**Министерство образования и науки Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

ИНСТИТУТ ФИЗИКИ

КАФЕДРА РАДИОФИЗИКИ

Направление: 03.03.03 - Радиофизика

Профиль: Информационные процессы и коммуникационные (киберфизические) системы

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

**МЕТОДЫ ДЕТЕКТИРОВАНИЯ СТЕПЕНИ МУСКУЛЬНОЙ УСТАЛОСТИ ПО ДАННЫМ ДАТЧИКОВ АКСЕЛЕРОМЕТРОВ И ГИРОСКОПОВ**

Студент 4 курса группы 06-921

«\_\_» \_\_\_\_\_\_\_\_2023 г. \_\_\_\_\_\_\_\_\_\_\_ (Якунин В.С.)

Научный руководитель

к.т.н., заместитель директора ИВМиИТ

«\_\_» \_\_\_\_\_\_\_\_2023 г. \_\_\_\_\_\_\_\_\_\_\_ (Егорчев А.А.)

Заведующий кафедрой киберфизических технологий

д.ф.-м.н., профессор

«\_\_» \_\_\_\_\_\_\_\_2023 г. \_\_\_\_\_\_\_\_\_\_\_ (Шерстюков О.Н.)

Казань — 2023

[ВВЕДЕНИЕ 3](#_Toc138052181)

[ГЛАВА I ИССЛЕДОВАНИЕ СУЩЕСТВУЮЩИХ МЕТОДОВ И ДАТАСЕТОВ ОПРЕДЕЛЕНИЯ СТЕПЕНИ МУСКУЛЬНОЙ УСТАЛОСТИ 6](#_Toc138052182)

[1.1 Исследование и анализ существующих методов 6](#_Toc138052183)

[1.2 Исследование существующих датасетов с записанной походкой при разном уровне усталости 14](#_Toc138052184)

[1.3 Выводы по первой главе 15](#_Toc138052185)

[ГЛАВА II МАТЕМАТИЧЕСКИЕ МЕТОДЫ И МОДЕЛИ. АЛГОРИТМЫ РЕШЕНИЯ 16](#_Toc138052186)

[2.1 Машинное обучение 16](#_Toc138052187)

[2.2 Random Forest 20](#_Toc138052188)

[2.3 Гиперпараметры Random Forest 29](#_Toc138052189)

[2.4 Метрики качества модели 31](#_Toc138052190)

[2.5 Библиотека TSFEL 34](#_Toc138052191)

[2.6 Выводы по второй главе 35](#_Toc138052192)

[ГЛАВА III ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 37](#_Toc138052193)

[3.1 Запись данных 37](#_Toc138052194)

[3.1 Предобработка датасета и извлечение признаков 40](#_Toc138052195)

[3.2 Обучение модели 45](#_Toc138052196)

[3.2.1 Настройка гиперпараметров 47](#_Toc138052197)

[3.3 Структурные схемы 55](#_Toc138052198)

[3.4 Блок-схемы 58](#_Toc138052199)

[3.5 Выводы по третьей главе 68](#_Toc138052200)

[ГЛАВА IV РЕЗУЛЬТАТЫ ИСПОЛЬЗОВАНИЯ И ВНЕДРЕНИЯ 71](#_Toc138052201)

[ЗАКЛЮЧЕНИЕ 74](#_Toc138052202)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ 76](#_Toc138052203)

[ПРИЛОЖЕНИЯ 78](#_Toc138052204)

[Приложение 1. Создание, настройка и обучение модели 78](#_Toc138052205)

[Приложение 2. Внедрение модели 91](#_Toc138052206)

# ВВЕДЕНИЕ

**Актуальность темы исследования.** В современном мире люди в погоне за успехом и высокой зарплатой часто забывают о своем состоянии здоровья. В Японии для этого имеется определенный термин - “кароси”, который означает смертельные случаи на фоне переутомления и стресса. Таким образом, актуальность алгоритмов, способных определять уровень мускульной усталости может быть высокой в контексте мониторинга за своим здоровьем и физической формы. Также такие алгоритмы способны предотвращать травмы на рабочем мест, связанные с физической усталостью работника и уменьшить риск несчастных случаев на производстве. Алгоритмы способные определять степень мускульной усталости могут стать мотивацией для занятий спортом и улучшению общего здоровья. Они также могут помочь спортсменам управлять своей нагрузкой и избежать перетренировки, которые могут привести к травмам. Развитие таких алгоритмов и технологий может привести к улучшению здоровья и повышению физической активности, что может иметь положительный эффект на общественное здоровье и благополучие.

**Цель работы** – разработка алгоритма машинного обучения для определения степени мускульной усталости с целью биомедицинского мониторинга.

Для достижения данной цели были сформулированы следующие **задачи**:

1)Исследование существующих решений: необходимо провести обзор и изучение существующих подходов и методов, применяемых для оценки мускульной усталости на основе данных акселерометра и гироскопа.

2)Сбор необходимых данных: необходимо собрать измерения с датчиков акселерометра и гироскопа телефона с последующей разметкой, указывающей степень мускульной усталости;

3)Предварительная обработка данных: отчистка и преобразование данных для дальнейшего анализа;

4)Выделение признаков: извлечение информативных признаков из данных акселерометра и гироскопа;

5)Построение модели машинного обучения: выбор подходящей модели, которая будет способна обрабатывать полученные данные и прогнозировать степень мускульной усталости;

6)Обучение модели: использовать извлеченные признаки для построения выбранной модели, настройка гиперпараметров модели;

7)Оценка производительности модели: необходимо оценить производительность модели на тестовом наборе данных, определить используемую метрику для оценки модели;

8)Развертывание модели: реализовать разработанную модель в виде программного кода, который можно будет использовать для определения степени мускульной усталости в реальном времени.

**Объект** **исследования** **–** процесс мускульной усталости у человека и его связь с данными, полученными от датчиков акселерометра и гироскопа.

**Предмет** **исследования** **–** разработка алгоритма машинного обучения, который способен анализировать данные с датчиков и предсказывать степень мускульной усталости. Это включает в себя предварительную обработку данных, выделение признаков, выбор и обучение модели, а также оценку производительности и оптимизации модели.

**Научная новизна –** использование датчиков на телефоне для определения степени мускульной усталости. Данная задача является одним из направлений развития медицинских технологий и биометрических систем, которые становятся все более доступными для широкого круга людей. Эта технология позволяет людям легко отслеживать свой уровень физической активности и мониторить свою форму с помощью устройств, которые у них уже есть – смартфоны. Применение датчиков на телефоне в области биометрических систем и здоровья является относительно новой областью и привлекает большое внимание исследователей. Данные технологии продолжают развиваться и усовершенствоваться. В контексте задачи определения степени мускульной усталости - научной новизной выступает разработка более эффективных методов мониторинга физической активности и управления уровнем мускульной усталости.

**Практическая ценность –** разработанная система биомедицинского мониторинга для определения степени мускульной усталости имеет ряд практических преимуществ, которые могут быть полезными для людей, занимающихся спортом и фитнесом, а также для тех, кто хочет следить за своим здоровьем и уровнем физической активности:

1. Мониторинг степени мускульной усталости: приложения, использующие датчики на телефоне, могут помочь людям мониторить свою степень мускульной усталости и предотвращать переутомление. Это может быть полезно для спортсменов, чтобы избежать травм и повысить эффективность тренировок.
2. Данные технологии способны помочь оптимизировать людям свои тренировки, чтобы достичь максимальной эффективности. Например, приложение может определить, когда мышцы находятся в оптимальном состоянии для выполнения упражнения.
3. Предотвращение травм: мониторинг степени мускульной усталости может помочь предотвратить травмы, связанные с перенапряжением и переутомлением мышц.

**Результаты исследования внедрены в НИОКР:** «Разработка программного комплекса централизованного дистанционного мониторинга основных показателей здоровья сотрудников с использованием технологий искусственного интеллекта» (Договор №04/22Б от 14.02.2022).

# ГЛАВА I ИССЛЕДОВАНИЕ СУЩЕСТВУЮЩИХ МЕТОДОВ И ДАТАСЕТОВ ОПРЕДЕЛЕНИЯ СТЕПЕНИ МУСКУЛЬНОЙ УСТАЛОСТИ

## Исследование и анализ существующих методов

1)Swapnali Babasaheb Karvekar, "Smartphone-based Human Fatigue Detection in an Industrial Environment Using Gait Analysis " (Декабрь 2019г). Thesis. Rochester Institute of Technology [1].

Цель исследования: разработка алгоритма способного определять физическую усталость, для классификации уровня усталости человека на рабочем месте с помощью датчиков движения, встроенных в телефон.

Основные особенности: детальное рассмотрение физиологии работы мышц при физической нагрузке, рассматривается точность предсказания в зависимости от количества уровней мускульной усталости. Для предсказания применяется метод опорных векторов. В статье предлагается калибровка для повышения точности алгоритма с помощью уровня эмоциональной усталости. При высоком уровне стресса человек должен уставать быстрее, а при маленьком уровне стресса медленнее. Используется шкала Борга для субъективной оценки уровня усталости.

Порядок проведения испытаний: в сборе данных участвовало 24 человека, к каждому крепились датчики трехосевого акселерометра и гироскопа (Рис. 1). Записи набора данных производились после выполнения определенного количества приседаний, телефон крепился к правой голени испытуемых. После этого с целью получения результатов и обучению модели данные фильтровались и сегментировались. Недостатками работы отмечается различная выносливость и физическая сила по сравнению с реальными рабочими, достижения необходимого уровня усталости за быстрый промежуток времени, который не соответствует реальной длительности рабочей смены.

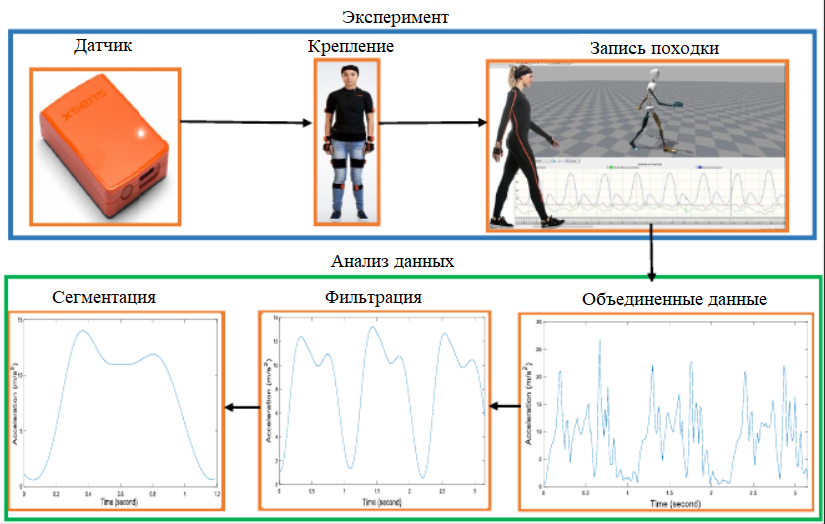


Рисунок 1. Структурная схема сбора и анализа данных

Результаты проведения испытаний: при первом варианте, где рассматривалась двухуровневая шкала усталости – устал и не устал, точность составила 91%. Для трехуровневой шкалы – не устал, средняя усталость, высокая усталость, точность 76%. Для четырёхуровневой – не устал, легкая усталость, средняя усталость, сильная усталость, точность 61%.

2)Nuno Cardoso, "MLFatigueDetection Machine Learning Based Walking Fatigue Detection" (Декабрь 2019г). Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa [2].

Цель исследования: разработка алгоритма определения усталости по походке человека в реальном времени с целью уменьшения рисков получения травмы на рабочем месте.

Основные особенности: в статье рассматривалась точность предсказания в зависимости от области крепления датчиков (Рис. 2). Протестированы различные условия сбора и обработки данных в попытке определить оптимальные условия для реализации алгоритмов. Для оценки уровня усталости у испытуемых используется шкала Борга (Таб. 1), предварительно уменьшив ее в масштабе. Шкала Борга – субъективный способ определения уровня нагрузки во время занятий лечебной физкультурой. Необходимые признаки рассчитывались с помощью библиотеки TSFEL.

Таблица 1. Шкала Борга

|  |  |
| --- | --- |
| Баллы | Интенсивность нагрузки |
| 20 | Максимально тяжело |
| 19 | Очень, очень тяжело |
| 18 | Очень тяжело |
| 17 | Очень тяжело |
| 16 | Трудно/Тяжело |
| 15 | Трудно/Тяжело |
| 14 | Довольно тяжело |
| 13 | Немного тяжело |
| 12 | Легко |
| 11 | Довольно легко |
| 10 | Очень легко |
| 9 | Очень легко |
| 8 | Исключительно легко |
| 7 | Исключительно легко |
| 6 | Очень, очень легко |

Порядок проведения испытаний: 12 участников, среди которых 9 мужчин и 3 женщины, просили выполнять такие упражнения, которые похожи на те, которые приходится выполнять людям на различных предприятиях (Рис. 3). Упражнения выполнялись до тех пор, пока испытуемый не оценит свой уровень усталости по определенному баллу. Дальше производилась запись данных, после этого испытуемый повторял упражнения до того момента, пока не оценит свой уровень усталости на следующий необходимый балл. Испытания производились до достижения максимального уровня усталости, которое соответствует 20 баллам по шкале Борга.

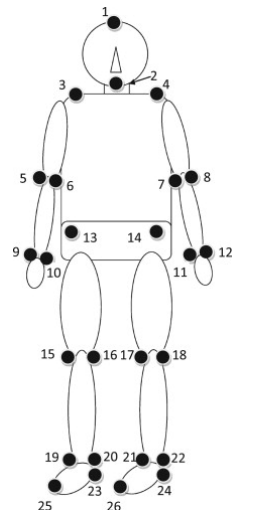


Рисунок 2. Варианты исследуемых креплений датчиков

Результаты проведения испытаний: лучшим расположением датчиков для классификации уровня мускульной усталости по метрике точность, оказалась поясничная область. Точность для разной длительности временного окна (3-10с) в области карманов брюк достигает 88–96%. Лучшие результаты для всех метрик достигались при длительности временного окна 5с. Наилучшая точность предсказания для трех разных временных окнах представлена в таблице 2.

Также была проверена возможность использования телефона в качестве датчика для определения уровней мускульной усталости. Полученные результаты представлены в таблице 3.

Таблица 2. Наилучшая точность предсказания для трех разных окнах

|  |  |  |  |
| --- | --- | --- | --- |
| Окно | Линейный метод опорных векторов | Полиноминальный метод опорных векторов | NN |
| 5 секунд | 86.4  (Лодыжка) | 97.2  (Пояс) | 93.7  (Пояс) |
| 3 секунды | 88.3  (Пояс) | 97.2  (Лодыжка) | 95.3  (Пояс) |
| 10 секунд | 79.8  (Грудь) | 90.4  (Пояс) | 93.8  (Пояс) |

Таблица 3. Полученные метрики моделей для телефона

|  |  |  |  |
| --- | --- | --- | --- |
| Метрика | Линейный метод опорных векторов | Полиноминальный метод опорных векторов | NN |
| Accuracy | 71.11 | 88.16 | 81.90 |
| Recall | 71.98 | 87.27 | 84.72 |
| Specificity | 70.39 | 89.06 | 78.62 |

3)[Sadra Fardhosseini](https://www.researchgate.net/profile/Sadra-Fardhosseini), [Mahmoud Habibnezhad](https://www.researchgate.net/profile/Mahmoud-Habibnezhad), [Houtan Jebelli](https://www.researchgate.net/profile/Houtan-Jebelli), [Giovanni Migliaccio](https://www.researchgate.net/profile/Giovanni-Migliaccio-2), [Hyun Woo Lee](https://www.researchgate.net/profile/Hyun-Woo-Lee-8), [Jay Puckett](https://www.researchgate.net/profile/Jay-Puckett-2),"Recognition of Construction Workers’ Physical Fatigue Based on Gait Patterns Driven from Three-axis Accelerometer Embedded in a Smartphone" (Март 2020г). Construction Research Congress: TEMPE, ARIZONA [3].

Цель исследования: разработка системы для определения усталости работников путем изучения их походки, измеряемой с помощью трехосевого акселерометра с целью предотвращения несчастных случаев со смертельным исходом в строительной отрасли.

Основные особенности: для классификации уровня усталости используется метод опорных векторов. Исследование было сосредоточено на локализованной усталости, а не на общей физической усталость. Реализована двухуровневая шкала усталости.

Порядок проведения испытаний: в исследовании участвовало пять строителей, производилась запись их походки до и после выполнения упражнений, вызывающих усталость (Рис. 3).



Рисунок 3. Этапы проведения эксперимента

Результаты проведения испытаний: усталость работников была обнаружена с точностью 82% и 87% соответственно с использованием метода опорных векторов (Таб. 4).

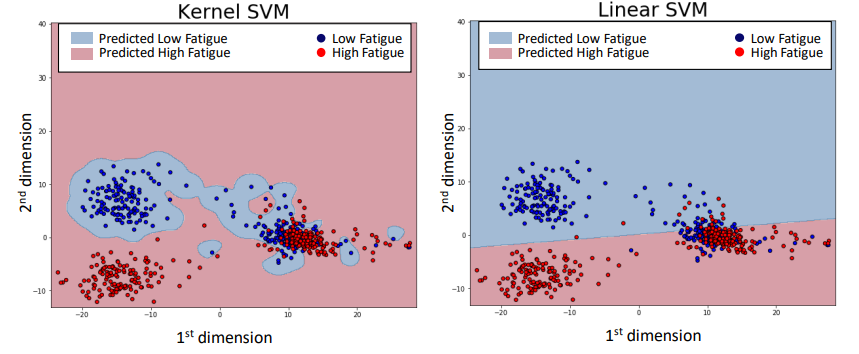
****

Рисунок 4. Результаты работы линейного и нелинейного метода опорных векторов

Таблица 4. Точность классификации для различных объектов с использованием линейного и нелинейного метода опорных векторов (SVM)

|  |  |  |
| --- | --- | --- |
| Участники | Линейный SVM  Точность (%) | Нелинейный SVM  Точность (%) |
| Участник1 | 82.75 | 87.93 |
| Участник2 | 82.44 | 89.41 |
| Участник3 | 84.42 | 86.45 |
| Участник4 | 80.19 | 83.77 |
| Участник5 | 83.56 | 85.98 |

4)Wen-Wen Yang, Kuan-Hsun Huang, En-Tzu Wang and Tzyy-Yuang Shiang , "DETECTING PHYSICAL FATIGUE USING ACCELEROMETER IN BASKETBALL PLAYERS" (Июль 2019г). Department of Athletic Performance, National Taiwan Normal University, Taipei, Taiwan [4].

Цель исследования: определение возможности использования трехосевого акселерометра для обнаружения изменения уровня усталости у баскетболистов.

Основные особенности: в данной статье определяли уровень усталости по шкале Борга, которая вычисляется путем деления измеренной частоты биений сердца на 10. Для этого используются дополнительные датчики: с помощью пульсометра на груди и на запястье измеряется частота биения сердца, после чего полученные показания переводятся в шкалу Борга.

Порядок проведения испытаний: восемь баскетболистов любителей мужского пола выполняли набор упражнений, связанных с баскетболом (Рис. 5). Для контроля изменений ускорения использовался трехосевой акселерометр. Было проанализировано суммарное ускорение для каждого выполненного упражнения. Усталость по шкале Борга (Таб. 1) – трехуровневая.

Результаты проведения испытаний: результаты показали, что суммарное ускорение в вертикальном направлении значительно различалось между уровнями усталости и имело положительную связь со временем выполнения упражнения (коэффициент корреляции Пирсона r = 0.87).

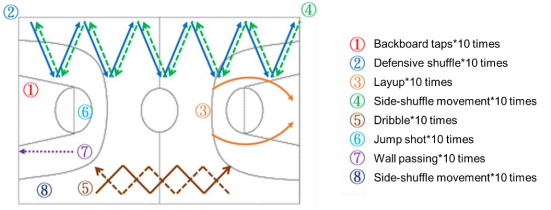


Рисунок 5. Упражнения для баскетболистов

Эти результаты свидетельствуют о том, что трехосевой акселерометр может быть использован в качестве инструмента мониторинга, позволяющего различать состояние усталости у баскетболистов (Рис. 6).

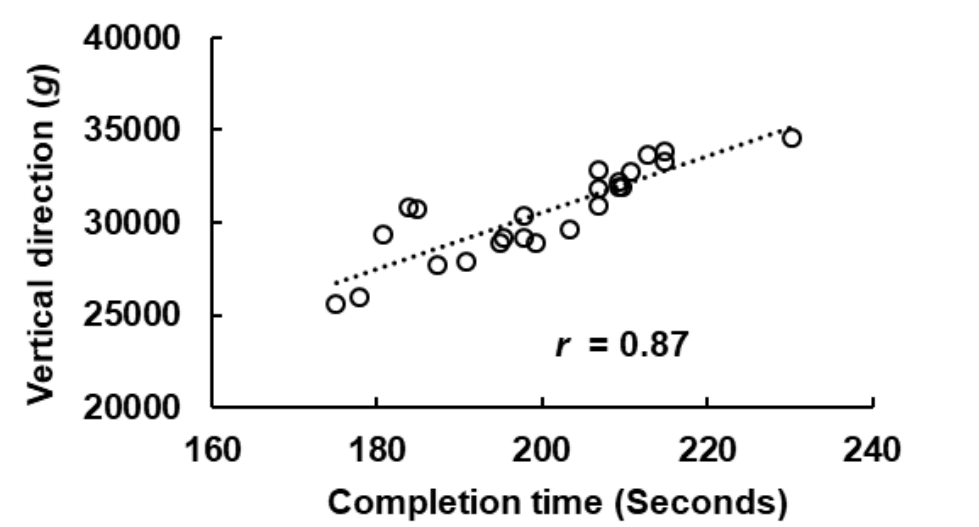


Рисунок 6. Корреляционная связь между ускорением и временем выполнения упражнений

5)[Karla Gonzalez](https://www.researchgate.net/profile/Karla-Gonzalez-5), [Farzan Sasangohar](https://www.researchgate.net/profile/Farzan-Sasangohar), [Ranjana K Mehta](https://www.researchgate.net/profile/Ranjana-Mehta), [Mark Lawley](https://www.researchgate.net/profile/Mark-Lawley-2), [Madhav Erraguntla](https://www.researchgate.net/profile/Madhav-Erraguntla), "Measuring Fatigue through Heart Rate Variability and Activity Recognition: A Scoping Literature Review of Machine Learning (Сентябрь 2017г). Environmental and Occupational Health, Texas A&M University [5].

Цель исследования: предоставить общий обзор литературы о современных направлений исследований в области определения усталости с приложения для распознавания активности человека и частоты сердечного ритма. Рассмотрена способность моделей машинного обучения распознавать вид трудовой деятельности. Обзор литературы был проведен по статьям, опубликованным с 1985 по 2016 год.

Основные особенности: рассматривалось множество исследований, но все они базировались на использовании различных датчиков, которые расположены на запястье, лодыжках и на бедре. Анализ статей показал возможность идентифицировать вид деятельности человека по данным датчиков: стоит, лежит, бежит, поднимается по лестнице.

## Исследование существующих датасетов с записанной походкой при разном уровне усталости

Поиск датасетов осуществлялся на следующих ресурсах:

* + На платформе Kaggle представлено более 23 000 наборов данных, охватывающих широкий спектр тематик от здравоохранения до мультфильмов. Эти наборы данных, используемые в соревнованиях, обычно более детализированы по сравнению с общедоступными ресурсами.;
  + Google Cloud Public Datasets предлагает более сотни наборов данных, размещенных в облачном хранилище BigQuery. Эти наборы данных были собраны из различных источников, таких как GitHub, бюро переписи населения США, NASA, BitCoin и многих других;
  + Amazon Web Services Open Data Registry позволяет загружать и изучать наборы данных с использованием Elastic Compute Cloud. Open Data Registry является частью программы AWS Public Dataset, которая направлена на демократизацию доступа к данным;
  + UCI Machine Learning Repository является одним из самых давних источников данных, функционирующих с 1987 года. Наборы данных UCI отлично подходят для задач машинного обучения благодаря возможности настройки параметров загрузки.

Критерии по поиску набора данных:

* + Набор данных должен содержать записи походки с разными уровнями усталости;
  + Набор данных должен содержать записи с датчиков трехосевого акселерометра и гироскопа;
  + К набору данных должна быть информация о том, как проводилась запись, какие датчики и их параметры использовались при записи;
  + Датчики должны быть расположены в области, которые соответствуют расположению карманов брюк.

Ввиду того, что задача является узкоспециализированной, найти готовый записанный набор данных, с различными уровнями усталой походкой, на всех перечисленных источниках не удалось. Ввиду этого было решено создать свой датасет.

## Выводы по первой главе

В рамках данной главы был проведен анализ возможности классифицировать степень мускульной усталости. Было выяснено, что можно достичь этой цели с помощью датчиков акселерометра и гироскопа, которые имеются на смартфонах без использования каких-либо других дополнительных датчиков. Основными моделями для классификации уровней мускульной усталости является модель SVM. Оптимальное временное окно для вычисляемых признаков, по статье [2], составляет пять секунд. Также в статье [2] было определено, что наилучшим местом крепления для датчиков является поясничная область. В данной статье были продемонстрированы наилучшие результаты, а именно разделение усталости на три класса с максимальной точностью предсказания среди всех исследуемых статей. Это говорит о благоприятной возможности использовании телефона в кармане брюк в качестве средства детектирования степени мускульной усталости. Исследование существующих открытых датасетов, на всех перечисленных ресурсов, указанных выше, показало их отсутствие.

# ГЛАВА II МАТЕМАТИЧЕСКИЕ МЕТОДЫ И МОДЕЛИ. АЛГОРИТМЫ РЕШЕНИЯ

## 2.1 Машинное обучение

Машинное обучение – это подраздел искусственного интеллекта, который изучает разработку алгоритмов и моделей, позволяющих компьютеру автоматически обучаться и делать прогнозы или принимать решения на основе данных, без явного программирования. Основная идея машинного обучения заключается в том, чтобы компьютер самостоятельно извлекал полезные закономерности из больших объемов данных и использовал их для решения конкретных задач.

Машинное обучение состоит из двух основных этапов: обучения и применения. Во время обучения модель анализирует обучающие данные и строит статистическую модель или алгоритм, который может делать предсказания или принимать решения на основе новых данных. Во время этапа применения модель используется для предсказания или принятия решений на новых данных, которые она ранее не видела.

В зависимости от данных и поставленной задачи машинное обучение делится на три больших раздела:

1. Обучение с учителем;
2. Обучение без учителя;
3. Обучение с подкреплением.

Обучение с учителем является одним способов в машинном обучении, когда есть размеченные данные, то есть известны целевые переменные, которые необходимо предсказать. В обучении с учителем разделяют задачи классификации и задачи регрессии. В рамках поставленной задачи определения степени мускульной усталости, задача относится к классификации.

Классификация – задача определения к какому из К классов принадлежит объект U на основе его характеристик, представленных D мерным вектором X входных переменных.

Входные данные (input variables): , где – некоторые признаки (среднее значение, медиана, максимальное значение и так далее).

Векторы признаков: Целевая переменная:

… …

Задача классификации: , то есть целевая переменная является категориальной. Например 0 – не усталая походка, 1 – средняя усталость, 2 – высокая усталость.

После получения датасета с признаками и метками класса, следующим этапом в построении модели машинного обучения является предобработка данных и обучение модели. К основным шагам, которые выполняются на этом этапе относят:

1. Предварительный анализ данных: этот шаг включает исследование и анализ датасета для понимания его структуры, особенностей и распределений. Выполняется проверка пропущенных значений, выбросов или ошибок в данных.
2. Предобработка данных: в этом шаге выполняются различные операции по предварительной обработке данных для их отчистки и приведения к удобному формату. Это может быть удаление пропущенных значений, масштабирование признаков, преобразование категориальных признаков в числовое и так далее.
3. Разделение датасета: обычно датасет разделяется на обучающую, валидационную и тестовую выборку. Это помогает оценить производительность модели на новых данных и контролировать ее способность к обобщению.

Подробнее о каждой выборке:

1. Обучающая выборка: обучающий набор данных используется для обучения модели. Он содержит примеры данных с признаками и соответствующими метками (целевыми значениями). Модель анализирует эти данные и настраивает свои внутренние параметры, чтобы предсказывать целевые значения на основе признаков. Обычно обучающая выборка составляет от 60% до 80% от всего датасета.
2. Валидационная выборка: валидационный набор данных используется для оценки производительности модели во время обучения. Он помогает в настройке гиперпараметров модели и контроле за переобучением. Валидационная выборка содержит данные с признаками и соответствующими метками, аналогично обучающей выборке. Обычно размер валидационной выборки составляет около 10-20% от всего датасета.
3. Тестовая выборка: тестовый набор данных используется для окончательной оценки производительности модели после завершения обучения. Эта независимая выборка содержит данные с признаками, а модель используется для предсказания целевых значений на основе этих признаков. Затем сравниваются предсказанные значения с истинными значениями для оценки производительности модели. Обычно размер тестовый выборки составляет около 10-20% от всего датасета.

Разделение датасета на обучающую, валидационную и тестовую выборку позволяет оценить модель на новых данных и проверить ее способность к обобщению. Важно, чтобы данные в различных выборках были представительными и имели схожие статистические свойства, чтобы обеспечить надежные оценки производительности модели.

Для выбора оптимальной модели датасет был загружен в MATLAB и были протестированы все доступные модели в приложении Classification Learner (Рис. 7).

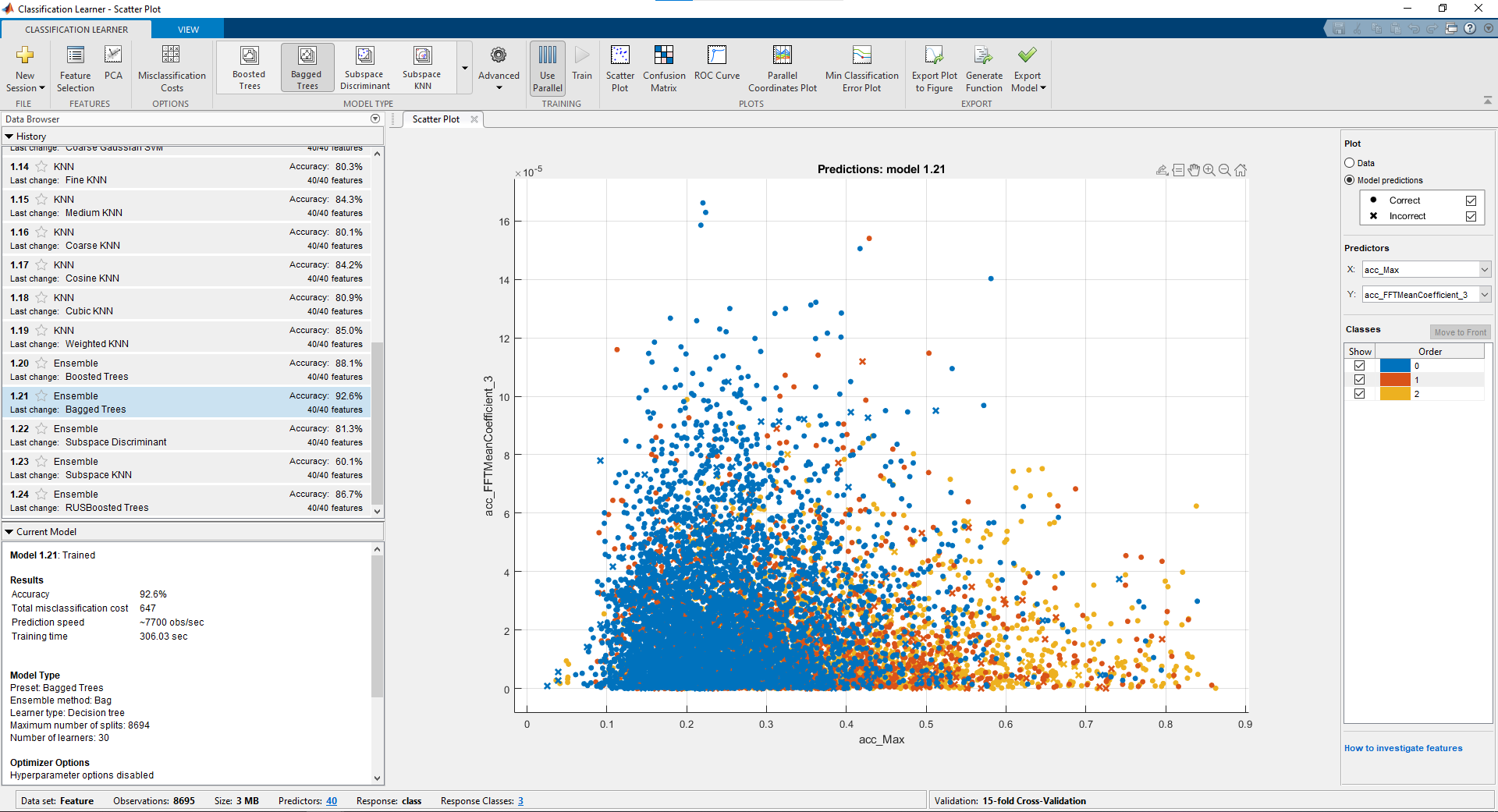


Рисунок 7. Результаты предсказаний моделей в MATLAB

В ввиду хорошей точности ансамблевых алгоритмов в MATLAB, было решено разрабатывать модель случайного леса, дополнительными преимуществами данной модели является возможность интерпретации результатов и хорошей устойчивости модели к переобучению.

## 2.2 Random Forest

Случайный лес (Random Forest) – это ансамблевая модель машинного обучения, которая комбинирует несколько деревьев решений для выполнения задач классификации и регрессии. Случайный лес относится к обучению с учителем.

Основные особенности модели случайного леса:

1. Ансамбль деревьев решений: случайный лес состоит из набора деревьев решений, называемых "деревьями решений в лесу". Каждое дерево строится независимо от остальных на разных подмножествах обучающего набора данных.
2. Случайность: в процессе построения каждого дерева в случайном лесу вводится случайность. Это происходит двумя способами. Во-первых, для каждого дерева случайным образом выбирается подмножество обучающих данных, называемое "случайной подвыборкой с возвращением" (bootstrap sample). Во-вторых, при разделении узлов дерева случайным образом выбираются только некоторые признаки из общего набора признаков.
3. Бэггинг: концепция бэггинга (bootstrap aggregating) используется для создания разных деревьев в случайном лесу. Бэггинг позволяет обучать каждое дерево на разных подмножествах данных, что способствует разнообразию и предотвращает переобучение.
4. Принятие решений на основе голосования: при классификации случайный лес принимает решение, основываясь на голосовании всех деревьев в лесу. Каждое дерево дает свой прогноз, и класс, получивший наибольшее количество голосов, становится окончательным предсказанием.
5. Оценка важности признаков: случайный лес предоставляет информацию о важности признаков. Признаки, которые чаще используются для разделения узлов в деревьях, считаются более важными для классификации или регрессии.

Преимущества модели случайного леса:

1. Хорошая обобщающая способность: случайный лес способен обобщать данные и обеспечивать хорошую производительность на новых примерах.
2. Устойчивость к переобучению: случайный лес имеет свойство устойчивости к переобучению, благодаря случайности в процессе построения модели.
3. Способность работать с большими объемами данных: модель случайного леса хорошо масштабируется и может обрабатывать большие наборы данных.
4. Интерпретируемость: модель случайного леса предоставляет информацию о важности признаков, что позволяет интерпретировать результаты. Этот факт делает его еще полезнее, потому что имеется возможность дальнейшего анализа связи мускульной усталости с походкой.

Недостатки модели случайного леса:

1. Потребление ресурсов: построение и использование случайного леса требуют большего объема вычислительных ресурсов по сравнению с некоторыми другими моделями.
2. Трудность интерпретации отдельных деревьев: в отличие от одиночных деревьев решений, каждое из которых может быть легко интерпретировано, интерпретация отдельных деревьев в случайном лесе сложнее.

В целом, случайный лес является мощным инструментом для задач классификации и регрессии, особенно когда имеется большой объем данных или когда требуется обобщение и устойчивость к переобучению.

Перед тем как переходить к описанию работы модели случайного леса, необходимо рассмотреть модель решающего дерева. Так как случайный лес является ансамблевой моделью, состоящей из нескольких деревьев решений, где каждое дерево в случайном лесу принимает решение независимо от остальных.

Дерево решений – алгоритм машинного обучения, представляющий собой набор слабых классификаторов, организованных в иерархическую структуру – дерево. Само дерево – это связанный ориентированный граф, в котором каждая вершина имеет не более одного входящего ребра (Рис. 8).

Принцип построения дерева заключается в максимальном приросте информации (уменьшении неопределенности) при спуске вниз по узлам. В самом терминальном узле будет находится целевая категориальная переменная, говорящая к какому классу относится объект.

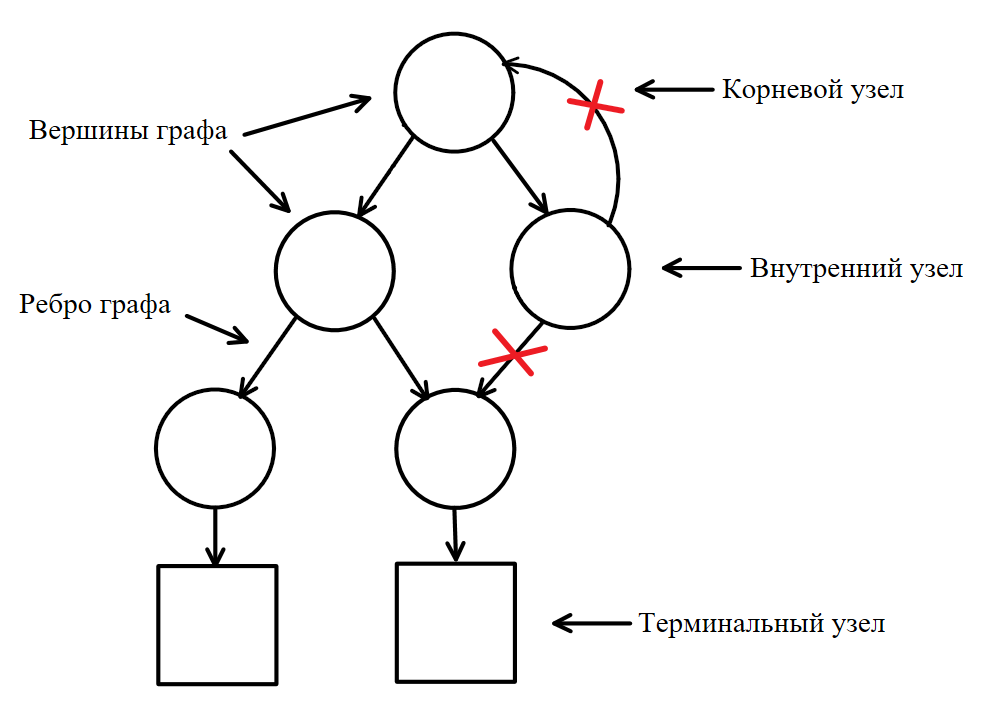


Рисунок 8. Основные понятия к определению решающего дерева

Рассмотрим некоторый внутренний узел, на вход которого подается вектор с признаками . Разделяющейся функцией будет называться такая функция, которая определяет следующий узел из m потомков в j-м внутренним узле для элемента :

где – параметры разделяющей функции.

Существуют различные разделяющие функции. Они могут разделять гиперплоскостью параллельно одной из осей путем задания некоторых порогов или возможно комбинировать с другими моделями и разделять с помощью линейной регрессии, но для разделения большого набора данных может потребоваться большое количество таких разделений, поэтому целесообразно использовать нелинейные функции:

где – индикаторные функции: [True] = 1, [False] = 0. – функция выбора характеристик, получающая на вход вектор входных переменных и возвращает вектор, содержащий подмножество входных переменных этого самого вектора , – вектор параметров модели.

Пусть: – набор классов;

– множество элементов обучающей выборки, дошедшей до i-го узла;

- множество элементов обучающей выборки k-го класса, дошедшей до i-го узла;

– количество элементов множества обучающей выборки, дошедшей до i-го узла;

- количество элементов множества обучающей выборки, дошедшей до i-го узла принадлежащих k-му классу;

– множество элементов в обучающей выборки, дошедших до узла j, потомка узла i.

Аналогичные определения для:

В качестве меры информации может выступать индекс Джини, который основан на понятии энтропии. Энтропия для системы с N возможными состояниями определяется по следующей формуле:

где – вероятность нахождения системы в i-ом состоянии.

Так как энтропия показывает степень хаоса системы, на основании этого дерево может понимать какие вопросы лучше задавать, которые приводят к увеличение информации. Приростом информации в данном случае является уменьшение энтропии. Целевая функция для внутреннего узла:

где q – число групп после разбиения.

Разделение будет считать хорошим, если после того, как будет известно в какого потомка будет спуск по дереву, это даст много информации о нашей выборке. То есть нужно смотреть на разницу энтропии, та которая была и та, которую получили. Целевая функция:

В итоге: имея для каждого перебираются все пороговые значения и для каждого разбиения считается целевая функция. Выбирается то разбиение, которое дает максимальный прирост информации.

Терминальный узел создается, когда выполнено одно из следующий условий:

1. Достигнута глубина дерева d.

Как правило, чем глубже дерево, тем чаще оно подвержено переобучению и наоборот, чем меньше оно, тем чаще недообучено.

Узел создается следующим образом:

где – вектор целевых переменных категориального класса.

Рассмотрим M деревьев решений: . Задача получить классификатор:

Все что нужно сделать, это взять и усреднить результат. Пусть каждое дерево возвращает K-мерный вектор следующего вида:

Если деревья одинаковы: . Объединение одинаковых деревьев в ансамбль не дает никакого преимущества по сравнению с одним деревом. Для создания разных деревьев существует несколько способов, один из них это Bagging.

Пусть имеется выборка X c N объектами. Необходимо равномерно и с возвращением выбирать N объектов из этой выборки. Это означает, что мы N раз будем случайным образом выбирать объект из исходной выборки, причем каждый объект будет иметь одинаковую вероятность выбыть выбранным, равную 1/N. При каждом выборе будут учитываться все N исходных объектов. Из-за возвращений среди набора могут оказаться повторы. Обозначим новую выборку через . Повторяя процедуру M раз, сгенерируем M подвыборок .

После обучения классификатора на каждой подвыборке получается итоговый классификатор, который представляет собой усреднение ответов от всех алгоритмов. В задаче классификации это соответствует процессу голосования, где каждый алгоритм дает свой прогноз, а итоговое решение определяется путем усреднения или голосования за наиболее часто встречающийся класс.

Эффективность ансамблей достигается благодаря нескольким факторам. Во-первых, базовые алгоритмы обучаются на различных подвыборках, что приводит к получению разнообразных моделей. Эти модели могут иметь различные сильные и слабые стороны, и их ошибки могут взаимно компенсироваться при голосовании или комбинировании предсказаний.

Во-вторых, ансамбли могут быть устойчивы к выбросам или аномалиям в данных. Поскольку каждая обучающая подвыборка содержит только часть исходных объектов, объекты-выбросы могут не попасть в некоторые подвыборки. Это позволяет ансамблю быть менее чувствительным к выбросам и снижает влияние неточностей, вызванных такими объектами.

Комбинируя предсказания разных моделей, ансамбли способны улучшить общую точность и устойчивость предсказаний. Каждый базовый алгоритм вносит свой вклад в окончательное решение, и совместное голосование или агрегация результатов позволяют получить более надежные и уверенные предсказания.

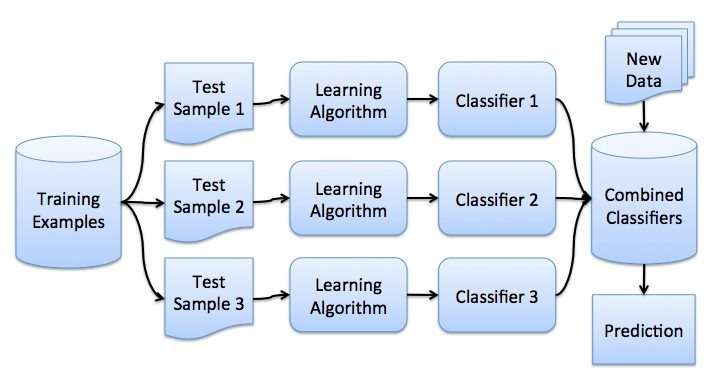


Рисунок 9. Ансамбль моделей

В случайном лесе выбирается случайное подмножество признаков, и лучший признак для разделения узла определяется из подвыборки признаков, в отличие от бэггинга, где все функции рассматриваются для разделения в узле.

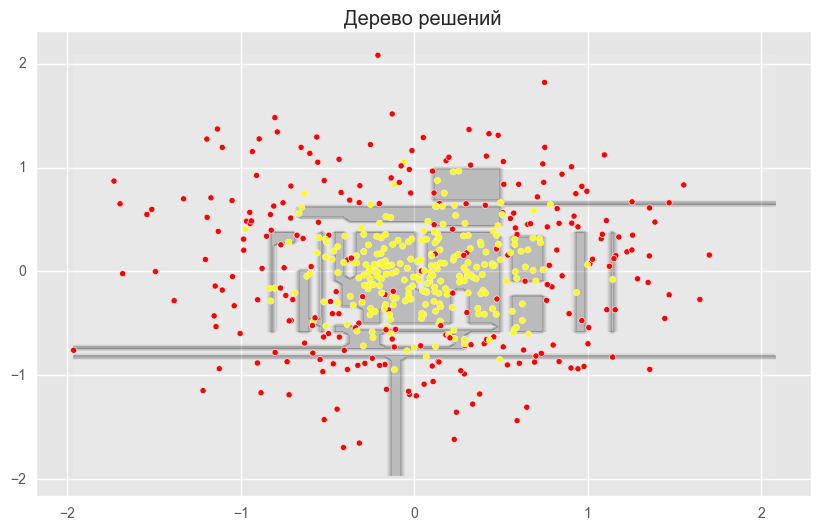


Рисунок 10. Дерево решений

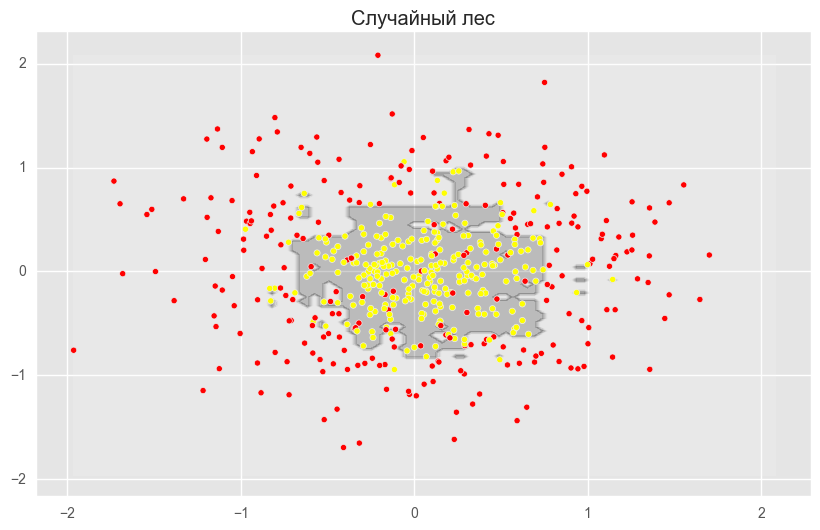


Рисунок 11. Случайный лес

Изображения, представленные на рисунках выше (Рис. 10-11), показывают, что разделяющая граница дерева решений имеет множество "рваных" и острых углов, что может указывать на проблему переобучения и недостаточную обобщающую способность модели. В отличие от этого, у случайного леса граница разделения является более сглаженной, и практически не наблюдается явных признаков переобучения.

## 2.3 Гиперпараметры Random Forest

Важно различать понятия параметров и гиперпараметров модели. Понимание разницы между ними является ключевым для разработки эффективных алгоритмов, которые могут быть успешно применены на практике.

Параметры модели представляют собой конкретные переменные, которые определяют конфигурацию модели. Они представляют собой фактические значения, которые могут быть заданы и изменены, и эти изменения могут влиять на работу модели. Примерами параметров являются веса нейронов в нейронных сетях, коэффициенты в линейных моделях и размерность пространства признаков в алгоритмах кластеризации.

Гиперпараметры представляют собой параметры самого алгоритма, изменение которых не приведет к изменению конечного результата, но может оказать влияние на точность и скорость вычислений. Примером гиперпараметра может служить глубина дерева. Этот параметр задается вручную и определяет, сколько раз будет выполняться разделение данных при построении дерева. Неправильное задание глубины дерева может привести к переобучению или недообучению модели.

Метод случайного леса, реализованный в библиотеки Scikit-Learn представлен двумя классами RandomForestClassifier и RandomForestRegressor.

Полный список параметров случайного леса для задачи классификации:

* n\_estimators — число деревьев в лесу (по умолчанию – 10)
* criterion — функция, которая измеряет качество разбиения ветки дерева (по умолчанию стоит “gini”, можно выбрать “entropy”)
* max\_features — число признаков, по которым ищется разбиение. Вы можете указать конкретное число или процент признаков, либо выбрать из доступных значений: "auto" (все признаки), "sqrt", "log2". По умолчанию стоит "auto".
* max\_depth — максимальная глубина дерева (по умолчанию глубина не ограничена)
* min\_samples\_split — минимальное количество объектов, необходимое для разделения внутреннего узла. Можно задать числом или процентом от общего числа объектов (по умолчанию— 2)
* min\_samples\_leaf — минимальное число объектов в листе. Можно задать числом или процентом от общего числа объектов (по умолчанию — 1)
* min\_weight\_fraction\_leaf — минимальная взвешенная доля от общей суммы весов (всех входных объектов) должна быть в листе (по умолчанию имеют одинаковый вес)
* max\_leaf\_nodes — максимальное количество листьев (по умолчанию нет ограничения)
* min\_impurity\_split — порог для остановки наращивания дерева (по умолчанию 1е-7)
* bootstrap — применять ли бустрэп для построения дерева (по умолчанию True)
* oob\_score — использовать ли out-of-bag объекты для оценки R^2 (по умолчанию False)
* n\_jobs — количество ядер для построения модели и предсказаний (по умолчанию 1, если поставить -1, то будут использоваться все ядра)
* random\_state — начальное значение для генерации случайных чисел (по умолчанию его нет, если хотите воспроизводимые результаты, то нужно указать любое число типа int
* class\_weight – вес каждого класса (по умолчанию все веса равны 1, но можно передать словарь с весами, либо явно указать “balanced”)

Для оптимальной модели, необходимо перебрать каждый значимый гиперпараметр, не меняя остальные по сетке и выбрать хорошую комбинацию. Тогда получится добиться хорошей модели, которая будет хорошо обобщать данные. В программной реализации будет рассмотрен данный подход.

## 2.4 Метрики качества модели

В машинном обучении, метрики представляют собой числовые показатели, которые используются для оценки качества моделей машинного обучения. Они позволяют оценить, насколько хорошо модель выполняет задачу и сравнивать модели между собой. Зная какую метрику необходимо получить, можно гораздо проще представлять, как решается задача.

Рассмотрим частный случай бинарной классификации:

;

Количество верно предсказанных значений, действительно принадлежащих первому классу (TP):

Количество верно предсказанных значений, действительно принадлежащих второму классу (TN):

Количество значений, ложно отнесенных ко второму классу (FN):

Количество значений, ложно отнесенных к первому классу (FP):

В задачах классификации метрики используются для измерения эффективности модели в определении классов для входных данных. Некоторые из наиболее распространенных метрик в задачах классификации:

1. Точность (Accuracy) – является самой простой метрики и показывает долю правильно классифицированных примеров от общего числа примеров. Вычисляется как сумма правильных классифицированных примеров, деленная на общее число примеров. Однако данная метрика может вводить в заблуждение, когда имеется дисбаланс классов или если у ошибок разная цена.
2. Полнота (Recall) – измеряет долю классифицированных положительных примеров от общего числа фактически положительных примеров. Она показывает, насколько хорошо модель обнаруживает положительные примеры. Полнота может быть важна, если ложноотрицательные примеры нежелательны.

Применяется там, где необходимо минимизировать ошибку ложного пропуска (FN)

1. Точность (Precision) – измеряет долю правильно классифицированных положительных примеров от общего числа предсказанных положительных примеров. Она показывает, насколько точно модель классифицирует положительные примеры. Точность может быть важна, если ложноположительные результаты недопустимы.

Применяется там, где необходимо минимизировать ошибку ложного срабатывания (FP)

1. F-мера (F1-score) – является сбалансированной метрикой, которая объединяет точность и полноту. Она вычисляется как гармоническое среднее между точностью и полнотой. F-мера представляет одно число, которое увеличивает и точность и полноту модели.

Применяется там, где необходимо минимизировать ошибку ложного срабатывания (FP) и ложного пропуска (FN) одновременно

В качестве метрики оценки модели будем использовать F1-score метрику (22), так как она предполагает минимизацию ложного срабатывания и минимизацию ложного пропуска одновременно.

Для безошибочной проверки модели используется перекрестный контроль (кросс-валидацию). Кросс-валидация – это методика обучения и оценки модели, которая разбивает данные на несколько секций и обучает несколько алгоритмов на этих секциях. Пример работы кросс-валидации проиллюстрирован на рисунке 12.

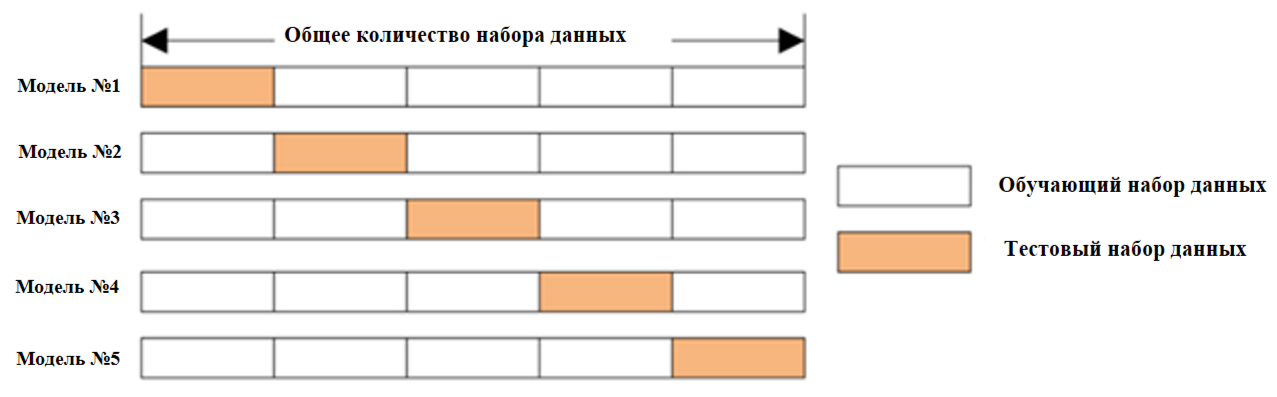


Рисунок 12. Работа кросс-валидации

При таком подходе модель обучается K раз на разных (К-1) поднаборах данных исходного набора данных (белый цвет), а проверяется на одном поднаборе (каждый раз на разном, оранжевый цвет). Получается К оценок качества модели, которые усредняются, выдавая среднюю оценку качества классификации модели на кросс-валидации. При обучении применение данного подхода не требуется, так как модель случайного леса сама по себе применяет подобный алгоритм усреднения, что было упомянуто выше.

## 2.5 Библиотека TSFEL

Для извлечения признаков из временных рядов в данной работе используется библиотека TSFEL [6] на языке программирования Python. TSFEL представляет собой пакет Python, специализирующийся на извлечении признаков из временных рядов. Он предоставляет решения для автоматического извлечения признаков без необходимости тратить значительные усилия на программирование. Библиотека TSFEL автоматически извлекает более 60 различных признаков в различных областях, включая статистическую, временную и спектральную. Она была создана специалистами по обработке данных с целью облегчить работу людям, занимающимся обработкой данных.

Временные ряды передаются в качестве входных данных для основного метода извлечения либо в виде массивов, предварительно загруженных в память, либо в виде сохраненных рядов. Для достижения наилучших результатов необходимо указывать частоту дискретизации исходных данных, так как это напрямую связано с определенными вычисляемыми признаками и процессом расчета временного окна. После извлечения признаков результат сохраняется в виде датафрейма, где каждая строка представляет окно данных с соответствующими значениями извлеченных признаков, сохраненными в соответствующих столбцах.

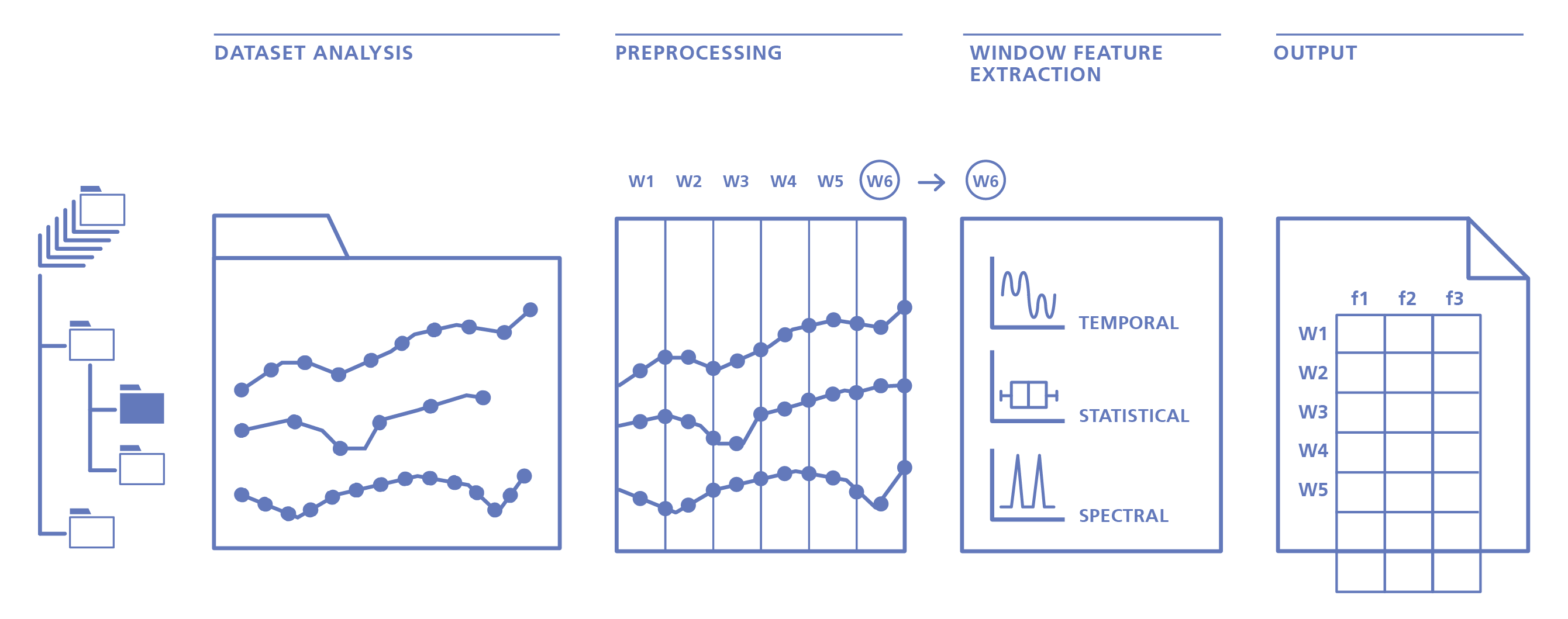


Рисунок 13. Этапы получения признаков с помощью библиотеки TSFEL

## 2.6 Выводы по второй главе

В рамках данный главы были описаны этапы построения моделей машинного обучения. Аргументировано использование ансамблевых моделей для увеличения точности модели. Также описаны концепции построения модели случайного леса и настройки гиперпараметров. Рассмотрены метрики качества моделей. В качестве метрики для сравнения моделей была выбрана метрика F1-score (22), которая позволяет уменьшить ошибку ложного срабатывания и ошибку ложного пропуска одновременно.

В качестве инструмента для извлечения признаков из данных была выбрана библиотека Time Series Feature Extraction Library (TSFEL). Данная библиотека использовалась в исследуемой статье [2] и была выбрана ввиду своего простого использования, которая позволяет проверить большое количество разнообразных признаков на их способность к определению степени мускульной усталости.

# ГЛАВА III ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

## 3.1 Запись данных

Для разнообразия данных были использованы следующие упражнения: приседания, приседания с выпрыгиванием, ходьба по лестнице и бег. Все упражнения выполнялись с учетом шкалы Борга (Таб. 1). Схема измерений представлена на рисунке 14.

Была введена трехуровневая шкала на основе шкалы Борга, которая соответствуют трем уровням усталости: не устал (6-10), средняя усталость (11-16), высокая усталость (16-20).

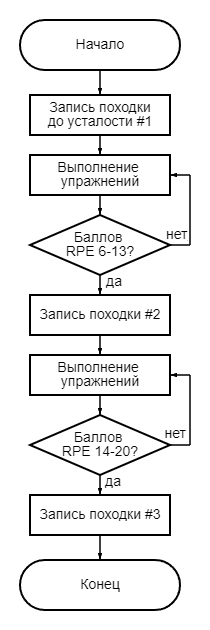


Рисунок 14. Порядок измерений

Сперва испытуемый записывает свою походку до выполнения каких-либо упражнений, после чего выполняет выбранное упражнение до тех пор, пока не оценит свое состояние мускульной усталости по шкале Борга как среднее состояние усталости. После чего выполнялась запись походки. Дальше испытуемый снова выполнял упражнение до тех пор, пока не оценит свое состояние мускульной усталости как очень усталое и тогда повторялась запись походки.

Данные записывались с помощью приложения phyphox (Рис. 15) с частотой дискретизации 50 Гц с линейным трехосным акселерометром и гироскопом.

Phyphox (Physical Phone Experiments) представляет собой инструмент, предназначенный для анализа данных, собранных с трехмерных магнитометров, трехмерных акселерометров и других датчиков. Этот инструмент обеспечивает возможность просмотра данных и выполнения вычислений над ними. Он может быть использован в научных исследованиях, например, для изучения магнитного поля Земли, эффектов гравитации или движения объектов.

Линейный трехосевой акселерометр записывал данные сразу вычитая ускорение свободного падения . В записи набора данных с походкой участвовало 8 человек, как мужского, так и женского пола. Запись производилась в здании и на улице на прямых участках дороги. Среднее время каждой записи составляет 40 секунд. Было записано порядка 20 записей для каждого уровня усталости. Каждая запись содержит столбец со временем и показаниями датчиков по трем осям (Рис. 16).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 15. Приложение phyphox для записи данных с датчиков на телефоне

Изображение выглядит как стол

Автоматически созданное описание Изображение выглядит как текст, кроссворд

Автоматически созданное описание

Рисунок 16. Записанные показания датчиков с трехосевого акселерометра и гироскопа через приложение phyphox

## 3.1 Предобработка датасета и извлечение признаков

Для работы с записанным набором данных была написана функция parsing (Рис. 38 и 40). С помощью аргумента folder\_name передается путь к записанному набору данных. Функция удаляет первые и последние пять секунд записи, находит модуль ускорения и угловой скорости, объединяет записи и удаляет выбросы по квартилям. Пример работы функции представлен на рисунке 20.

Для получения хороших предсказаний необходимо обработать записанный набор данных. Для этого в каждой записи были удалены первые и последние 5 секунд записи, что соответствует времени, за которое человек убирает телефон в карман и достает его чтобы остановить запись.

Удаление выбросов производилось с помощью межквартильного диапазона (IQR). Для этого строится коробчатая диаграмма (Рис. 17), так же ее еще называют ящик с усами. Такой вид диаграммы в удобной форме показывает медиану, нижний и верхний квартили, минимальное значение выборки и выбросы. Расстояние между различными частями ящика позволяют определить степен разброса относительно среднего значения (дисперсии) и асимметрии данных и выявить выбросы.

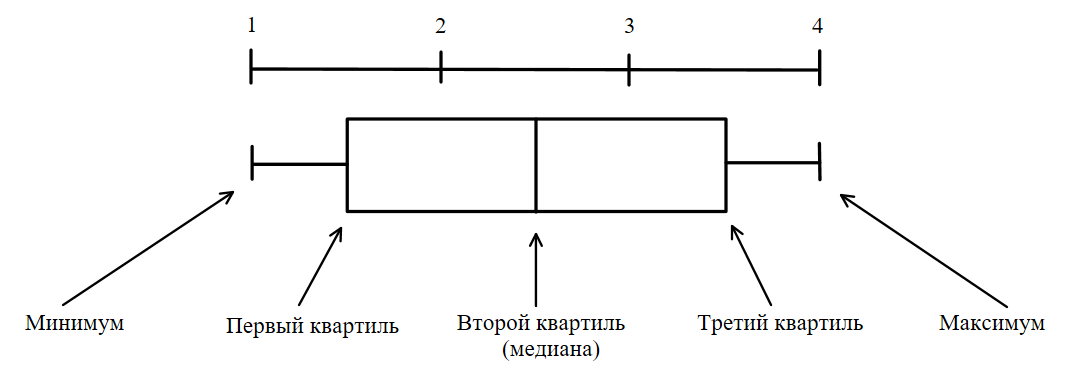


Рисунок 17. Определение межквартильных диапазонов

Границами ящика служат первый и третий квартили (25-й и 75-й процентили соответственно), линия в середине ящика – медиана (50-й процентиль). Концы усов – края статистически зависимой выборки (без выбросов), они определяются следующим образом:

где – нижняя граница уса, – верхняя граница уса, - первый квартиль, – третий квартиль, k – коэффициент, наиболее частое значение которого 1.5.

Данные, выходящие за границы усов (выбросы), отображаются на графике в виде точек или маленьких кружков. На рисунке 18 сверху изображена коробчатая диаграмма, а снизу функция вероятности нормального распределения. На межквартильный диапазон приходится 50% данных, на усы и приходится по 24.65% данных. Все что находится дальше усов, считается выбросами.

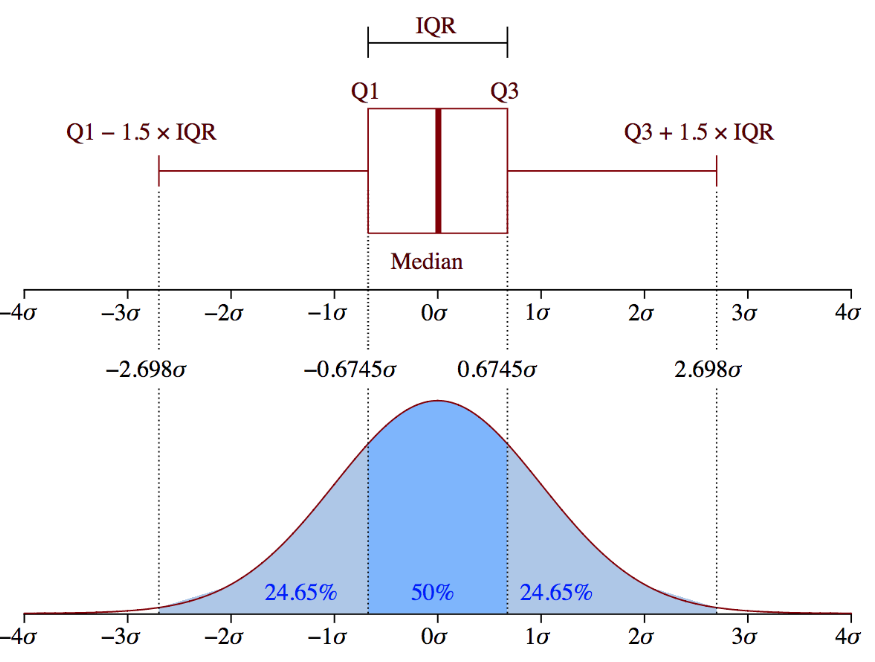


Рисунок 18. Прямоугольная диаграмма (с межквартильным диапазоном) и функция плотности вероятности нормального распределения N (0, σ2)

Для записанных данных получается следующая картина, на рисунке 19 представлены данные до удаления выбросов (слева) и после (справа).

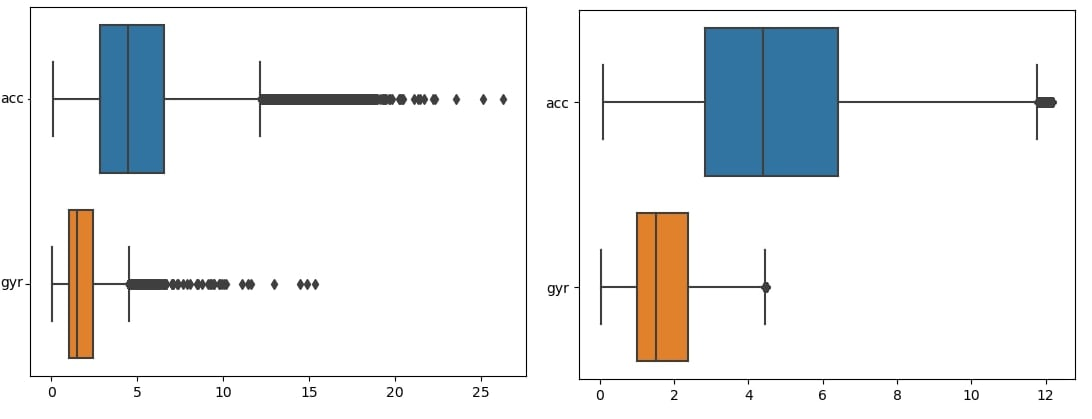


Рисунок 19. Удаление выбросов для данных с датчиков трехосевого акселерометра и гироскопа по квартилям

После объединения наборов данных и удаление выбросов, получаются три набора данных df1, df2, df3, где в каждом столбце находятся данные с трехосевого акселерометра и гироскопа для трех уровней усталости (Рис. 20).

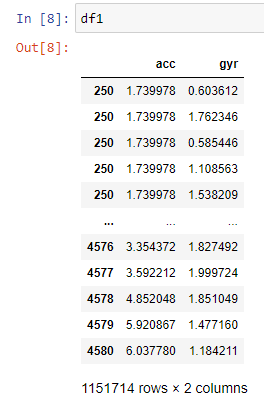


Рисунок 20. Очищенный набор данных с трехосевого акселерометра и гироскопа для усталой походки полученной функцией parsing

Дальше была написана функция – features (df, feature\_class, fs=50, window\_size=250) (Рис. 39, 41). Эта функция принимает на вход объединенный набор данных с определенной походкой (не усталая, средняя усталость, высокая усталость), параметр feature\_class отвечает за тип признаков, который будет извлечен из этих данных, с помощью этого параметра можно извлечь спектральные, статистические и временные признаки. Для того чтобы эти признаки были правильно вычислены, указывается частота дискретизации, при которой производилась запись данных. В нашем случае частота дискретизации в функции по умолчанию стоит 50Гц, что соответствует той частоте, при которой производилась запись набора данных. С помощью параметра window\_size можно указать для какой длины временного окна будут вычисляться признаки. В исследуемой статье «MLFatigueDetection Machine Learning Based Walking Fatigue Detection» [2] указывается, что лучшие результаты достигались при окне в 5 секунд, поэтому необходимо установить параметр window\_size равным 250 при частоте дискретизации равной 50Гц, что будет соответствовать пятисекундному временному окну.

Так как извлекается очень большое количество признаков, не стоит усложнять модель большим ее количеством, к тому же существует достаточно много пропущенных значений или просто нулевых колонок. Все это необходимо отфильтровать. Поэтому после извлечения признаков, функция features находит попарную корреляцию между всеми извлеченными признаками и по установленному пороговому уровню корреляции удаляет те признаки, которые имеют высокую корреляцию. Это делается из тех соображений, что нет необходимости использовать два признака, которые зависели бы друг от друга, в таком случае, достаточно ограничится одним признаком из этой пары. По аналогичным соображениям производится удаление признаков с низкой дисперсией.

Таким образом, указывая параметр feature\_class для всех трех уровней усталости получается 9 наборов данных со статистическими, временными и спектральными признаками (Рис. 21).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 21. Не очищенный набор данных статистических признаков для не усталой походки

После объединения всех набор данных, возможно существование столбцов с пропущенными значениями. Для обработки таких столбцов были найдены столбцы признаков, в которых пропущено больше 30% данных и удалены. Для всех остальных столбцов пропущенные значения были заменены на средние значения (Рис. 22).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 22. Очищенный и объединенный набор данных со всеми признаками для не усталой походки

Дальше для каждого уровня усталости были проставлены метки: 0 – не усталая походка, 1 – средняя усталость, 2 – высокая усталость.

Получается следующее распределение по классам (таблица 5), видно, что записей для не усталой походки больше, чем остальных, то есть у нас несбалансированный датасет. Это необходимо учитывать при обучении модели. Вектор признаков представляет из себя совокупность признаков, относящихся к одному объекту.

Таблица 5. Распределение по классам

|  |  |
| --- | --- |
| Уровень усталости | Количество вектора признаков |
| 0 | 4606 |
| 1 | 2167 |
| 2 | 1922 |

## 3.2 Обучение модели

С помощью библиотеки sklearn и функции train\_test\_split можно разделить данные на тестовые и тренировочные. Для тренировочных будет использоваться 70% данных, а остальные 30% будут использоваться в качестве данных для тестирования. Для этого в функцию train\_test\_split передается набор данных без меток и отдельный набор данных с метками, также там указывается как будет разделен датасет.

Первым делом была проверена классическая модель случайного леса из библиотеки sklearn, где все параметры выставлены по умолчанию.

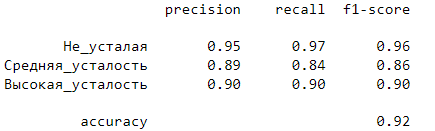


Рисунок 23. Метрики качества модели по умолчанию

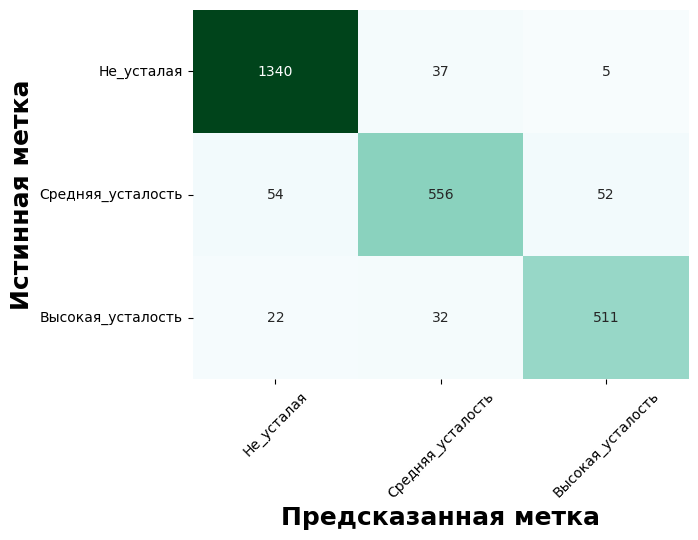


Рисунок 24. Матрица ошибок для модели по умолчанию

Посмотрим на важность признаков в модели. То есть на те признаки, по разбиению выборки которых прирост информации оказался выше.

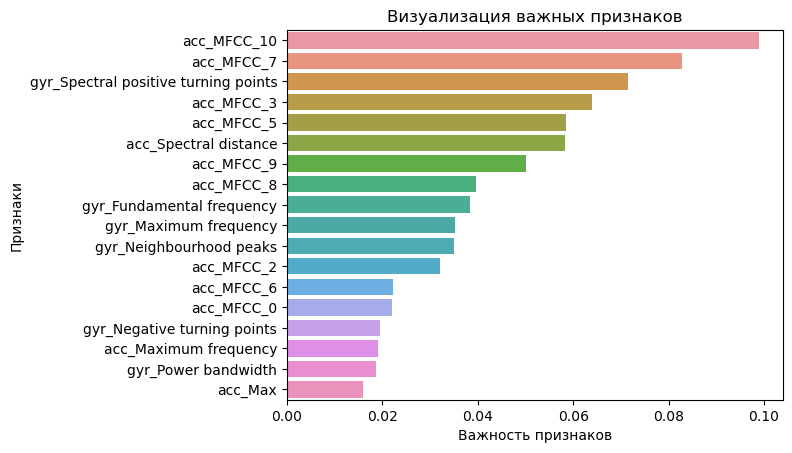


Рисунок 25. Визуализация важности признаков в модели случайного леса

Таблица 6. Числовые значения важности признаков

|  |  |
| --- | --- |
| Признак | Числовое значение важности (0,1) |
| acc\_MFCC\_10 | 0.099013 |
| acc\_MFCC\_7 | 0.082854 |
| gyr\_Spectral positive turning point | 0.071499 |
| acc\_MFCC\_3 | 0.063897 |
| acc\_MFCC\_5 | 0.058580 |
| acc\_Spectral distance | 0.058360 |
| acc\_MFCC\_9 | 0.050045 |
| acc\_MFCC\_8 | 0.039728 |
| gyr\_Fundamental frequency | 0.038313 |
| gyr\_Maximum frequency | 0.035164 |
| gyr\_Neighbourhood peaks | 0.034921 |
| acc\_MFCC\_2 | 0.032129 |
| Acc\_MFCC\_6 | 0.022251 |
| acc\_MFCC\_0 | 0.021998 |
| gyr\_Negative turning points | 0.019516 |
| acc\_Maximum frequency | 0.019072 |
| Gyr\_Power bandwidth | 0.018629 |
| Acc\_Max | 0.015886 |

## Настройка гиперпараметров

Настройку гиперпараметров лучше всего начинать с помощью алгоритма RandomizerSearchCV. Так как перебирать все комбинации гиперпараметры очень долго и трудозатратно, можно грубо оценить их широкие диапазоны.

Делается это следующим образом: создается некий объект param\_dist, в котором указывается диапазон значений для каждого гиперпараметра, чтобы потом исследовать их в процессе работы. Затем мы создаем объект rs с помощью функции RandomizedSearchCV(). В эту функцию мы передаем модель RF, объект param\_dist, число итераций и число кросс-валидаций, которые нужно выполнить.

Гиперпараметр n\_jobs позволяет указать, сколько процессорных ядер использовать для работы модели. Если установить n\_jobs в значение -1, модель будет работать быстрее, так как будут использоваться все ядра процессора.

Оптимизировались основные гиперпараметры модели, которые описаны выше, это: n\_estimators – число деревьев в случайно лесу, max\_features – число признаков для выбора расщепления, max\_depth - максимальная глубина дерева, min\_samples\_split – минимальное число объектов, необходимое для того, чтобы узел дерева смог расщепиться, min\_samples\_leaf – минимальное число объектов в листьях и bootstrap – использование для построения дерева подвыборки с возвращением.

При использовании значений n\_iter = 100 и cv = 3 создается 300 RF-моделей, выбирая случайные комбинации гиперпараметров, которые были указаны ранее. Чтобы узнать о наборе параметров, которые создали наилучшую модель, можно обратиться к атрибуту best\_params\_. Однако, на данном этапе это может не предоставить наиболее интересные данные о диапазонах параметров, которые стоит исследовать в следующем этапе оптимизации. Чтобы определить, в каком диапазоне значений продолжить поиск, можно легко получить таблицу (датафрейм), содержащую результаты работы алгоритма RandomizedSearchCV и визуализировать ее. Результаты представлены на рисунке 26. В столбчатых графиках на оси X расположены значения гиперпараметров, а на оси Y – средние значения, показываемые моделями. Данная методика позволяет сказать какие гиперпараметры показывают лучшие результаты.

Лучшие средние значения модели для n\_estimators показываются на значениях 500, 700 и 900. Для min\_samples\_split значение 23 показывает наилучший результат, имеется смысл исследовать значения около 23.

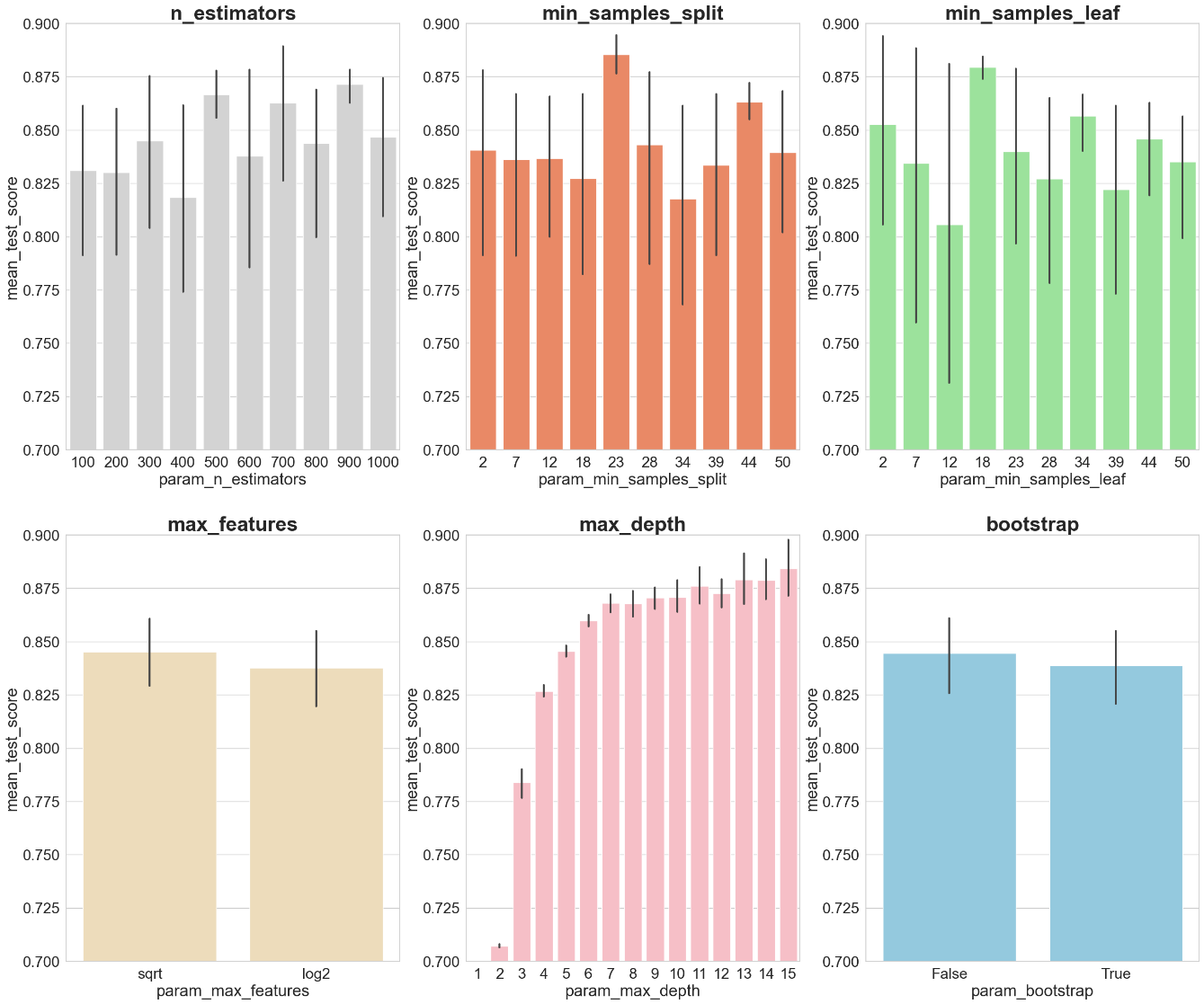


Рисунок 26. Средний балл за тест для гиперпараметров

Значение min\_samples\_leaf имеет смысл исследовать в диапазоне от 16 до 18. Гиперпараметр max\_features показывает лучший вариант для значения равного квадратному корню из числа признаков. Максимальная глубина дерева (max\_depth) имеет тенденцию увеличения при росте параметра, это может быть связано с переобучением модели и требуется посмотреть на график ошибки валидационной и тренировочной выборки на кросс-валидации. Также для гиперпараметра bootstrap лучшее значение показывает значение False. В сумме получается около 1620 моделей.

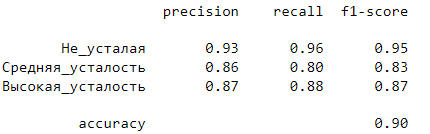


Рисунок 27. Метрики после оптимизации гиперпараметров путем поиска по сетке

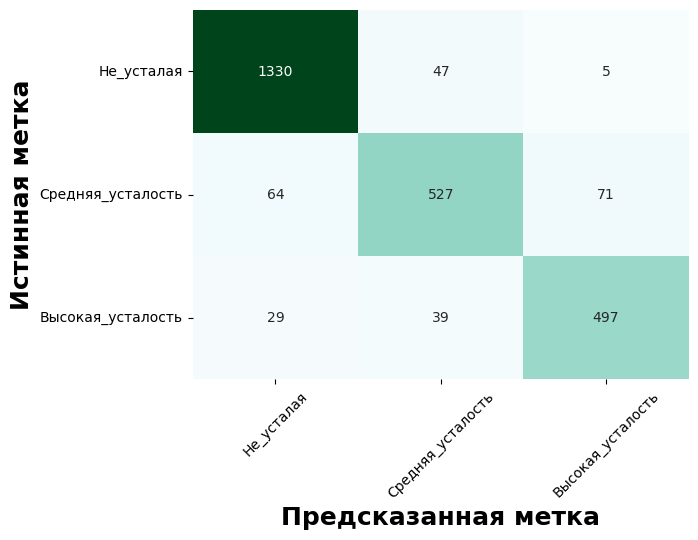


Рисунок 28. Матрица ошибок после оптимизации гиперпараметров путем поиска по сетке

По полученным результатам видно, что уменьшит ошибки не удалось, а наоборот, они даже увеличились. В данном случае, скорее всего ошибка возрастает из-за переобучения. Для решения этой проблемы оптимизируем гиперпараметры ориентируясь на графики ошибок обучающей и валидационной выборки с помощью кросс-валидации. Для этого будем менять один гиперпараметр, оставляя все остальные гиперпараметры по умолчанию.

Из рисунка 29 видно, что около значения n\_estimators = 150 ошибка на валидационной выборке начинает расти, тогда как на тренировочном наборе она уменьшается. Это означает, что после этого порога происходит переобучение модели, то есть модель подстраивается под данные и начинает плохо обобщать данные. Это связанно непосредственно с тем, что модель начинает повторять шумы в данных.

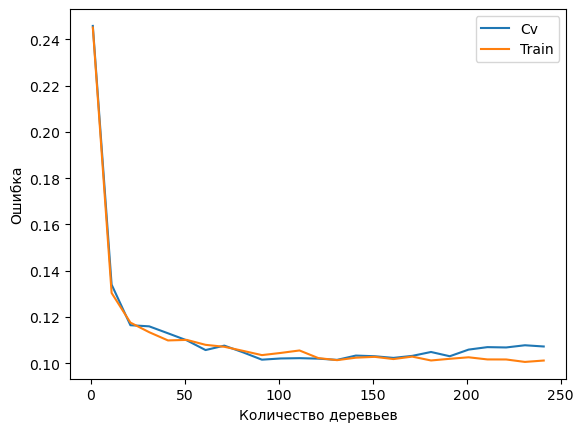


Рисунок 29. Ошибка F1-score на тренировочном и валидационном наборе при кросс-валидации для параметра n\_estimators

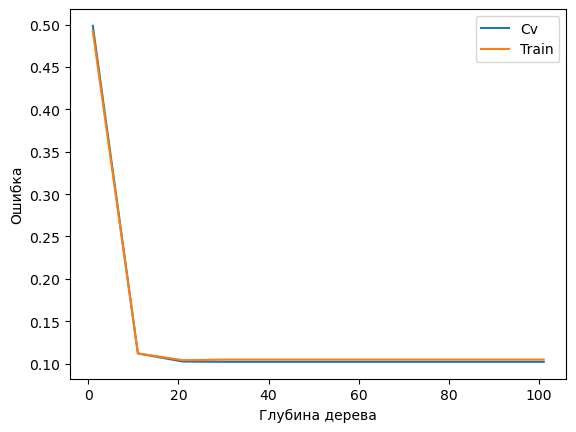


Рисунок 30. Ошибка F1-score на тренировочном и валидационном наборе при кросс-валидации для параметра max\_depth

В данном случае (Рис. 30) переобучение наступает с порогового значения max\_depth равным 20. Таким образом глубину дерева лучше всего устанавливать равным 19.

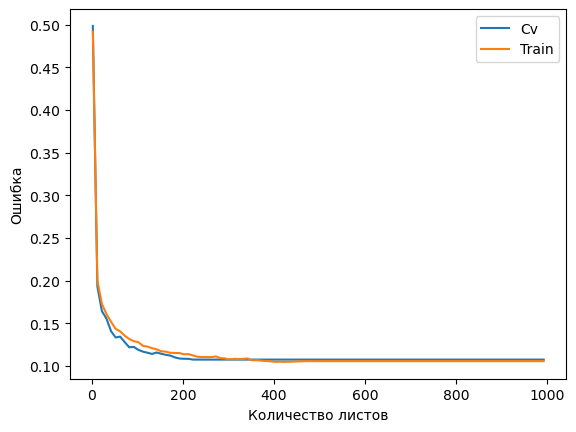


Рисунок 31. Ошибка F1-score на тренировочном и валидационном наборе при кросс-валидации для параметра max\_leaf\_nodes

В случае с неограниченном параметре количества листов в дереве его необходимо ограничивать, чтобы не получить переобучение. На рисунке 31 переобучение происходит при значении max\_leaf\_nodes равным 250.

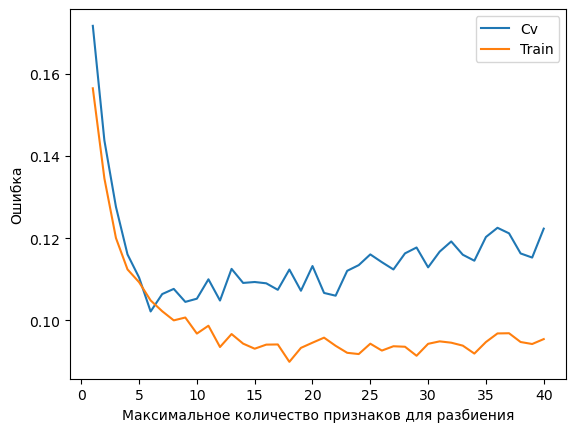


Рисунок 32. Ошибка F1-score на тренировочном и валидационном наборе при кросс-валидации для параметра max\_features

Для значения числа признаков для разбиения на рисунке 32 оптимальным значением является число 4. Интересно отметить, что это значение лежит ближе к логарифму от числа признаков, а не к квадратному корню, хотя полученные результаты (Рис. 26) говорят об обратном.

Такими гиперпараметрами как min\_samples\_split и min\_samples\_leaf получить лучший результат на кросс-валидации не получается (Рис. 33)

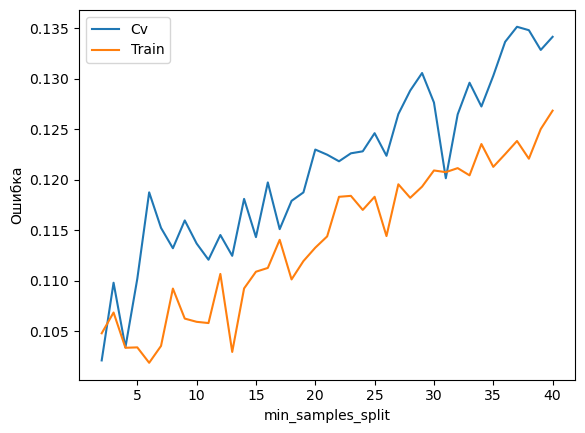
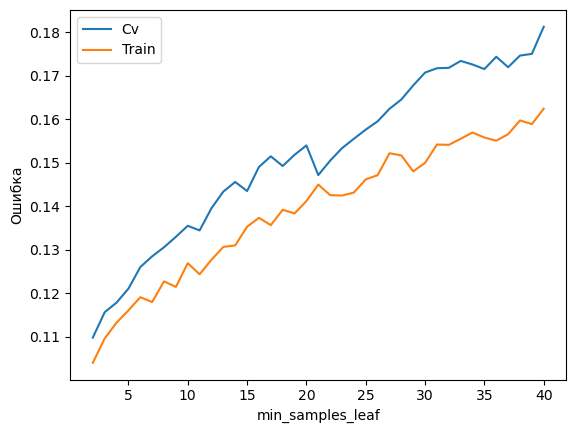
 

Рисунок 33. Ошибка F1-score на тренировочном и валидационном наборе при кросс-валидации для параметров min\_samples\_split (слева) и min\_samples\_leaf (справа)

После определения оптимальных значений гиперпараметров модели нужно передать их в модель и оценить ее.

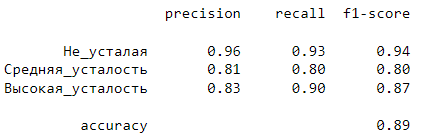


Рисунок 34. Метрики модели RF после ручной оптимизации гиперпараметров модели

Вместо увеличение метрики F1-score по сравнению с базовой моделью, получилось лишь незначительно увеличить метрику precision для не усталой походки, остальные же метрики либо остались неизменны, либо стали чуть хуже. Однако даже полученных значений является достаточно, благодаря тому что получилось однозначно избавиться от переобучения модели и можно точно говорить о том, что модель покажет себя хорошо на новых данных.

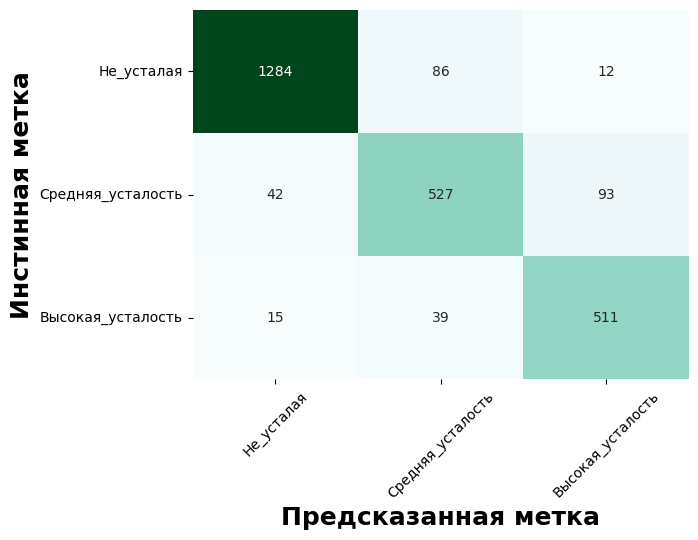


Рисунок 35. Матрица ошибок для RF после ручной оптимизации гиперпараметров модели

После того как было выяснено, какие гиперпараметры являются оптимальными для модели, проводится окончательное обучение. Для этого в итоговую модель с оптимизированными параметрами передается весь датасет с тренировочным и валидационном набором данных. Далее в главе 4 будут описаны результаты полученной модели.

## 3.3 Структурные схемы



Рисунок 36. Структурная схема обучения модели



Рисунок 37. Структурная схема работы модели на сервере



Рисунок 38. Структурная схема работы функции parsing



Рисунок 39. Структурная схема работы функции fuatures

## 3.4 Блок-схемы

Функция parsing (Рис. 38). Предназначена для извлечения файлов с записанной походкой, отчистки и их объединении.

Входные данные: folder\_name – название папки с определенным уровнем походки, fs –частота дискретизации, при которой записывался набор данных.

Выходные данные: df - очищенный набор данных, состоящий из двух столбцов с очищенными значениями модулей трехосевого акселерометра и гироскопа (Рис. 20).

Язык python. Библиотеки: glob, numpy (np), pandas (pd). Из библиотеки glob используется функция glob() которая возвращает список путей, соответствующих шаблону pathname. Из библиотеки numpy используются математические операции. Из библиотеки pandas используется класс pd.Dataframe для работы с данными в виде таблицы, функции и методы для работы с объектами класса pd.Dataframe.

1. Описание библиотечных классов, объектов, методов, полей и функций. Пусть df - объект класса pd.Dataframe табличного типа данных, filepath – путь к директории с набором данных, folder\_name - название папки с не усталой, средней и высокой усталости, fs – частота дискретизации, files\_acc[k] и files\_gyr[k] – название csv файлов с походкой для трехосевого акселерометра и гироскопа соотвественно.

1.1 glob.glob(‘filepath’) – возвращает название файлов по указанному шаблону в указанной директории. Если шаблон не указан, то возвращает название всех файлов в директории.

1.2 pd.read\_csv(‘filepath’ + ‘files\_acc[k]’) – считывает csv файл и возвращает объект класса DataFrame.

1.3 df.iloc – возвращает индексирование на основе целочисленного местоположения для выбора по положению.

1.4 df.append(df\_old) – добавляет снизу в df столбцы из df\_old.

1.5 df.columns(‘name\_columns’) – назначает df название колонок ‘name\_columns’.

1.6 df.columns = [‘acc’] – добавляет название столбцам в таблице с данными df.

1.7 df.quantile(q) – возвращает значения в заданном квантиле по запрошенному значению q.

1.8 df.apply(function) – применяет функцию function вдоль оси df.

1.9 stats.iqr – возвращает межквартильный диапазон данных по указанной оси.

1.10 df.any(axis=1) – проверяет, является ли какой-либо элемент в столбце истинным. Возвращает False, если в столбце из набора данных df нет хотя бы одного элемента, который является True.

1.11 pd.merge(df1, df2) – объединяет df1 и df2.

Функция features (Рис. 39). Предназначена для вычисления признаков.

Входные данные: df – набор данных, полученных с помощью функции parsing, feature\_class – параметр, отвечающий за тип вычисляемых признаков, fs – частота дискретизации, window\_size – размер окна, на которые будет разделен набор данных, для каждого окна будут вычислены признаки.

Выходные данные: набор данных с вычисленными признаками (Рис. 21).

Язык python. Библиотеки: tsfel, pandas (pd), sklearn. Из библиотеки tsfel используются функции get\_features\_by\_domain для выбора типа вычисляемых признаков, time\_series\_features\_extractor для вычисления самих признаков, correlated\_features для вычисления корреляции между вычисленными признаками. Из библиотеки pandas используется класс pd.Dataframe для работы с данными в виде таблицы, функции и методы для работы с объектами класса pd.Dataframe. Из библиотеки sklearn используется класс VarianceThreshold который удаляет все признаки, дисперсия которых не соответствует заданному порогу.

1. Описание библиотечных классов, объектов, методов, полей и функций. Пусть df - объект класса pd.Dataframe табличного типа данных, feature\_class – класс вычисляемых признаков, fs – частота дискретизации, window\_size – размер окна, по которому будет разделен набор данных, cfg – словарь с конфигурациями, задает какие признаки будут вычислены.
   1. tsfel.get\_features\_by\_domain(feature\_name) возвращает файл конфигурации, словарь, содержащий какие функции и гиперпараметры будут вычисляться. Передавая в функцию feature\_name = ‘statistical’, ‘temporal’, ‘spectral’ можно вычислить статистические, временные или спектральные признаки соответственно.
   2. tsfel.time\_series\_features\_extractor(cfg,df) – возвращает указанные словарем тип признаков.
   3. tsfel.correlated\_features(df, threshold) – возвращает признаки из df у которых корреляция превышает заданного значением threshold.
   4. VarianceThreshold(threshold) – отбирает признаки у которых дисперсия меньше заданного значения threshold.
   5. fit\_transform(df) – центрирование df.
   6. get\_support() – возвращает маску или целочисленный индекс выбранной функции.
   7. pd.DataFrame() - переводит значение в объект табличного типа данных.
   8. preprocessing.normalize(df) – масштабирует входные данные к единичной норме.
   9. df.drop(labels, axis=1) – удаление из df указанных меток labels из столбцов.

Функция isMuscle (Рис. 37). Предназначена для работы на сервере. Принимает на входе данные с трехосевого акселерометра и гироскопа, вычисляет признаки и классифицирует походку.

Входные данные: acc\_filepath – данные с трехосевого акселерометра, gyro\_filepaht – дынные с гироскопа, fs – частота дискретизации, window\_size – размер окна, на которые будет разделен набор данных.

Выходные данные: возвращает строковый параметр result, который является результатом классификации: 0 – не усталая походка, 1 – средняя усталость, 2 – высокая усталость.

Язык python. Библиотеки: numpy (np), pandas (pd), tsfel, sklearn, pickle. Библиотека pickle используется для сохранения и загрузки обученной модели.

1. Описание библиотечных классов, объектов, методов, полей и функций. Пусть df - объект класса pd.Dataframe табличного типа данных, acc\_filepath и gyro\_filepath – входные аргументы функции isMuscle для трехосевого акселерометра и гироскопа соотвественно, data – неразмеченный вектор признаков, который нужно предсказать, ‘Shapka.xlsx’ – Excel файл с названиями признаков участвующих при обучении модели.
   1. pd.DataFrame() - переводит значение в объект табличного типа данных.

1.2 df.columns = [‘acc’] – добавляет название столбцам в таблице с данными df.

1.3 df.quantile(q) – возвращает значения в заданном квантиле по запрошенному значению q.

1.4 df.apply(function) – применяет функцию function вдоль оси df.

1.5 stats.iqr – возвращает межквартильный диапазон данных по указанной оси.

1.6 df.any(axis=1) – проверяет, является ли какой-либо элемент в столбце истинным. Возвращает False, если в столбце из набора данных df нет хотя бы одного элемента, который является True.

1.7 pd.merge(df1,df2) - объединяет df1 и df2.

1.8 features(df, ‘feature\_class’) – вычисляет и возвращает в виде объекта класса pd.DataFrame признаки для df. Передавая в функцию feature\_name = ‘statistical’, ‘temporal’, ‘spectral’ можно вычислить статистические, временные или спектральные признаки соответственно.

1.9 pd.concat(df1,df2) – объединяет наборы данных df1 и df2.

1.10 preprocessing.normalize(df) – масштабирует входные данные к единичной норме.

1.11 pd.read\_excel(‘Shapka.xlsx’) – считывает Excel файл.

1.12 df.columns.intersection() – фильтрует набор данных df по переданному значению.

1.13 pickle.load(file) – загружает обученную модель.

1.14 classifier.predict(data) – предсказывает и возвращает метку у неразмеченного вектора признаков.

1.15 str() – приводит набор данных в строковый формат.



Рисунок 40. Блок-схема функции parsing



Рисунок 41. Блок-схема функции features





Рисунок 42. Блок-схема работы алгоритма на сервере

## 3.5 Выводы по третьей главе

В рамках данной главы были описаны функции способные объединять различные записи о походках и функции для очищения датасета по межквартильному диапазону. Рассмотрены основные принципы построения моделей машинного обучения, подробно объяснено как происходит построение случайного леса и почему использовать ансамблевые модели лучше.

Были получены результаты для стандартной модели случайного леса из библиотеки sklearn (Рис. 23 и 24), для модели, где гиперпараметры подбирались с помощью сетки значений (Рис. 27 и 28), а также рассмотрено как подбирать оптимальные гиперпараметры путем ручного поиска на кросс-валидации (Рис. 34 и 35).

Лучшие значения по метрики F1-score показала стандартная модель из библиотеки sklearn, однако использовать было решено модель, полученную путем ручного поиска. Это связано непосредственно с лучшими показаниями на практике и тем фактом, что у стандартной модели нет ограничения по глубине дерева, что приводит к переобучению модели. Такая модель лучше обобщала данные и лучше предсказывала состояние мускульной усталости.

Также благодаря использованию модели случайного леса получилось выяснить какие признаки являются наиболее важными для задачи определения степени мускульной усталости (Рис.25). Одними из самых важных оказались признаки MFCC (мел-кепстральные коэффициенты) для акселерометра.

Коэффициенты MFCC (Mel Frequency Cepstral Coefficient) – это коэффициенты, которые в большей степени используются для распознавания и анализа речевых сигналов. Однако также могут использоваться и для анализа других видов сигналов. Основной их смысл заключается в выделении хороших признаков для идентификации лингвистического содержания. Вычисление мел-частотных кепстральных коэффициентов включается в себя следующие шаги:

1. Исходный сигнал разделяют на фреймы. Речевой сигнал записывается в виде:

где N – размер фрейма или длина окна, – j-ый фрейм. Следующие шаги будут применятся для каждого отдельного фрейма.

1. Сигнал умножается на оконную функцию Хемминга:

К полученному результату применяют дискретное преобразование Фурье:

1. Вычисляется периодограмма для каждого фрейма по следующей формуле:
2. Вычисляется набор мел-фильтров. Для этого треугольные фильтры умножаются на периодограмму и суммируются. В результате получаются энергии набора фильтров. Каждый треугольный фильтр моделируется с помощью функции следующего вида:

где m – это число фильтров, f(m) – это параметр функции, отписывающей шкалу Мел, которая соотносит воспринимаемую частоту с фактической измеренной частотой.

1. Полученные энергии логарифмируются. Получается некоторый набор коэффициентов, которые еще не являются MFCC:
2. Далее делается косинусное преобразование и получаются коэффициенты MFCC:

MFCC коэффициенты могут предоставить информацию о различных аспектах сигнала, таких как его спектральная форма, ритмические особенности и характеристики гармонических структур. Они широко используются в задачах распознавания речи, идентификации диктора и других аналогичных областях. При анализе данных с акселерометра и гироскопа, MFCC коэффициенты могут помочь в извлечении информации о движении, особенно в задачах, связанных с распознаванием жестов, определения позы или обнаружением активности.

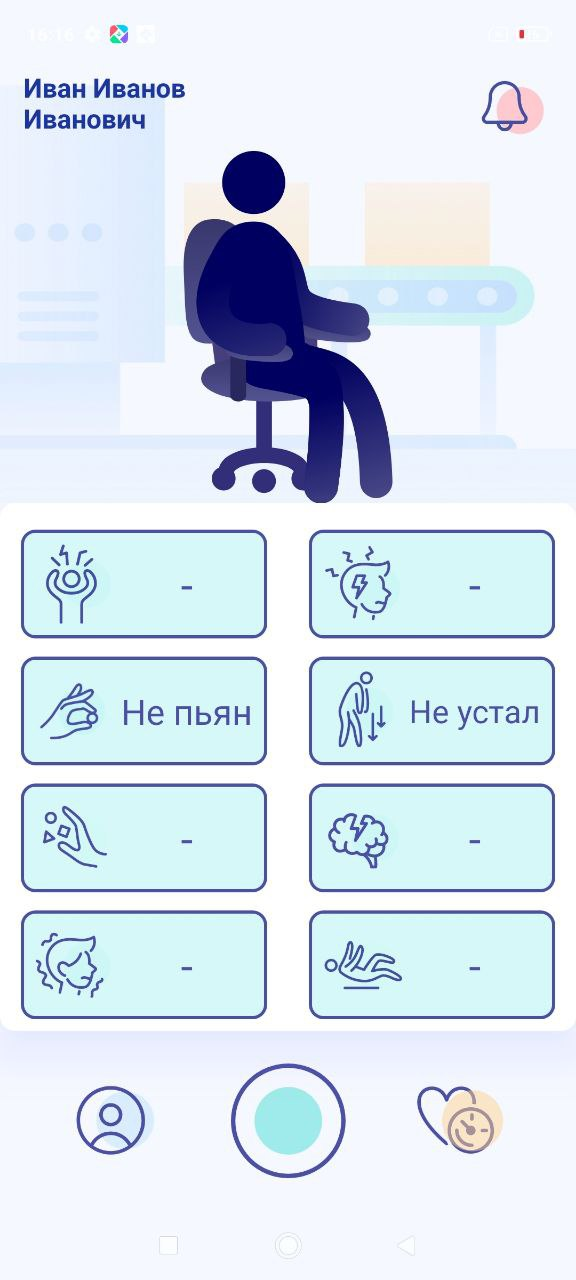
# ГЛАВА IV РЕЗУЛЬТАТЫ ИСПОЛЬЗОВАНИЯ И ВНЕДРЕНИЯ

Для испытаний использовалась клиент-серверная система. В качестве сервера выступал персональный компьютер Forsite(Intel® Core™ i9-9920X, NVIDIA Quadro RTX 6000; ОП 64 Гб; SSD 1Тб; клавиатура; мышь; монитор). Серверный персональный компьютер работал с СУБД PostgreSQL, в таблицах которого выполнялись записи результатов измерений испытуемых посредством выполнения SQL запросов, получаемых от клиентского устройства. В качестве клиентских устройств использовались следующие модели смартфонов: Huawei nova 8i, Samsung A8 (2018), Xiaomi Mi 9 Lite, Redmi Note 9S, POCO X3 Pro, Samsung SM-A515F. Клиентские устройства, которыми пользовались испытуемые содержали установленное приложение.

Так как при обучении использовался модуль ускорения без ускорения свободного падения, то данные, которые идут на сервер и обрабатываются должны быть аналогичными. Поэтому, когда приходят данные с датчиков, необходимо вычесть ускорение свободного падения по формуле 32:

Ввиду того, что при классификации используются не все извлекаемые функции на этапе обучения, при работе алгоритма на сервере необходимо в обученный классификатор передавать только те признаки, которые учувствовали при обучении. Для этого после обучения модели создается и сохраняется шапка таблицы с полученного набора с признаками. При работе алгоритма после вычисления признаков происходит фильтрация по заранее сохраненному списку имен столбцов и таким образом в классификатор поступают только нужные признаки.

После проверки алгоритм был встроен в приложение АМАЛЬТЕЯ, которое разрабатывалось в составе научной группы КФУ (Рис. 43). Данные с датчиков на смартфоне пользователя отправляются на сервер, где происходит вычисление признаков и классификация по ним (Рис. 44). Уровень усталости определяется каждые 5 секунд, результаты накапливаются в течение одной минуты, к пользователю отправляется результат частота реализации которого наибольшая за одну минуту.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 43. Работа алгоритма в приложении АМАЛЬТЕЯ

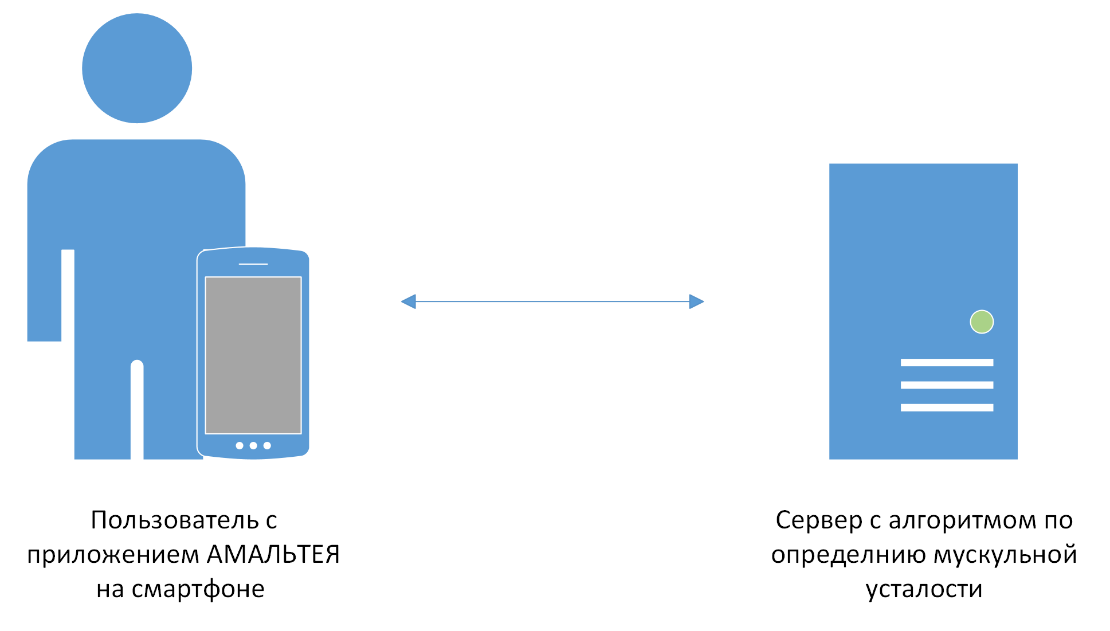


Рисунок 44. Связь пользователь – сервер

Таблица 7. Матрица ошибок

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Предсказанное значение | | | |
| Истинное значение |  | Не устал | Средняя усталость | Высокая усталость |
| Не устал | 27 | 3 | 0 |
| Средняя усталость | 3 | 25 | 2 |
| Высокая усталость | 0 | 4 | 26 |

Таблица 8. Метрики

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Уровень усталости | Accuracy | Precision | Recall | F1 Score |
| Не устал | 93% | 90% | 90% | 90% |
| Средняя усталость | 87% | 78% | 83% | 81% |
| Высокая усталость | 93% | 93% | 87% | 90% |

Для тестирования разработанной модели были проведены испытания, включающие по 30 испытаний для каждого уровня усталости. Каждое испытание представляло собой процесс сбора данных с датчиков на смартфонах пользователей, их отрпавку на сервер и последующую классификацию уровней усталости. Всего было проведено 90 испытаний, это позволило получить статистически значимый объем данных для каждой категории усталости и достоверно оценить производительность модели.

После проведения испытаний и классификации уровня усталости, полученные результаты были использованы для оценки модели с использованием матрицы ошибок (таблица 7) и метрик (таблица 8), что позволяет получить обобщенную оценку производительности модели и сравнить ее с заранее установленными стандартами и требованиями. Среднее значение F1-score для трех уровней усталости составило 87%.

# ЗАКЛЮЧЕНИЕ

В рамках данной работы был проведен анализ существующих методов детектирования степени мускульной усталости по данным датчиков акселерометра и гироскопа. Основополагающее большинство решений основывается на применении нейронный сетей и метода опорных векторов. Одним из дополнительных требований выдвинутых к поставленной задачи являлось использование алгоритма, когда смартфон находится в кармане брюк. Рассмотренные решения подтверждают возможность использования смартфона в кармане в качестве такого датчика. Однако большинство полученных результатов далеки для применения в промышленной среде из-за низкой точности и недостаточной надежности.

Одной из главных проблем, выявленных в процессе анализа существующих методов, является ограничение точности классификации уровней усталости. В работе [1] была использована двухуровневая шкала усталости, что позволило достичь точности 91%. Однако для более детализированной классификации уровней усталости требуется использование шкал с большим числом уровней, что снижает точность предсказаний. Например, в работе [1] точность для трехуровневой шкалы составила 76%, а для четырехуровневой уже 61%.

Другой важной проблемой является ограничение применимости существующих методов в различных условиях и для разных видов деятельности. Например, в работе [2] был проведен анализ точности предсказания уровня усталости в зависимости от области крепления датчиков. Результаты показали, что наилучшие результаты были достигнуты при креплении датчиков в поясничной области, однако для других видов активности или рабочих условий могут потребоваться иные оптимальные места крепления. Также следует отметить, что большинство исследований были проведены на небольшой выборке испытуемых, что может ограничивать обобщение полученных результатов на более широкую аудиторию. Так в работе [3] участвовало всего пять строителей, что может снижать репрезентативность полученных результатов.

Для решения поставленной задачи детектирования степени мускульной усталости была выбрана модель случайного леса. Построение которого включает в себя концепцию использования ансамблей решающих деревьев, что способствует улучшению обобщающих способностей модели.

Одними из наиболее важных признаков для классификации оказались MFCC признаки для акселерометра. Эти признаки позволили учесть особенности сигнала и существенно повысить точность классификации. Данный факт открывает возможность дальнейшего исследования связи кинематических параметров походки с MFCC коэффициентами.

Для оценки качества модели и сравнения результатов использовалась метрика F1-score (29). Эта метрика предоставила более объективную оценку модели, позволяя уменьшить ложные детектирования. Также увеличить метрики удалось благодаря накоплению в течении одной минуты результатов работы программы и вывода значения, частота реализации которого наибольшая за этот промежуток. По полученным метрикам моделей было принято выбрать модель, у которой гиперпараметры были настроены вручную. Таким образом удалось исключить переобучение, что является важным аспектом для хорошей обобщающей способности модели на новых данных.

Результаты эксперимента показали, что выбранная модель случайного леса достигла высоких значений метрики F1-score для трех уровней мускульной усталости. Среднее значение метрики составило 87%, что свидетельствует о хорошей способности модели классифицировать уровни усталости. Важно отметить, что в рамках данной работы было подтверждено, что использование смартфона в кармане в качестве датчика является возможным и достаточным для детектирования мускульной усталости с достаточно высокой детализированной классификации. Это открывает перспективы для применения разработанной модели в реальных условиях.

В целом, несмотря на определенные достижения и прогресс в области детектирования мускульной усталости, остается необходимость в дальнейших исследованиях и разработке более точных и надежных методов. Более широкий охват испытуемых, учет различных условий и видов деятельности, а также применение современных технологий машинного обучения и глубокого обучения могут значительно улучшить результаты и применимость этих методов в реальных условиях.

Таким образом, цель работы была успешна достигнута, а поставленные задачи выполнены. Результаты исследования позволяют сделать вывод о применимости разработанных методов для детектирования степени мускульной усталости и открыть перспективы для дальнейших исследований в этой области.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Swapnali Babasaheb Karvekar, "Smartphone-based Human Fatigue Detection in an Industrial Environment Using Gait Analysis " (Декабрь 2019г). Thesis. Rochester Institute of Technology.
2. Nuno Cardoso, "MLFatigueDetection Machine Learning Based Walking Fatigue Detection" (Декабрь 2019г). Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa.
3. Sadra Fardhosseini, Mahmoud Habibnezhad, Houtan Jebelli, Giovanni Migliaccio, Hyun Woo Lee, Jay Puckett, "Recognition of Construction Workers’ Physical Fatigue Based on Gait Patterns Driven from Three-axis Accelerometer Embedded in a Smartphone" (Март 2020г). Construction Research Congress: TEMPE, ARIZONA.
4. Wen-Wen Yang, Kuan-Hsun Huang, En-Tzu Wang and Tzyy-Yuang Shiang, "DETECTING PHYSICAL FATIGUE USING ACCELEROMETER IN BASKETBALL PLAYERS" (Июль 2019г). Department of Athletic Performance, National Taiwan Normal University, Taipei, Taiwan.
5. Karla Gonzalez, Farzan Sasangohar, Ranjana K Mehta, Mark Lawley, Madhav Erraguntla, "Measuring Fatigue through Heart Rate Variability and Activity Recognition: A Scoping Literature Review of Machine Learning (Сентябрь 2017г). Environmental and Occupational Health, Texas A&M University.
6. Barandas M. et al. TSFEL: Time series feature extraction library //SoftwareX. – 2020. – Т. 11. – С. 100456.
7. Первичный анализ речевых сигналов // Альфацефей URL: https://alphacephei.com/ru/lecture1.pdf (дата обращения: 10.06.2023).
8. Paluszek M., Thomas S. MATLAB machine learning. – Apress, 2016.
9. Raschka S. Python machine learning. – Packt publishing ltd, 2015.
10. Открытый курс машинного обучения. Тема 3. Классификация, деревья решений и метод ближайших соседей // Хабр URL: https://habr.com/ru/companies/ods/articles/322534/ (дата обращения: 10.06.2023).
11. Открытый курс машинного обучения. Тема 5. Композиции: бэггин, случайный лес // Хабр URL: https://habr.com/ru/companies/ods/articles/324402/#1-begging (дата обращения: 10.06.2023).
12. Открытый курс машинного обучения. Тема 4. Линейные модели классификации и регрессии // Хабр URL: https://habr.com/ru/companies/ods/articles/323890/ (дата обращения: 10.06.2023).
13. Лекции по машинному обучению // razinkov URL: https://razinkov.ai/lectures (дата обращения: 10.06.2023).

## ПРИЛОЖЕНИЯ

## Приложение 1. Создание, настройка и обучение модели

**import** tsfel

**import** glob

**import** joblib

**import** pickle

**import** pandas as pd

**import** numpy as np

**import** seaborn as sns

**import** scipy.stats as stats

**import** matplotlib.pyplot as plt

**from** sklearn **import** preprocessing

**from** sklearn.model\_selection **import** train\_test\_split, StratifiedKFold

**from** sklearn.ensemble **import** RandomForestClassifier

**from** sklearn.metrics **import** classification\_report, f1\_score

**from** sklearn.feature\_selection **import** VarianceThreshold

**from** sklearn.metrics **import** confusion\_matrix, accuracy\_score

**from** sklearn.model\_selection **import** cross\_val\_score

**from** sklearn.model\_selection **import** GridSearchCV

**from** sklearn.impute **import** SimpleImputer

**import** warnings

**def** parsing(folder\_name, fs**=**50):

    """Принимает на вход название папки  из директории в которой находится датасет.

    Файлы должны быть записаны как Accel\_sub\_ и Gyr\_sub\_ соотвественно.

    feature\_class = temporal or statistical or spectral.

    Возвращает df с модулями акселерометра и гироскопа"""

    # Загрузка датасета

    files **=** glob.glob('C:\\Users\\Virtus\\Desktop\\Muscule\_fatigue\\DataSet\\'+folder\_name+'\\Accel\_sub**\***.csv')

    df1 **=** pd.DataFrame()

**for** file **in** files:

        data **=** pd.read\_csv(file)

        data **=** data.iloc[:**-**fs**\***5]

        data **=** data.iloc[fs**\***5:]

        df1 **=** df1.append(data)

    files **=** glob.glob('C:\\Users\\Virtus\\Desktop\\Muscule\_fatigue\\DataSet\\'+folder\_name+'\\Gyr\_sub**\***.csv')

    df2 **=** pd.DataFrame()

**for** file **in** files:

        data **=** pd.read\_csv(file)

        data **=** data.iloc[:**-**fs**\***5]

        data **=** data.iloc[fs**\***5:]

        df2 **=** df2.append(data)

    # Находим модуль

    df1 **=** pd.DataFrame((df1['X (m/s^2)'] **\*\*** 2 **+** df1['Y (m/s^2)'] **\*\*** 2 **+** df1['Z (m/s^2)'] **\*\*** 2) **\*\*** 0.5)

    df2 **=** pd.DataFrame((df2['X (rad/s)'] **\*\*** 2 **+** df2['Y (rad/s)'] **\*\*** 2 **+** df2['Z (rad/s)'] **\*\*** 2) **\*\*** 0.5)

    df1.columns **=** ['acc']

    df2.columns **=** ['gyr']

    # Удаление выбросов по квантилям

    Q1 **=** df1.quantile(q**=**.25)

    Q3 **=** df1.quantile(q**=**.75)

    IQR **=** df1.apply(stats.iqr)

    df1 **=** df1[~((df1 < (Q1 **-** 1.5 **\*** IQR)) | (df1 > (Q3 **+** 1.5 **\*** IQR))).any(axis**=**1)]

    Q1 **=** df2.quantile(q**=**.25)

    Q3 **=** df2.quantile(q**=**.75)

    IQR **=** df2.apply(stats.iqr)

    df2 **=** df2[~((df2 < (Q1 **-** 1.5 **\*** IQR)) | (df2 > (Q3 **+** 1.5 **\*** IQR))).any(axis**=**1)]

    #Объединяем в один датафрейм

    df **=** pd.merge(df1, df2, left\_index**=**True, right\_index**=**True)

    #sns.boxplot(data=df,orient='h')

**return** df

**def** features(df, feature\_class, fs**=**50, window\_size**=**250):

    '''Принимает на вход df, feature\_class - параметр, определяющий какой тип

    признаков будет извлечен, feature\_class = temporal or statistical or spectral.

    Возвращает df с признаками.'''

    # Выбор признаков которые будут извлечены

    cfg **=** tsfel.get\_features\_by\_domain(feature\_class)

    # Извлечение признаков

    X\_train **=** tsfel.time\_series\_features\_extractor(cfg, df, fs**=**fs, window\_size**=**window\_size,

                                                    header\_names**=**['acc', 'gyr'])

    # Удаление признаков с высокой корреляцией

    corr\_features **=** tsfel.correlated\_features(X\_train, threshold**=**0.6)

    X\_train.drop(corr\_features, axis**=**1, inplace**=**True)

    # Удаление признаков с низкой дисперсией, threshold - уровень 'схожести' столбцов

    var\_thr **=** VarianceThreshold(threshold**=**0.3)

    var\_thr.fit\_transform(X\_train)

    concol **=** [column **for** column **in** X\_train.columns

**if** column **not** **in** X\_train.columns[var\_thr.get\_support()]]

    X\_train **=** X\_train.drop(concol, axis**=**1)

    # Нормализация признаков

    nX\_train **=** pd.DataFrame(preprocessing.normalize(X\_train))

    nX\_train.columns **=** X\_train.columns

    # Запись таблицы с признаками

    nX\_train **=** pd.DataFrame(nX\_train)

**return** nX\_train

# Создаем df с датасета

df1 **=** parsing('Not\_Tired')

df2 **=** parsing('Average\_Fatigue')

df3 **=** parsing('High\_Fatigue')

# Вычисляем статистические признаки

NotFatig\_stat  **=** features(df**=**df1, feature\_class**=**'statistical')

AverFatig\_stat **=** features(df**=**df2, feature\_class**=**'statistical')

HighFatig\_stat **=** features(df**=**df3, feature\_class**=**'statistical')

# Вычисляем временные признаки

NotFatig\_temp  **=** features(df**=**df1, feature\_class**=**'temporal')

AverFatig\_temp **=** features(df**=**df2, feature\_class**=**'temporal')

HighFatig\_temp **=** features(df**=**df3, feature\_class**=**'temporal')

# Вычисляем спектральные признаки

NotFatig\_spec  **=** features(df**=**df1, feature\_class**=**'spectral')

AverFatig\_spec **=** features(df**=**df2, feature\_class**=**'spectral')

HighFatig\_spec **=** features(df**=**df3, feature\_class**=**'spectral')

# Объединяем df с признакми

NotFatig\_Data  **=** pd.concat([NotFatig\_stat, NotFatig\_temp, NotFatig\_spec], axis**=**1, join**=**'inner')

AverFatig\_Data **=** pd.concat([AverFatig\_stat, AverFatig\_temp, AverFatig\_spec], axis**=**1, join**=**'inner')

HighFatig\_Data **=** pd.concat([HighFatig\_stat, HighFatig\_temp, HighFatig\_spec], axis**=**1, join**=**'inner')

NotFatig\_Data2 **=** NotFatig\_Data[NotFatig\_Data !**=** 0]

#В каких колонках больше 30% нулей

missing\_features **=** NotFatig\_Data2.columns[NotFatig\_Data2.isnull().mean() > 0.30]

NotFatig\_Data **=** NotFatig\_Data2.drop(missing\_features, axis**=**1)#, inplace=True)

missing\_features

AverFatig\_Data2 **=** AverFatig\_Data[AverFatig\_Data !**=** 0]

#В каких колонках больше 30% нулей

missing\_features **=** AverFatig\_Data2.columns[AverFatig\_Data2.isnull().mean() > 0.30]

AverFatig\_Data **=** AverFatig\_Data2.drop(missing\_features, axis**=**1)#, inplace=True)

missing\_features

HighFatig\_Data2 **=** HighFatig\_Data[HighFatig\_Data !**=** 0]

#В каких колонках больше 30% нулей

missing\_features **=** HighFatig\_Data2.columns[HighFatig\_Data2.isnull().mean() > 0.30]

HighFatig\_Data **=** HighFatig\_Data2.drop(missing\_features, axis**=**1)#, inplace=True)

missing\_features

NotFatig\_Data **=** NotFatig\_Data[NotFatig\_Data !**=** 0]

NotFatig\_Data **=** NotFatig\_Data.apply(**lambda** row: row.fillna(row.mean()), axis**=**1)

AverFatig\_Data **=** AverFatig\_Data[AverFatig\_Data !**=** 0]

AverFatig\_Data **=** AverFatig\_Data.apply(**lambda** row: row.fillna(row.mean()), axis**=**1)

HighFatig\_Data **=** HighFatig\_Data[HighFatig\_Data !**=** 0]

HighFatig\_Data **=** HighFatig\_Data.apply(**lambda** row: row.fillna(row.mean()), axis**=**1)

# Прописываем метки

NotFatig\_Data['class']  **=** 0

AverFatig\_Data['class'] **=** 1

HighFatig\_Data['class'] **=** 2

# Объединяем в один датафрейм

FeatureData **=** pd.concat([NotFatig\_Data, AverFatig\_Data, HighFatig\_Data], axis**=**0) #AverFatig\_Data

FeatureData **=** FeatureData.dropna(axis**=**1)

FeatureData.to\_csv('C:\\Users\\Virtus\\Desktop\\Muscule\_fatigue\\isMuscle\_notebook\\Feature.csv', index**=**False)

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(FeatureData, FeatureData['class'], train\_size**=**0.7, random\_state**=**42)

a **=** pd.DataFrame(X\_train.columns)

a **=** a.transpose()

a.to\_excel('C:\\Users\\Virtus\\Desktop\\Muscule\_fatigue\\Shapka.xlsx', index **=** False)

X\_train **=** X\_train.drop(columns**=**['class'], axis**=**1)

X\_test **=** X\_test.drop(columns**=**['class'], axis**=**1)

activity\_labels **=** np.array(pd.read\_csv('C:\\Users\\Virtus\\Desktop\\Muscule\_fatigue\\isMuscle\_notebook\\activity\_labels.txt', header**=**None, delimiter**=**' '))[:,1]

activity\_labels

classifier **=** RandomForestClassifier(n\_jobs**=-**1, random\_state**=**42)

classifier.fit(X\_train, y\_train.ravel())

 y\_test\_predict **=** classifier.predict(X\_test)

accuracy **=** accuracy\_score(y\_test, y\_test\_predict) **\*** 100

print(classification\_report(y\_test, y\_test\_predict, target\_names**=**activity\_labels))

print("Accuracy: " **+** str(accuracy) **+** '%')

cm **=** confusion\_matrix(y\_test, y\_test\_predict)

df\_cm **=** pd.DataFrame(cm, index**=**[i **for** i **in** activity\_labels], columns**=**[i **for** i **in** activity\_labels])

plt.figure()

ax**=** sns.heatmap(df\_cm,  cbar**=**False, cmap**=**"BuGn", annot**=**True, fmt**=**"d")

plt.setp(ax.get\_xticklabels(), rotation**=**45)

plt.ylabel('Истинная метка', fontweight**=**'bold', fontsize **=** 18)

plt.xlabel('Предсказанная метка', fontweight**=**'bold', fontsize **=** 18)

bottom, top **=** ax.get\_ylim()

plt.show()

with open('RandomForest.pkl', 'wb') as file:

    pickle.dump(classifier, file)

with open('RandomForest.pkl', 'rb') as file:

    classifier\_new **=** pickle.load(file)

feature\_names **=** X\_train.columns

feature\_imp **=** pd.Series(classifier.feature\_importances\_,index**=**feature\_names).sort\_values(ascending**=**False)

sns.barplot(x**=**feature\_imp[0:18], y**=**feature\_imp.index[0:18])

plt.xlabel('Важность признаков')

plt.ylabel('Признаки')

plt.title('Визуализация важных признаков')

plt.show()

**from** sklearn.model\_selection **import** RandomizedSearchCV

n\_estimators **=** [int(x) **for** x **in** np.linspace(start **=** 100, stop **=** 1000, num **=** 10)]

max\_features **=** ['log2', 'sqrt']

max\_depth **=** [int(x) **for** x **in** np.linspace(start **=** 1, stop **=** 15, num **=** 15)]

min\_samples\_split **=** [int(x) **for** x **in** np.linspace(start **=** 2, stop **=** 50, num **=** 10)]

min\_samples\_leaf **=** [int(x) **for** x **in** np.linspace(start **=** 2, stop **=** 50, num **=** 10)]

bootstrap **=** [True, False]

param\_dist **=** {'n\_estimators': n\_estimators,

               'max\_features': max\_features,

               'max\_depth': max\_depth,

               'min\_samples\_split': min\_samples\_split,

               'min\_samples\_leaf': min\_samples\_leaf,

               'bootstrap': bootstrap}

rs **=** RandomizedSearchCV(classifier,

                        param\_dist,

                        n\_iter **=** 100,

                        cv **=** 3,

                        verbose **=** 1,

                        n\_jobs**=-**1,

                        random\_state**=**42)

rs.fit(X\_train, y\_train)

rs.best\_params\_

rs\_df **=** pd.DataFrame(rs.cv\_results\_).sort\_values('rank\_test\_score').reset\_index(drop**=**True)

rs\_df **=** rs\_df.drop([

            'mean\_fit\_time',

            'std\_fit\_time',

            'mean\_score\_time',

            'std\_score\_time',

            'params',

            'split0\_test\_score',

            'split1\_test\_score',

            'split2\_test\_score',

            'std\_test\_score'],

            axis**=**1)

rs\_df.head(10)

fig, axs **=** plt.subplots(ncols**=**3, nrows**=**2)

sns.set(style**=**"whitegrid", color\_codes**=**True, font\_scale **=** 2)

fig.set\_size\_inches(30,25)

sns.barplot(x**=**'param\_n\_estimators', y**=**'mean\_test\_score', data**=**rs\_df, ax**=**axs[0,0], color**=**'lightgrey')

axs[0,0].set\_ylim([0.7,.9])

axs[0,0].set\_title(label **=** 'n\_estimators', size**=**30, weight**=**'bold')

sns.barplot(x**=**'param\_min\_samples\_split', y**=**'mean\_test\_score', data**=**rs\_df, ax**=**axs[0,1], color**=**'coral')

axs[0,1].set\_ylim([0.7,.9])

axs[0,1].set\_title(label **=** 'min\_samples\_split', size**=**30, weight**=**'bold')

sns.barplot(x**=**'param\_min\_samples\_leaf', y**=**'mean\_test\_score', data**=**rs\_df, ax**=**axs[0,2], color**=**'lightgreen')

axs[0,2].set\_ylim([0.7,.9])

axs[0,2].set\_title(label **=** 'min\_samples\_leaf', size**=**30, weight**=**'bold')

sns.barplot(x**=**'param\_max\_features', y**=**'mean\_test\_score', data**=**rs\_df, ax**=**axs[1,0], color**=**'wheat')

axs[1,0].set\_ylim([0.7,.9])

axs[1,0].set\_title(label **=** 'max\_features', size**=**30, weight**=**'bold')

sns.barplot(x**=**'param\_max\_depth', y**=**'mean\_test\_score', data**=**rs\_df, ax**=**axs[1,1], color**=**'lightpink')

axs[1,1].set\_ylim([0.7,.9])

axs[1,1].set\_title(label **=** 'max\_depth', size**=**30, weight**=**'bold')

sns.barplot(x**=**'param\_bootstrap',y**=**'mean\_test\_score', data**=**rs\_df, ax**=**axs[1,2], color**=**'skyblue')

axs[1,2].set\_ylim([0.7,.9])

axs[1,2].set\_title(label **=** 'bootstrap', size**=**30, weight**=**'bold')

plt.show()

**from** sklearn.model\_selection **import** GridSearchCV

n\_estimators **=** [900]

max\_features **=** ['sqrt', 'log2']

max\_depth **=** [13, 14, 15]

min\_samples\_split **=** [20, 21, 22, 23, 44]

min\_samples\_leaf **=** [16, 17, 18]

bootstrap **=** [False, True]

param\_grid **=** {'n\_estimators': n\_estimators,

               'max\_features': max\_features,

               'max\_depth': max\_depth,

               'min\_samples\_split': min\_samples\_split,

               'min\_samples\_leaf': min\_samples\_leaf,

               'bootstrap': bootstrap}

gs **=** GridSearchCV(classifier, param\_grid, cv **=** 3, verbose **=** 1, n\_jobs**=-**1)

gs.fit(X\_train, y\_train)

rfc\_3 **=** gs.best\_estimator\_

gs.best\_params\_

y\_test\_predict **=**  gs.best\_estimator\_.predict(X\_test)

# Get the classification report

accuracy **=** accuracy\_score(y\_test, y\_test\_predict) **\*** 100

print(classification\_report(y\_test, y\_test\_predict, target\_names**=**activity\_labels))

print("Accuracy: " **+** str(accuracy) **+** '%')

cm **=** confusion\_matrix(y\_test, y\_test\_predict)

df\_cm **=** pd.DataFrame(cm, index**=**[i **for** i **in** activity\_labels], columns**=**[i **for** i **in** activity\_labels])

plt.figure()

ax**=** sns.heatmap(df\_cm,  cbar**=**False, cmap**=**"BuGn", annot**=**True, fmt**=**"d")

plt.setp(ax.get\_xticklabels(), rotation**=**45)

plt.ylabel('Истинная метка', fontweight**=**'bold', fontsize **=** 18)

plt.xlabel('Предсказанная метка', fontweight**=**'bold', fontsize **=** 18)

#bottom, top = ax.get\_ylim()

plt.show()

mean\_scores\_train **=** []

mean\_scores\_val **=** []

**for** n **in** range(1,251,10):

    classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, n\_estimators **=** n)

    scores\_train **=** cross\_val\_score(classifier, X\_train, y\_train, cv**=**5, scoring**=**'f1\_macro')

    scores\_val **=** cross\_val\_score(classifier, X\_test, y\_test, cv**=**5, scoring**=**'f1\_macro')

    scores\_train **=** 1 **-** scores\_train

    scores\_val **=** 1 **-** scores\_val

    mean\_scores\_train.append(scores\_train.mean())

    mean\_scores\_val.append(scores\_val.mean())

n **=** range(1,251,10)

fig1 **=** plt.figure()

ax1 **=** fig1.add\_subplot(111)

ax1.plot(n, mean\_scores\_val, label **=** 'Cv')

ax1.plot(n, mean\_scores\_train, label **=** 'Train')

plt.legend()

plt.xlabel("Количество деревьев")

plt.ylabel("Ошибка")

mean\_scores\_train **=** []

mean\_scores\_val **=** []

**for** n **in** range(1,102,10):

    classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, max\_depth **=** n)

    scores\_train **=** cross\_val\_score(classifier, X\_train, y\_train, cv**=**5, scoring**=**'f1\_macro')

    scores\_val **=** cross\_val\_score(classifier, X\_test, y\_test, cv**=**5, scoring**=**'f1\_macro')

    scores\_train **=** 1 **-** scores\_train

    scores\_val **=** 1 **-** scores\_val

    mean\_scores\_train.append(scores\_train.mean())

    mean\_scores\_val.append(scores\_val.mean())

n **=** range(1,102,10)

fig1 **=** plt.figure()

ax1 **=** fig1.add\_subplot(111)

ax1.plot(n, mean\_scores\_val, label **=** 'Cv')

ax1.plot(n, mean\_scores\_train, label **=** 'Train')

plt.legend()

plt.xlabel("Глубина дерева")

plt.ylabel("Ошибка")

mean\_scores\_train **=** []

mean\_scores\_val **=** []

**for** n **in** range(2,1000,10):

    classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, max\_leaf\_nodes **=** n)

    scores\_train **=** cross\_val\_score(classifier, X\_train, y\_train, cv**=**5, scoring**=**'f1\_macro')

    scores\_val **=** cross\_val\_score(classifier, X\_test, y\_test, cv**=**5, scoring**=**'f1\_macro')

    scores\_train **=** 1 **-** scores\_train

    scores\_val **=** 1 **-** scores\_val

    mean\_scores\_train.append(scores\_train.mean())

    mean\_scores\_val.append(scores\_val.mean())

n **=** range(2,1000,10)

fig1 **=** plt.figure()

ax1 **=** fig1.add\_subplot(111)

ax1.plot(n, mean\_scores\_val, label **=** 'Cv')

ax1.plot(n, mean\_scores\_train, label **=** 'Train')

plt.legend()

plt.xlabel("Количество листов")

plt.ylabel("Ошибка")

mean\_scores\_train **=** []

mean\_scores\_val **=** []

**for** n **in** range(1,41):

    classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, max\_features **=** n)

    scores\_train **=** cross\_val\_score(classifier, X\_train, y\_train, cv**=**5, scoring**=**'f1\_macro')

    scores\_val **=** cross\_val\_score(classifier, X\_test, y\_test, cv**=**5, scoring**=**'f1\_macro')

    scores\_train **=** 1 **-** scores\_train

    scores\_val **=** 1 **-** scores\_val

    mean\_scores\_train.append(scores\_train.mean())

    mean\_scores\_val.append(scores\_val.mean())

n **=** range(1,41)

fig1 **=** plt.figure()

ax1 **=** fig1.add\_subplot(111)

ax1.plot(n, mean\_scores\_val, label **=** 'Cv')

ax1.plot(n, mean\_scores\_train, label **=** 'Train')

plt.legend()

plt.xlabel("Максимальное количество признаков для разбиения")

plt.ylabel("Ошибка")

mean\_scores\_train **=** []

mean\_scores\_val **=** []

**for** n **in** range(2,41):

    classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, min\_samples\_split **=** n)

    scores\_train **=** cross\_val\_score(classifier, X\_train, y\_train, cv**=**5, scoring**=**'f1\_macro')

    scores\_val **=** cross\_val\_score(classifier, X\_test, y\_test, cv**=**5, scoring**=**'f1\_macro')

    scores\_train **=** 1 **-** scores\_train

    scores\_val **=** 1 **-** scores\_val

    mean\_scores\_train.append(scores\_train.mean())

    mean\_scores\_val.append(scores\_val.mean())

n **=** range(2,41)

fig1 **=** plt.figure()

ax1 **=** fig1.add\_subplot(111)

ax1.plot(n, mean\_scores\_val, label **=** 'Cv')

ax1.plot(n, mean\_scores\_train, label **=** 'Train')

plt.legend()

plt.xlabel("min\_samples\_split")

plt.ylabel("Ошибка")

mean\_scores\_train **=** []

mean\_scores\_val **=** []

**for** n **in** range(2,41):

    classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, min\_samples\_leaf **=** n)

    scores\_train **=** cross\_val\_score(classifier, X\_train, y\_train, cv**=**5, scoring**=**'f1\_macro')

    scores\_val **=** cross\_val\_score(classifier, X\_test, y\_test, cv**=**5, scoring**=**'f1\_macro')

    scores\_train **=** 1 **-** scores\_train

    scores\_val **=** 1 **-** scores\_val

    mean\_scores\_train.append(scores\_train.mean())

    mean\_scores\_val.append(scores\_val.mean())

n **=** range(2,41)

fig1 **=** plt.figure()

ax1 **=** fig1.add\_subplot(111)

ax1.plot(n, mean\_scores\_val, label **=** 'Cv')

ax1.plot(n, mean\_scores\_train, label **=** 'Train')

plt.legend()

plt.xlabel("min\_samples\_leaf")

plt.ylabel("Ошибка")

classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, min\_impurity\_decrease **=** 1e**-**12)

classifier.fit(X\_train, y\_train)

print(f1\_score(y\_train, classifier.predict(X\_train), average**=**'micro'),

      f1\_score(y\_test, classifier.predict(X\_test), average**=**'micro'))

classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, min\_impurity\_decrease **=** 1e**-**7)

classifier.fit(X\_train, y\_train)

print(f1\_score(y\_train, classifier.predict(X\_train), average**=**'micro'),

      f1\_score(y\_test, classifier.predict(X\_test), average**=**'micro'))

classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, min\_impurity\_decrease **=** 1e**-**5)

classifier.fit(X\_train, y\_train)

print(f1\_score(y\_train, classifier.predict(X\_train), average**=**'micro'),

      f1\_score(y\_test, classifier.predict(X\_test), average**=**'micro'))

classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, min\_impurity\_decrease **=** 1e**-**4)

classifier.fit(X\_train, y\_train)

print(f1\_score(y\_train, classifier.predict(X\_train), average**=**'micro'),

      f1\_score(y\_test, classifier.predict(X\_test), average**=**'micro'))

classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, min\_impurity\_decrease **=** 1e**-**3)

classifier.fit(X\_train, y\_train)

print(f1\_score(y\_train, classifier.predict(X\_train), average**=**'micro'),

      f1\_score(y\_test, classifier.predict(X\_test), average**=**'micro'))

classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, min\_impurity\_decrease **=** 1e**-**2)

classifier.fit(X\_train, y\_train)

print(f1\_score(y\_train, classifier.predict(X\_train), average**=**'micro'),

      f1\_score(y\_test, classifier.predict(X\_test), average**=**'micro'))

classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, min\_impurity\_decrease **=** 1e**-**1)

classifier.fit(X\_train, y\_train)

print(f1\_score(y\_train, classifier.predict(X\_train), average**=**'micro'),

      f1\_score(y\_test, classifier.predict(X\_test), average**=**'micro'))

classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, class\_weight**=**'balanced')

classifier.fit(X\_train, y\_train)

print(f1\_score(y\_train, classifier.predict(X\_train), average**=**'micro'),

      f1\_score(y\_test, classifier.predict(X\_test), average**=**'micro'))

classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, class\_weight**=**{0: 1, 1: 2.12, 2:2.39})

classifier.fit(X\_train, y\_train)

print(f1\_score(y\_train, classifier.predict(X\_train), average**=**'micro'),

      f1\_score(y\_test, classifier.predict(X\_test), average**=**'micro'))

classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, class\_weight**=**{0: 1, 1: 3, 2:3})

classifier.fit(X\_train, y\_train)

print(f1\_score(y\_train, classifier.predict(X\_train), average**=**'micro'),

      f1\_score(y\_test, classifier.predict(X\_test), average**=**'micro'))

classifier **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1, class\_weight**=**{0: 1, 1: 4, 2:4})

classifier.fit(X\_train, y\_train)

print(f1\_score(y\_train, classifier.predict(X\_train), average**=**'micro'),

      f1\_score(y\_test, classifier.predict(X\_test), average**=**'micro'))

classifier **=** RandomForestClassifier(n\_jobs**=-**1,

                                    random\_state **=** 42,

                                    n\_estimators  **=** 150,

                                    max\_depth **=** 18,

                                    max\_leaf\_nodes **=** 250,

                                    max\_features **=** 4,

                                    min\_samples\_split **=** 2,

                                    min\_samples\_leaf **=** 1,

                                    min\_impurity\_decrease **=** 1e**-**3,

                                    class\_weight**=**{0: 1, 1: 2.12, 2:2.39})

classifier.fit(X\_train, y\_train.ravel())

y\_test\_predict **=** classifier.predict(X\_test)

f1score **=** f1\_score(y\_test, y\_test\_predict, average**=**'micro') **\*** 100

print(classification\_report(y\_test, y\_test\_predict, target\_names**=**activity\_labels))

print("f1\_score: " **+** str(f1score) **+** '%')

cm **=** confusion\_matrix(y\_test, y\_test\_predict)

df\_cm **=** pd.DataFrame(cm, index**=**[i **for** i **in** activity\_labels], columns**=**[i **for** i **in** activity\_labels])

plt.figure()

ax**=** sns.heatmap(df\_cm,  cbar**=**False, cmap**=**"BuGn", annot**=**True, fmt**=**"d")

plt.setp(ax.get\_xticklabels(), rotation**=**45)

plt.ylabel('Инстинная метка', fontweight**=**'bold', fontsize **=** 18)

plt.xlabel('Предсказанная метка', fontweight**=**'bold', fontsize **=** 18)

bottom, top **=** ax.get\_ylim()

plt.show()

X\_train **=** FeatureData.drop('class', axis**=**1)

y\_train **=** FeatureData['class']

classifier **=** RandomForestClassifier(n\_jobs**=-**1,

                                    random\_state **=** 42,

                                    n\_estimators  **=** 500,

                                    max\_depth **=** 15,

                                    max\_leaf\_nodes **=** 200,

                                    max\_features **=** 10,

                                    min\_samples\_split**=**8,

                                    min\_samples\_leaf**=**2,

                                    min\_impurity\_decrease **=** 1e**-**3,

                                    class\_weight**=**{0: 1, 1: 2.12, 2:2.39})

classifier.fit(X\_train, y\_train)

# Сохраняем

with open('RandomForest.pkl', 'wb') as file:

    pickle.dump(classifier, file)

## Приложение 2. Внедрение модели

1. **import** tsfel
2. **import** pickle
3. **import** pandas as pd
4. **import** numpy as np
6. **import** scipy.stats as stats
7. **from** sklearn **import** preprocessing
9. **import** warnings
10. warnings.filterwarnings("ignore")
12. fs **=** 50
13. window\_size **=** 250
15. df1  **=** pd.DataFrame()
17. data **=** file.iloc[:**-**fs**\***3]
18. data **=** data.iloc[fs**\***3:]
19. df1  **=** df1.append(data)
20. df1  **=** df1.drop(df1.columns[[0]], axis**=**1)
21. df2  **=** pd.DataFrame()
23. data2 **=** file.iloc[:**-**fs**\***3]
24. data2 **=** data2.iloc[fs**\***3:]
25. df2  **=** df2.append(data2)
26. df2  **=** df2.drop(df2.columns[[0]], axis**=**1)
28. df1 **=** pd.DataFrame((df1['X (m/s^2)'] **\*\*** 2 **+** df1['Y (m/s^2)'] **\*\*** 2 **+** df1['Z (m/s^2)'] **\*\*** 2) **\*\*** 0.5)
29. df2 **=** pd.DataFrame((df2['X (rad/s)'] **\*\*** 2 **+** df2['Y (rad/s)'] **\*\*** 2 **+** df2['Z (rad/s)'] **\*\*** 2) **\*\*** 0.5)
31. df1.columns **=** ['acc']
32. df2.columns **=** ['gyr']
34. # Удаление выбросов по квантилям
35. Q1 **=** df1.quantile(q**=**.25)
36. Q3 **=** df1.quantile(q**=**.75)
37. IQR **=** df1.apply(stats.iqr)
38. df1 **=** df1[~((df1 < (Q1 **-** 1.5 **\*** IQR)) | (df1 > (Q3 **+** 1.5 **\*** IQR))).any(axis**=**1)]
40. Q1 **=** df2.quantile(q**=**.25)
41. Q3 **=** df2.quantile(q**=**.75)
42. IQR **=** df2.apply(stats.iqr)
43. df2 **=** df2[~((df2 < (Q1 **-** 1.5 **\*** IQR)) | (df2 > (Q3 **+** 1.5 **\*** IQR))).any(axis**=**1)]
45. #Объединяем в один датафрейм
46. df **=** pd.merge(df1, df2, left\_index**=**True, right\_index**=**True)
48. # Выбор признаков которые будут извлечены
49. cfg1 **=** tsfel.get\_features\_by\_domain('statistical')
50. cfg2 **=** tsfel.get\_features\_by\_domain('temporal')
51. cfg3 **=** tsfel.get\_features\_by\_domain('spectral')
53. # Извлечение признаков
54. X\_train\_stat **=** tsfel.time\_series\_features\_extractor(cfg1, df, fs**=**fs, window\_size**=**window\_size, header\_names**=**['acc','gyr'])
56. X\_train\_temp **=** tsfel.time\_series\_features\_extractor(cfg2, df, fs**=**fs, window\_size**=**window\_size,  header\_names**=**['acc','gyr'])
58. X\_train\_spec **=** tsfel.time\_series\_features\_extractor(cfg3, df, fs**=**fs, window\_size**=**window\_size, header\_names**=**['acc','gyr'])
60. X\_train **=** pd.concat([X\_train\_stat, X\_train\_temp, X\_train\_spec], axis**=**1, join**=**'inner')
62. nX\_train **=** pd.DataFrame(preprocessing.normalize(X\_train))
63. nX\_train.columns **=** X\_train.columns
65. colum **=** pd.read\_excel('C:\\Users\Virtus\\Desktop\Muscule\_fatigue\\Shapka.xlsx')
66. data **=** nX\_train[nX\_train.columns.intersection(colum.columns)]
67. data **=** data.tail(1)
69. # Загружаем модель
70. with open('RandomForest.pkl', 'rb') as file:
71. classifier **=** pickle.load(file)
73. result **=** classifier.predict(data)
74. RESULT **=** pd.DataFrame(result)