящий средний квадрат градиентов из RMSprop. Это делает его эффективным для широкого спектра задач и может ускорить сходимость модели.

Каждый из этих алгоритмов оптимизации обладает своими преимуществами и недостатками, и выбор подходящего алгоритма зависит от конкретной задачи и характеристик данных.

**Вывод**

В ходе лабораторной работы была реализована задача классификации данных. Для подготовки данных была проведена предварительная обработка, включающая масштабирование признаков с помощью StandardScaler. Далее были обучены две модели нейронной сети: Perceptron и MLPClassifier, используя библиотеку sklearn. Обучение моделей проводилось на обучающей выборке, предварительно масштабированной. Затем была проверена точность полученных моделей на тестовой выборке.

Также были проведены эксперименты с целью оптимизации параметров моделей, таких как коэффициент обучения, параметры регуляризации и алгоритмы оптимизации..