МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

**Системы искусственного интеллекта**

**Отчет по выполнению лабораторной работы №4**

Выполнил

студент группы ИВТАПбд-31

Гаязов Т. Р.

Проверил:

преподаватель

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.
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**

1. Загрузка и предобработка данных

Загрузка данных из файла abalone.data в DataFrame с помощью библиотеки pandas.

Преобразование категориального признака 'class' в набор бинарных признаков с использованием pd.drop на рисунке 1.

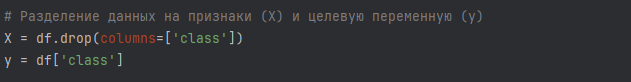


Рис. 1. Данные

1. Разделение данных на обучающую и тестовую выборки

Разделение данных на признаки (X) и целевую переменную (y).

Разделение данных на обучающую и тестовую выборки с использованием функции train\_test\_split из библиотеки scikit-learn на рисунке 2.

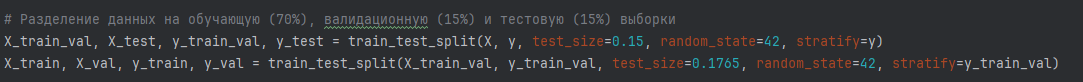


Рис. 2. Разделение данных

1. Масштабирование данных

Масштабирование признаков с использованием StandardScaler для приведения данных к стандартному нормальному распределению на рисунке 3.

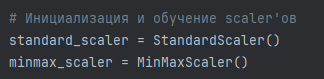


Рис. 3. Масштабирование данных

1. Обучение моделей

Обучение модели Персептрона (Perceptron).

Обучение модели многослойного персептрона (MLPClassifier) с различными параметрами на рисунке 4.

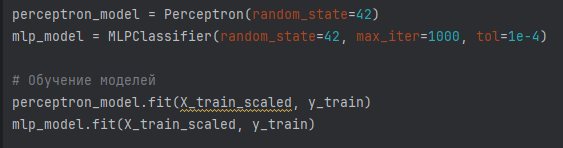


Рис. 4. Обучение моделей

1. Оценка моделей

Предсказание значений целевой переменной на тестовой выборке для каждой модели. Вычисление точности предсказаний с использованием метрики accuracy\_score на рисунке 5.

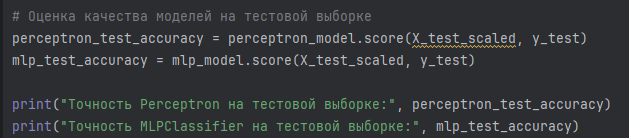


Рис. 5. Вычисление точности

**Тестирование**

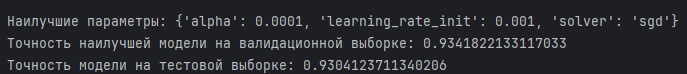


Рис.6. Точности моделей

**Заключение**

В процессе выполнения лабораторной работы проводилось обучение моделей с различными методами на обучающей выборке, а их точность проверялась на тестовой выборке. Были построены и протестированы несколько моделей машинного обучения для задачи классификации данных о моллюсках абалон (abalone).

Сначала была обучена модель персептрона, а затем многослойные персептроны (MLP) с различными алгоритмами оптимизации, включая Adam, SGD и L-BFGS. Для каждой модели была оценена точность предсказаний на тестовой выборке.

Результаты показали, что:

Модель персептрона продемонстрировала определенный уровень точности, но не была лидером среди тестируемых моделей.

Многослойные персептроны с различными параметрами и алгоритмами оптимизации показали лучшую производительность. Наибольшую точность среди них продемонстрировал MLP с алгоритмом Adam.**Список литературы**

1. <https://scikit-learn.org/stable/>
2. <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html>
3. <https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html>

**Приложение**

|  |
| --- |
| from scipy.io import arff import pandas as pd from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler, MinMaxScaler from sklearn.linear\_model import Perceptron from sklearn.neural\_network import MLPClassifier from sklearn.model\_selection import GridSearchCV  # Чтение файла ARFF и загрузка данных data, meta = arff.loadarff('seismic-bumps.arff')  # Преобразование данных в DataFrame df = pd.DataFrame(data)  # Разделение данных на признаки (X) и целевую переменную (y) X = df.drop(columns=['class']) y = df['class']  # Разделение данных на обучающую (70%), валидационную (15%) и тестовую (15%) выборки X\_train\_val, X\_test, y\_train\_val, y\_test = train\_test\_split(X, y, test\_size=0.15, random\_state=42, stratify=y) X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train\_val, y\_train\_val, test\_size=0.1765, random\_state=42, stratify=y\_train\_val)  # Сохранение обучающей выборки в CSV файл Xy\_train = pd.concat([X\_train, y\_train], axis=1) Xy\_train.to\_csv('train\_data.csv', index=False)  # Сохранение валидационной выборки в CSV файл Xy\_val = pd.concat([X\_val, y\_val], axis=1) Xy\_val.to\_csv('val\_data.csv', index=False)  # Сохранение тестовой выборки в CSV файл Xy\_test = pd.concat([X\_test, y\_test], axis=1) Xy\_test.to\_csv('test\_data.csv', index=False)  print("Выборки сохранены в CSV файлы.")  # Загрузка данных из CSV файлов train\_data = pd.read\_csv('train\_data.csv') val\_data = pd.read\_csv('val\_data.csv') test\_data = pd.read\_csv('test\_data.csv')  # Разделение числовых признаков и целевой переменной numerical\_features = ['genergy', 'gpuls', 'gdenergy', 'gdpuls', 'nbumps', 'nbumps2', 'nbumps3',   'nbumps4', 'nbumps5', 'nbumps6', 'nbumps7', 'nbumps89', 'energy', 'maxenergy']  X\_train = train\_data[numerical\_features] y\_train = train\_data['class']  X\_val = val\_data[numerical\_features] y\_val = val\_data['class']  X\_test = test\_data[numerical\_features] y\_test = test\_data['class']  # Инициализация и обучение scaler'ов standard\_scaler = StandardScaler() minmax\_scaler = MinMaxScaler()  # Проверка наличия данных перед масштабированием if not X\_train.empty:  # Стандартизация признаков  X\_train\_scaled = standard\_scaler.fit\_transform(X\_train)  # Нормализация признаков  X\_train\_normalized = minmax\_scaler.fit\_transform(X\_train)  print("Масштабирование числовых признаков обучающей выборки выполнено.")  print("Масштабированные числовые признаки обучающей выборки (стандартизация):")  print(X\_train\_scaled)  print("\nНормализованные числовые признаки обучающей выборки:")  print(X\_train\_normalized) else:  print("Ошибка: Не удалось найти данные для масштабирования числовых признаков обучающей выборки.")  if not X\_val.empty:  # Стандартизация признаков  X\_val\_scaled = standard\_scaler.transform(X\_val)  # Нормализация признаков  X\_val\_normalized = minmax\_scaler.transform(X\_val)  print("Масштабирование числовых признаков валидационной выборки выполнено.")  print("\nМасштабированные числовые признаки валидационной выборки (стандартизация):")  print(X\_val\_scaled)  print("\nНормализованные числовые признаки валидационной выборки:")  print(X\_val\_normalized) else:  print("Ошибка: Не удалось найти данные для масштабирования числовых признаков валидационной выборки.")  if not X\_test.empty:  # Стандартизация признаков  X\_test\_scaled = standard\_scaler.transform(X\_test)  # Нормализация признаков  X\_test\_normalized = minmax\_scaler.transform(X\_test)  print("Масштабирование числовых признаков тестовой выборки выполнено.")  print("\nМасштабированные числовые признаки тестовой выборки (стандартизация):")  print(X\_test\_scaled)  print("\nНормализованные числовые признаки тестовой выборки:")  print(X\_test\_normalized) else:  print("Ошибка: Не удалось найти данные для масштабирования числовых признаков тестовой выборки.")  # Инициализация моделей с увеличенным количеством итераций perceptron\_model = Perceptron(random\_state=42) mlp\_model = MLPClassifier(random\_state=42, max\_iter=1000, tol=1e-4)  # Обучение моделей perceptron\_model.fit(X\_train\_scaled, y\_train) mlp\_model.fit(X\_train\_scaled, y\_train)  print("Модели нейронной сети обучены успешно.")  # Оценка качества моделей на тестовой выборке perceptron\_test\_accuracy = perceptron\_model.score(X\_test\_scaled, y\_test) mlp\_test\_accuracy = mlp\_model.score(X\_test\_scaled, y\_test)  print("Точность Perceptron на тестовой выборке:", perceptron\_test\_accuracy) print("Точность MLPClassifier на тестовой выборке:", mlp\_test\_accuracy)  param\_grid = {  'learning\_rate\_init': [0.001, 0.01, 0.1],  'alpha': [0.0001, 0.001, 0.01],  'solver': ['adam', 'sgd', 'lbfgs'] }  # Создаем экземпляр модели MLPClassifier mlp = MLPClassifier(random\_state=42, max\_iter=2000)  # Инициализируем GridSearchCV для перебора параметров grid\_search = GridSearchCV(mlp, param\_grid, cv=5, scoring='accuracy')  # Обучаем GridSearchCV на обучающей выборке grid\_search.fit(X\_train\_scaled, y\_train)  # Получаем наилучшие параметры и точность на валидационной выборке best\_params = grid\_search.best\_params\_ best\_accuracy = grid\_search.best\_score\_  print("Наилучшие параметры:", best\_params) print("Точность наилучшей модели на валидационной выборке:", best\_accuracy)  # Используем наилучшие параметры для обучения модели на всей обучающей выборке best\_mlp = MLPClassifier(random\_state=42, max\_iter=2000, \*\*best\_params) best\_mlp.fit(X\_train\_scaled, y\_train)  # Оцениваем качество модели на тестовой выборке test\_accuracy = best\_mlp.score(X\_test\_scaled, y\_test) print("Точность модели на тестовой выборке:", test\_accuracy) |