**Описание проекта**

Проект реализует систему машинного обучения для **обнаружения** и **классификации** дефектов в электрических системах на основе анализа сигнатур тока.  
Используется обработка временных рядов, выделение признаков и обучение моделей для многоклассовой классификации.

**Основные возможности:**

* Полный **пайплайн обработки данных**:

1. загрузка и предобработка данных,
2. извлечение признаков (и таргета, при обучении) из сигналов тока с помощью математической модели,
3. трансформация таргета (кодирование).

* **Двухэтапная классификация**:

1. Определение типа дефекта.
2. Определение степени его тяжести.
3. Поддержка классов дефектов:

* **Inner Race, Outer Race, Ball, Cage, Rotor, Misalignment, Normal.**

1. Поддержка степени тяжести:

* **None**, **Low**, **Medium**, **High**.
* Обучение и инференс с использованием **CatBoost**.

**Технический стек**

* **Язык:** Python 3.10+
* **Основные библиотеки:**
  + 1. CatBoost — обучение моделей классификации.
    2. scikit-learn — метрики и разбиение данных.
    3. pandas — обработка таблиц.
    4. numpy — численные вычисления.
    5. Fastapi – создание api для backend
    6. Uvicorn – для запуска frontend

**Структура проекта**

project/ # Корень проекта

├── data/ # Данные

│ ├── data\_raw/ # Измерения тока

│ ├── data\_processed/ # Предобработанные данные

│ ├── data\_feature/ # Выделенные признаки и таргет

│ ├── data\_summary/ # Результат работы

│ └── data\_ml/ # Данные для инференса и генерации отчетов

│

├── model/ # Тренированные модели

│ ├── model\_defect.cbm # Модель классификации дефектов

│ └── model\_severity.cbm # Модель классификации степени дефектов

│

├── public/ # Веб-интерфейс

│ ├── index.html # Основной интерфейс

│ ├── script.js # Логика фронтенда

│ └── style.css # Стилизация

│

├── src/ # Исходный код

│ ├── feature\_extraction/ # Модуль извлечения признаков

│ │ ├── \_\_init\_\_.py

│ │ ├── fault\_params.py # Расчет параметров дефектов

│ │ ├── feature\_classifier.py # Классификация признаков

│ │ └── feature\_extraction.py # Логика выделения признаков

│ │

│ ├── load\_preprocessing/ # Загрузка и предобработка данных

│ │ ├── \_\_init\_\_.py

│ │ ├── data\_loader.py # Утилиты для загрузки данных

│ │ ├── full\_prep.py # Создание структуры папок, предобработка и разметка данных

│ │ ├── pipeline\_load\_refactor.py # Основной пайплайн обработки

│ │ └── preprocessing.py # Предобработка сигналов

│ │

│ ├── model/ # Логика моделей

│ │ ├── \_\_init\_\_.py

│ │ ├── data\_transform.py # Подготовка данных к обучению модели

│ │ ├── model.py # Основная реализация моделей

│ │ ├── model\_sandbox.py # Экспериментальные модели

│ │ └── predict\_process\_defects.py # Предсказание дефектов и формирование отчета

│ │

│ ├── visualise\_reports/ # Визуализация и отчеты

│ │ ├── data\_visualiser.py # Визуализация данных и метрик

│ │ └── report\_generator.py # Генерация отчетов

│ │

│ ├── \_\_init\_\_.py

│ ├── app.py # Основное приложение

│ ├── file\_monitor.py # Мониторинг файлов

│ ├── main.py # Точка входа

│ └── monitor.py # Утилиты мониторинга

│

├── requirements.txt # Зависимости проекта

└── README.md # Основная документация

**Поток данных**

flowchart LR

A[Сырые данные тока] --> B[Предобработка и очистка]

B --> C[Извлечение признаков]

C --> D[Подготовка обучающей выборки]

D --> E[Обучение модели]

E --> F[Предсказание типа дефекта]

E --> G[Предсказание степени дефекта]

F --> H[Отчеты и визуализации]

G --> H[Отчеты и визуализации]

**Работа математической модели для извлечения признаков (и таргета, при обучении) из сигналов тока**

**Основные модули**

**1.1. feature\_extraction.py — стартовый модуль**

Этот файл управляет всем процессом.  
Главная функция — extract\_features\_from\_file(csv\_path):

1. **Читает CSV чанками** (pandas.read\_csv).
2. **Нормализует названия колонок** → всегда получаем Current\_R, Current\_S, Current\_T.
3. **Разбивает сигнал на окна**:
   * длина окна = WINDOW\_SEC \* fs (по умолчанию 1 секунда);
   * шаг окна = STEP\_SEC \* fs (0.5 секунды);
   * fs = частота дискретизации (по умолчанию **25.6 кГц**).
4. **Чистит NaN и выбросы** (\_clean\_window):
   * если меньше 80% корректных значений → окно пропускается.
5. **Восстанавливает недостающие фазы**:
   * 1 фаза → копируем её на все три;
   * 2 фазы → пропавшую заполняем средним двух других.
6. Передаёт каждое окно в get\_feature\_vector и classify\_defect\_scored из feature\_classifier.py.
7. Сохраняет результат — таблицу признаков + метки дефекта и степени развития дефекта.

**1.2. feature\_classifier.py — извлечение признаков и классификация**

Этот файл — сердце модели. Тут три ключевых блока:

**А. Предобработка сигнала**

Функция preprocess\_per\_phase(x):

* вычитает среднее (убираем постоянную составляющую);
* заменяет NaN и бесконечности на нули;
* применяет **высокочастотный фильтр Баттерворта**:

где:

* = CONFIG["HPF\_HZ"] = **1 Гц** (по умолчанию);
* = частота дискретизации.

**B. Спектр огибающей**

Функция envelope\_spectrum\_2\_3k(x, fs):

1. **Выбирает резонансную полосу** → ищет такой диапазон частот, где максимальна куртозис (мера "ударности" сигнала).
2. Пропускает сигнал через **полосовой фильтр** (например, 2–3 кГц).
3. Применяет **преобразование Хилберта** для получения **огибающей**:
4. Строит **амплитудный спектр огибающей** (БПФ):

где — окно Ханна.

Этот спектр нужен, чтобы вытащить частоты, связанные с **дефектами подшипника**.

**C. Расчёт частот дефектов подшипника**

Подшипник задаётся файлом bearing\_config.json.

Из этих данных считаются **характерные частоты дефектов**

где:

* ​ — частота вращения вала;
* **BPFI** — дефект внутреннего кольца;
* **BPFO** — дефект внешнего кольца;
* **BSF** — дефект шарика;
* **FTF** — дефект сепаратора.

**D. Извлечение 26 признаков**

Главная функция — get\_feature\_vector().  
Вычисляет 26 признаков (f1...f26):

* **f1...f4** — нормализованные пики в огибающей на частотах BPFI, BPFO, BSF, FTF.
* **f5** — боковые полосы вокруг сетевой частоты (60 Гц), отражают модуляцию.
* **f6...f25** — статистики спектра вокруг каждой частоты:
* стандартное отклонение,
* асимметрия,
* эксцесс,
* среднее,
* энтропия.
* **f26** — наклон спектра огибающей в логарифмическом масштабе (**shape-фактор**).

**E. Классификация дефекта**

Функция classify\_defect\_scored():

1. **Считает скоринги семейств подшипников**:

где:

* — энергия гармоник и боковых полос в спектре огибающей;
* — оценка боковых в основном спектре (MCSA);
* веса = 0.5, = 0.5

1. Если дефект **явно выражен**:

* смотрим, у кого самый высокий K;
* проверяем разрыв с остальными;
* проверяем качество огибающей (PSNR).

1. Если не уверен — проверяем **ротор**:

* симметричные пики вокруг сетевой частоты → сломанные стержни;
* энергия боковых → дисбаланс или несоосность.

1. Если признаков нет → класс **Normal**.

**1.3. fault\_params.py — конфигурация модели**

Задаёт все пороговые значения и константы:

* полоса фильтрации;
* веса скорингов;
* минимальные SNR;
* пороги для серьёзности дефекта;
* флаги адаптивной обработки.

**1.4. bearing\_config.json — геометрия подшипника**

Используется для точного вычисления частот дефектов (см. формулы выше).

**2. Полный процесс для «сырых» данных**

1. **Читаем CSV** → берём трёхфазные токи.
2. **Разбиваем на окна** по 1 секунде.
3. Для каждого окна:
   1. Чистим данные и фильтруем;
   2. Вычисляем спектры и огибающую;
   3. Считаем 26 признаков;
   4. Классифицируем дефект и серьёзность.
4. Возвращаем таблицу:

|  |  |  |
| --- | --- | --- |
| F1…F26 | defect | severity |

**Классификация типа дефекта**

• **Тип задачи**: многоклассовая классификация.

• **Классы**: Inner Race, Outer Race, Ball, Cage, Rotor, Misalignment, Normal.

• **Модель**: CatBoostClassifier

• **Гиперпараметры**:

* iterations=200,
* depth=6,
* learning\_rate=0.1,
* loss\_function="MultiClass",
* random\_seed=42

**Классификация степени дефекта**

• **Тип задачи**: многоклассовая классификация.

• **Классы**: None, Low, Medium, High.

• **Модель**: CatBoostClassifier

• **Гиперпараметры**: аналогичны предыдущей модели.

**Установка и запуск**

Описано в README.md, приложенному к коду.