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»

Выполнил:

студент группы ИВТАПбд-41

Сокольский Р.С

Проверил:

преподаватель кафедры «ВТ»

Святов К.В.

Ульяновск, 2025

Оглавление

[1. Постановка задачи 3](#_Toc210580418)

[2. Теоретические данные 4](#_Toc210580419)

[3. Описание набора данных 6](#_Toc210580420)

[4. Реализация 8](#_Toc210580421)

[5. Тестирование 13](#_Toc210580422)

[6. Выводы 14](#_Toc210580423)

# **1. Постановка задачи**

Перед выполнением лабораторной работы необходимо загрузить набор данных в соответствии с вариантом на диск

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).

Вариант 22: Indoor User Movement Prediction from RSS data

# **2. Теоретические данные**

**Perceptron** — это базовая модель искусственной нейронной сети, представляющая собой однослойный классификатор. Он принимает входные данные, перемножает их на веса, суммирует и пропускает через активационную функцию для получения окончательного результата. Принцип работы перцептрона хорошо иллюстрирует основы обучения с корректировкой весов, что позволяет ему выполнять задачи классификации. В данной работе Perceptron использовался с L2-регуляризацией и фиксированным значением коэффициента обучения (eta0=0.1), что демонстрирует принцип балансировки между переобучением и недостаточным обучением.

**MLPClassifier (Multi-Layer Perceptron Classifier)** — это многослойная нейронная сеть, способная решать более сложные задачи классификации за счёт наличия скрытых слоёв. В классическом представлении MLP включает входной, один или несколько скрытых слоёв и выходной слой. Обучение сети происходит методом обратного распространения ошибки, что позволяет корректировать веса на каждом слое и минимизировать функцию ошибки. В нашем примере для решения задачи классификации использовалась архитектура с одним скрытым слоем, состоящим из 100 нейронов, а также произведён эксперимент по сравнению различных вариантов конфигурации скрытых слоёв, что позволяет оценить влияние архитектурных изменений на качество классификации.

**StandardScaler** — это метод предварительной обработки данных, предоставляемый библиотекой scikit-learn, который стандартизирует признаки путем вычитания среднего значения и деления на стандартное отклонение для каждого ознак. Такой подход позволяет устранить влияние различий в масштабах признаков, что особенно важно при работе с методами, чувствительными к масштабу данных, такими как нейронные сети. Стандартизация улучшает сходимость алгоритмов оптимизации и может повысить общую точность модели.

**PCA (Principal Component Analysis)** — метод снижения размерности, который позволяет выделить основные компоненты, объясняющие наибольшую дисперсию в данных. Применение PCA способствует устранению избыточности и коррелированности признаков, что упрощает обучение моделей без существенной потери информативности. В данной работе сокращение размерности до 50 главных компонент позволило не только ускорить процесс обучения, но и улучшить качество модели за счёт устранения шума.

**Экспериментальный подбор гиперпараметров** Для оптимизации моделей важно подбирать такие гиперпараметры, как скорость обучения, степень регуляризации и структура сети. В нашем коде для Perceptron использовалась функция валидационной кривой для оценки влияния изменения коэффициента обучения (eta0) на точность работы модели, а для MLPClassifier – сравнение различных вариантов архитектур скрытых слоёв. Хотя эксперименты можно расширять (например, подбирая параметры learning\_rate\_init, alpha, или выбирая разные алгоритмы оптимизации), предложенный набор экспериментов демонстрирует базовый принцип оптимизации параметров нейронных сетей.

# **3. Описание набора данных**

Данный датасет представляет собой сбор данных о перемещениях пользователя в помещении, полученных с помощью беспроводной сенсорной сети (WSN). В отличие от статических признаков, как в задачах классификации изображений, этот набор данных содержит временные последовательности сигналов RSS (Received Signal Strength), отражающие динамику перемещения человека в реальном офисе. Такие сигналы позволяют исследовать закономерности в изменении радиочастотных характеристик при различных типах движения. Ниже приведено краткое описание ключевых составляющих набора данных:

**Временные последовательности RSS сигналов:** Основой набора данных являются временные ряды, представляющие изменения мощности радиосигнала между мобильным датчиком пользователя и стационарными анкерами. Каждый сигнал записывается с четырёх анкеров (столбцы RSS\_anchor1, RSS\_anchor2, RSS\_anchor3, RSS\_anchor4). Эти записи отражают динамику изменения силы сигнала во времени и служат исходными признаками для последующего анализа.

**Статистические признаки:** Для преобразования временных последовательностей в формат, пригодный для классификации, из каждого временного ряда извлекаются статистические характеристики - среднее значение и стандартное отклонение по каждому из четырёх анкеров. Это позволяет сохранить информацию как об уровне сигнала, так и о его изменчивости в течение последовательности.

**Классовая метка:** Каждая временная последовательность снабжена бинарной меткой, соответствующей типу перемещения пользователя. В датасете предусмотрены два класса: "+1" для перемещений со сменой пространственного контекста (переход в другую комнату) и "-1" для перемещений без смены контекста (остаётся в той же комнате). Эти метки используются при обучении моделей для классификации типа движения.

**Объем и структура данных:** Набор данных содержит 314 временных последовательностей, суммарно составляющих 13197 отдельных измерений. Данные организованы в виде отдельных CSV-файлов для каждой последовательности (MovementAAL\_RSS\_1.csv, ..., MovementAAL\_RSS\_314.csv), что обеспечивает модульность и удобство обработки. Такое представление позволяет эффективно работать с разнородными по длине временными рядами и учитывать индивидуальные особенности каждого траекторного движения.

# **4. Реализация**

**Загрузка и подготовка данных**

Программа начинает с загрузки целевых меток из файла MovementAAL\_target.csv, который содержит информацию о классах перемещений для каждой последовательности. Данные считываются с игнорированием строк-комментариев, начинающихся с символа '#', что обеспечивает корректное чтение заголовков и основных данных.

**Листинг 1.** Загрузка MovementAAL\_target.csv

|  |
| --- |
| **# Загружаем целевые метки, игноря заголовок**  **targets\_data = pd.read\_csv('Database/dataset/MovementAAL\_target.csv', header=None)**  **# Ввод названия колонок вручную, так как заголовок игнорится**  **targets\_df.columns = ['sequence\_ID', 'class\_label']** |

После загрузки целевых меток программа последовательно обрабатывает каждый файл с данными RSS сигналов, соответствующий идентификаторам последовательностей из целевого файла. Для каждого файла производится загрузка временных рядов сигналов с четырех анкеров.:

**Листинг 2.** Обработка остальных файлов БД

|  |  |
| --- | --- |
| |  | | --- | | **# Проходим по всем строкам файла с метками**  **for index, row in targets\_df.iterrows():**  **seq\_id = row['sequence\_ID']**  **label = row['class\_label']**    **# Путь к файлам с данными**  **file\_path = f'Database/dataset/MovementAAL\_RSS\_{seq\_id}.csv'**    **# Существует ли файл**  **if os.path.exists(file\_path):**  **# Загрузка CSV-файла, пропускаем строку с заголовком**  **df = pd.read\_csv(file\_path, comment='#', header=None)**  **# Названия колонок, так как заголовок с комментом**  **df.columns = ['RSS\_anchor1', 'RSS\_anchor2', 'RSS\_anchor3', 'RSS\_anchor4']** | |
|  |

**Разделение данных**

После загрузки и обработки временных последовательностей программа преобразует данные в единый массив признаков, используя статистические характеристики сигналов. Для каждой последовательности вычисляются средние значения и стандартные отклонения по четырем анкерам, что позволяет сохранить информацию об уровне и изменчивости RSS сигналов в компактном представлении.

**Листинг 3.** Преобразование данных

|  |  |
| --- | --- |
| |  | | --- | | **# Представления последовательности**  **# 1. Усреднение. Берётся среднее по каждому из 4 якорей.**  **sequence\_features = df.mean().values**    **# 2. Стандартное отклонение для учета изменчивости сигнала.**  **sequence\_std = df.std().values**    **# Объединение всех (средних и стандартных) отклонений**  **sequence\_features = np.concatenate([sequence\_features, sequence\_std])**    **# Добавление признаков и меток в списки**  **X\_list.append(sequence\_features)**  **y\_list.append(label)** | |

Затем с помощью функции train\_test\_split данные разбиваются на обучающую (60%), валидационную (20%) и тестовую (20%) выборки. Разделение выполняется в два этапа с сохранением распределения классов через параметр stratify, что позволяет оценивать модели на ранее не встречавшихся данных.

**Листинг 4.** Разделение данных

|  |  |
| --- | --- |
| |  | | --- | | **# Разделяем данные. 60% - обучение, 20% - валидация, 20% - финальный тест.**  **X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.4, random\_state=42, stratify=y)**  **X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42, stratify=y\_temp)** | |

**Обучение моделей**

В эксперименте используются две модели нейронных сетей для сравнения их эффективности в задаче классификации перемещений пользователя:

**Perceptron**: Модель однослойного перцептрона обучается с поиском оптимальных гиперпараметров, включая коэффициент обучения (eta0), параметр регуляризации (alpha) и тип штрафа (penalty). Для комплексного подбора параметров используется метод GridSearchCV с кросс-валидацией, что позволяет найти наилучшую комбинацию настроек для данной задачи.

**Листинг 5.** Perceptron (один нейрон)

|  |  |
| --- | --- |
| |  | | --- | | **param\_grid\_perceptron = {**  **'alpha': [0.0001, 0.001, 0.01], # Штраф за сложность**  **'eta0': [0.001, 0.01, 0.1], # Шаги при обучении (чем меньше, тем точнее и наоборот, грубее)**  **'penalty': [None, 'l2', 'l1', 'elasticnet'] # None - без регуляризации (риск переобучения), l2 - штраф за большие веса, l1 - обнуляет неважные веса, elasticnet - комбинация l1 и l2.**  **}**  **grid\_perceptron = GridSearchCV(Perceptron(random\_state=42, max\_iter=1000),**  **param\_grid\_perceptron, cv=5, scoring='accuracy', n\_jobs=-1) # 180 обучений**  **grid\_perceptron.fit(X\_train\_scaled, y\_train)** | |

**MLPClassifier:** Многослойный перцептрон обучается с поиском оптимальной архитектуры сети, включая размер скрытых слоев, функции активации и параметры оптимизации. Для исследования влияния структуры сети на качество классификации проводится эксперимент с различными конфигурациями скрытых слоев и функциями активации, что позволяет выявить наиболее эффективную архитектуру для обработки RSS сигналов.

**Листинг 6.** MLPClassifier (многослойная нейросеть)

|  |  |
| --- | --- |
| |  | | --- | | **param\_grid\_mlp = {**  **'hidden\_layer\_sizes': [(50,), (100,)], # Нейронов в скрытом слое (чем выше, тем больше мощи, но риск переобучения)**  **'activation': ['relu', 'tanh'], # Функции активации (relu - выпрямитель (быстрый), tanh - тангенс гиперболический (плавный) (остальные либо устарело (как logistic), либо неактуально для MLP (identity))**  **'alpha': [0.0001, 0.001, 0.01], # штрафы за сложность (регуляризация) - чем меньше, тем стабильнее и медленнее, и наоборот, если больше, то модель проще и меньше переобучение**  **'learning\_rate\_init': [0.001, 0.01], # Скорость обучения**  **'max\_iter': [2000] # Итерации**  **}**  **grid\_mlp = GridSearchCV(**  **MLPClassifier(random\_state=42, early\_stopping=True, n\_iter\_no\_change=10),**  **param\_grid\_mlp, # Даём список всех настроек для перебора**  **cv=5, # Проверка каждой настройки 5 раз на разных данных (надёжность) - 24 параметра = 120 обучений.**  **scoring='accuracy', # Сравнение по точности**  **n\_jobs=-1**  **)** | |

**Оценка моделей**

После завершения обучения и настройки гиперпараметров производится финальная оценка качества моделей на тестовой выборке. Для этого лучшие версии моделей, найденные в результате GridSearch, переобучаются на объединенных обучающих и валидационных данных, что позволяет максимально эффективно использовать доступную информацию перед финальным тестированием.

**Листинг 7.** Переобучение лучших моделей на объединённых данных

|  |  |
| --- | --- |
| |  | | --- | | **# Переобучение лучших моделей на объединенных данных**  **X\_train\_val\_scaled = np.vstack([X\_train\_scaled, X\_val\_scaled])**  **y\_train\_val = np.hstack([y\_train, y\_val])**  **best\_mlp.fit(X\_train\_val\_scaled, y\_train\_val)**  **best\_perceptron.fit(X\_train\_val\_scaled, y\_train\_val)** | |

Точность каждой модели измеряется на тестовой выборке с использованием метрики accuracy\_score из библиотеки scikit-learn. Дополнительно для лучшей модели строится детализированный отчет классификации, включающий precision, recall и F1-score для каждого класса, что дает комплексное представление о качестве предсказаний.

**Листинг 8.** Финальная точность

|  |  |
| --- | --- |
| |  | | --- | | **# Предсказание на отложенной тестовой выборке**  **y\_test\_pred\_mlp = best\_mlp.predict(X\_test\_scaled)**  **y\_test\_pred\_perceptron = best\_perceptron.predict(X\_test\_scaled)**  **# Финальная точность**  **final\_accuracy\_mlp = accuracy\_score(y\_test, y\_test\_pred\_mlp)**  **final\_accuracy\_perceptron = accuracy\_score(y\_test, y\_test\_pred\_perceptron)**  **# Детальный отчет по классификации**  **print("\nДетальный отчет для MLP:")**  **print(classification\_report(y\_test, y\_test\_pred\_mlp))** | |

Полученные значения точности дают объективное представление о производительности каждой модели и позволяют сравнивать их между собой, обеспечивая надежную основу для выводов об эффективности различных архитектур нейронных сетей для задачи классификации перемещений пользователя.

**Поиск оптимальных параметров**

Для нахождения наилучших параметров моделей используется систематический подход с применением GridSearchCV, который позволяет автоматизировано перебирать заданные комбинации гиперпараметров. Для каждой модели определяется сетка параметров, охватывающая ключевые аспекты их архитектуры и процесса обучения.

Для MLPClassifier проводится сравнительный анализ различных архитектур скрытых слоёв и функций активации, что позволяет определить оптимальную структуру сети для обработки RSS сигналов. Одновременно исследуется влияние параметра регуляризации alpha на способность модели к обобщению.

**Листинг 8.** Поиск параметров для MLPClassifier

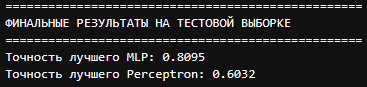
|  |  |
| --- | --- |
| |  | | --- | | **grid\_mlp = GridSearchCV(**  **MLPClassifier(random\_state=42, early\_stopping=True, n\_iter\_no\_change=10),**  **param\_grid\_mlp, cv=5, scoring='accuracy', n\_jobs=-1**  **)** | |

Для Perceptron анализируется зависимость точности от значений коэффициента обучения eta0 и типа регуляризации через перебор всех возможных комбинаций. Такой комплексный подход позволяет выявить оптимальные настройки для каждой модели и предоставляет информацию для дальнейшей оптимизации при решении задачи классификации перемещений пользователя.

**Листинг 9.** Поиск параметров для Perceptron

|  |  |
| --- | --- |
| |  | | --- | | **grid\_perceptron = GridSearchCV(Perceptron(random\_state=42, max\_iter=1000),**  **param\_grid\_perceptron, cv=5, scoring='accuracy', n\_jobs=-1)** | |

# **5. Тестирование**



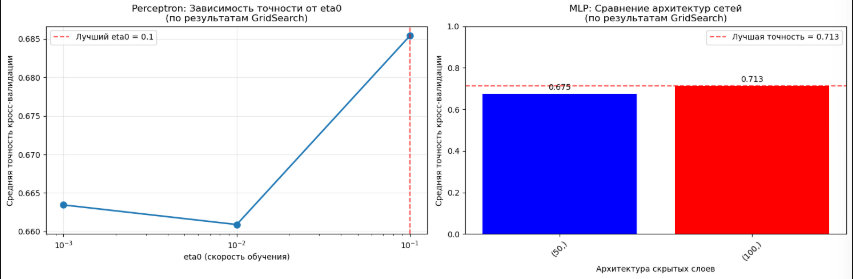


Рис. 1 – Вывод в консоль

# **6.** **Выводы**

Результаты экспериментов демонстрируются посредством консольного вывода и графического представления. Сначала отображаются базовые показатели точности моделей Perceptron и MLPClassifier, полученные на валидационной выборке с параметрами по умолчанию. Затем, после подбора оптимальных гиперпараметров с использованием GridSearchCV, выводятся финальные метрики на тестовой выборке, что позволяет оценить улучшение качества классификации перемещений пользователя.

Экспериментальные результаты показывают, что MLPClassifier значительно превосходит однослойный перцептрон в задаче классификации перемещений пользователя по RSS сигналам. MLP достиг точности 80.95% на тестовой выборке, в то время как Perceptron показал результат 60.32%. Это подтверждает, что задача классификации перемещений требует нелинейной модели для выявления сложных закономерностей в временных последовательностях RSS сигналов.

Анализ валидационных кривых и сравнение архитектур сетей демонстрируют важность тщательного подбора гиперпараметров для достижения оптимальной производительности моделей. Найденные оптимальные параметры для MLP (скрытый слой из 100 нейронов, функция активации relu, коэффициент регуляризации 0.0001) обеспечивают наилучшее качество классификации при сохранении вычислительной эффективности.