Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«**Национальный исследовательский университет ИТМО**»

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

“Идентификация неисправностей асинхронного двигателя с применением машинного обучения”

Выполнили:
Демкин А. И. R3343
Мовчан И. Е. R3380
Кашапова А. К. R3243
Руководитель:
Демидова Г. Л.

Санкт-Петербург
2025

Оглавление

Введение.

Современные промышленные комплексы и системы автоматизированного управления в значительной степени основаны на применении электрических машин, среди которых одни из самых популярных - асинхронные двигатели. Они просты, надежны и экономичны.

Методы машинного обучения открывают новые возможности для создания автономных систем диагностики и прогнозирования остаточного ресурса электроприводов. Способность ML-алгоритмов выявлять сложные, неочевидные для человека закономерности и скрытые признаки неисправностей в многомерных потоках данных (токи статора и ротора, напряжения, скорость, вибрация, температура, акустические эмиссии) позволяет решать задачи, недоступные классическим подходам. Технологии ML, такие как методы классификации (SVM, деревья решений, случайные леса, градиентный бустинг), искусственные нейронные сети (включая глубокое обучение для анализа временных рядов или изображений тепловых карт), кластеризация и аномальное обнаружение, способны:

- Автоматизировать процесс диагностики.
- Повысить точность и надежность распознавания конкретного типа неисправности на ранней стадии.
- Обеспечить адаптивность к различным режимам работы и условиям эксплуатации.
- Сократить время простоя оборудования за счет оперативного предупреждения о дефекте.

Целью данной производственной практики является исследование возможностей и разработка подходов к идентификации неисправностей типового промышленного электропривода на основе методов машинного обучения.

В рамках практики предполагается: изучить характерные неисправности основных компонентов электропривода (двигатель, преобразователь, датчики) и их проявления в сигналах; собрать набор данных, отражающий работу привода в нормальном режиме и при различных неисправностях; разработать и протестировать прототип системы диагностики.

Глава 1. Сбор данных.

Мы использовали виртуальную модель трёхфазного асинхронного двигателя, созданную в среде MATLAB/Simulink.

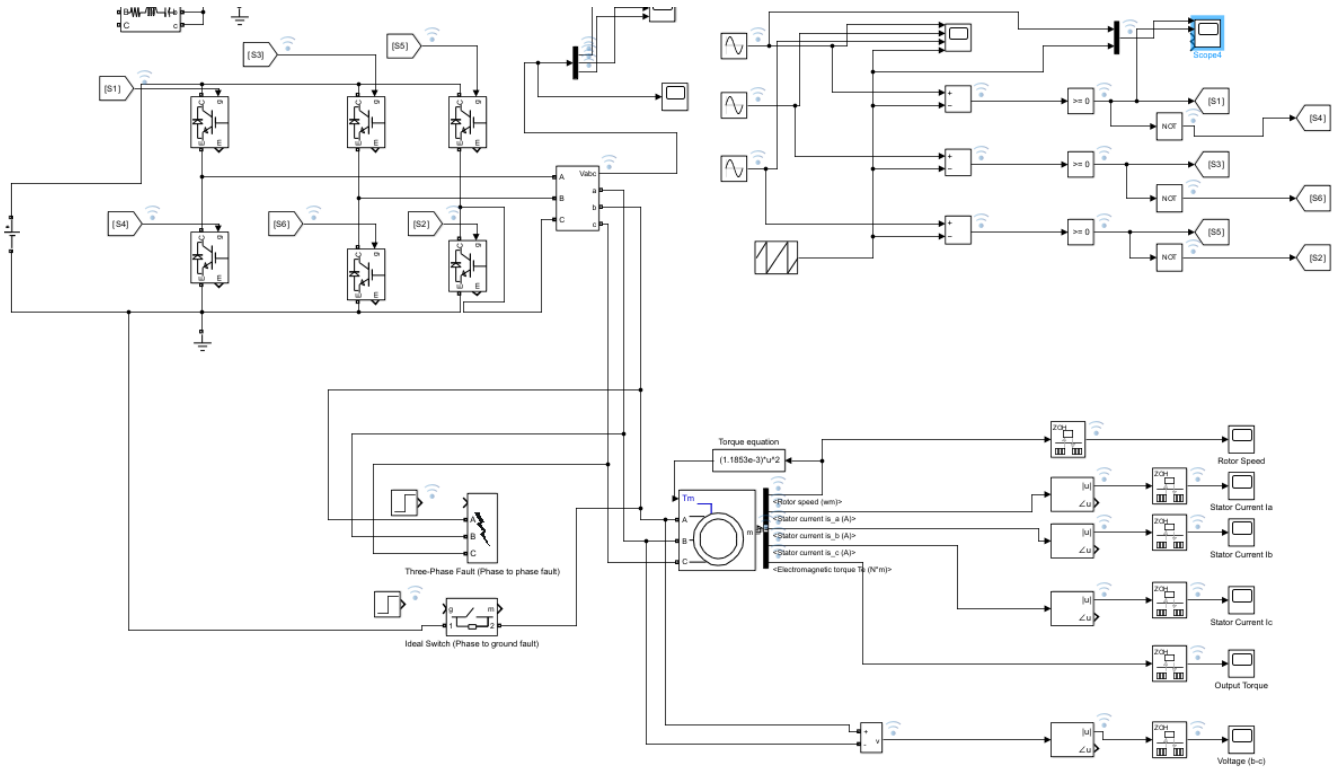

Рисунок 1. Модель трёхфазного асинхронного двигателя.

Для воспроизведения результатов работы укажем параметры двигателя:

- **Номинальная мощность:** 4000 Вт;
- **Частота напряжения статора:** 50 Гц;
- **Сопротивление статора:** 1.405 Ом;
- **Индуктивность статора:** 0.005839 Гн;
- **Сопротивление ротора:** 1.395 Ом;
- **Индуктивность ротора:** 0.005839 Гн;
- **Взаимная индуктивность:** 0.1722 Гн;
- **Инерция:** 0.0131 кг·м²;
- **Коэффициент трения:** 0.0131 Н·м·с;
- **Пара полюсов:** 2;
- **Тип ротора:** squirrel cage, за счет чего внешнее напряжение на роторе отсутствовало, и токи на нем возникали за счет индукции, через воздействие статора;
- **Входное воздействие:** механический крутящий момент $T_e = 1.1853e^{-3}\omega_R^2$;

На этой модели имитировались четыре режима работы:

- **normal** – нормальная работа без дефектов;
- **ag** – замыкание фазы A на землю;
- **bg** – замыкание фазы B на землю;
- **cg** – замыкание фазы C на землю.

Для каждого из этих режимов снимались следующие временные ряды:

- трёхфазовые токи статора – $I_a,\ I_b,\ I_c$;
- скорость ротора –$\omega_R(t)$;
- входной крутящий момент –$T_e(t)$;
- выходной (нагрузочный) электромагнитный крутящий момент – $T_m(t)$.
- частотный анализ указанных токов статора на фундаментальной частоте $f_n = 50$ (амплитуда);
- частотный анализ разности подаваемых напряжений на фазу A и B статора на фундаментальной частоте $f_n = 50$ (амплитуда).

Как уже обозначалось, токи ротора полностью зависят от токов на статоре, так что имело место снимать только их.

Собранные данные затем были сохранены в датасеты и использованы для обучения модели машинного обучения.

Графики собранных величин:

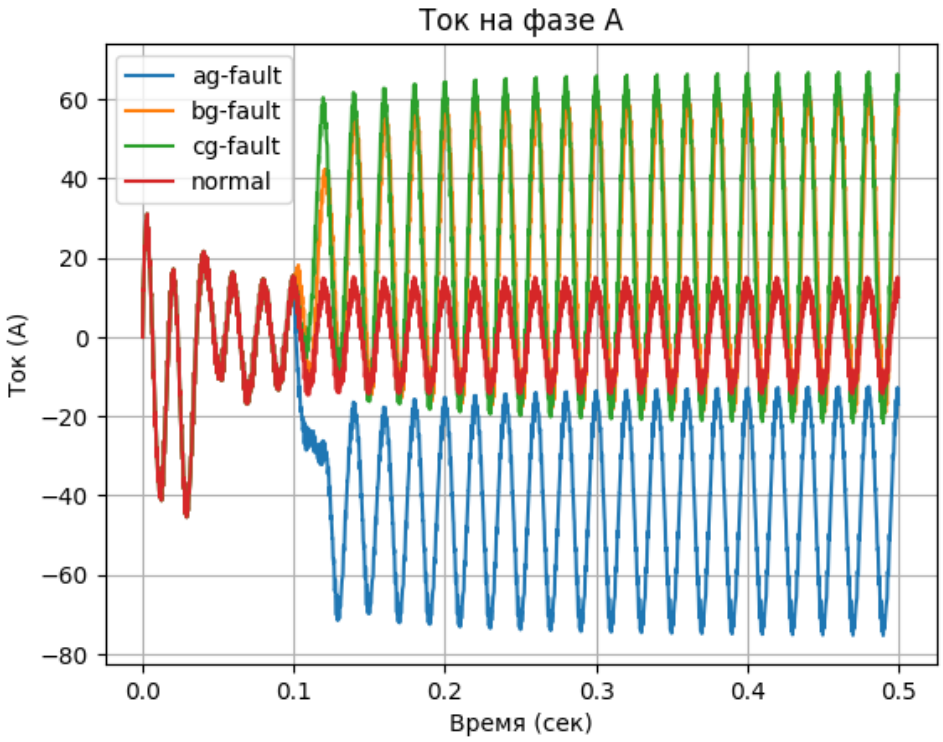

Рисунок 2. График тока первой фазы (A).

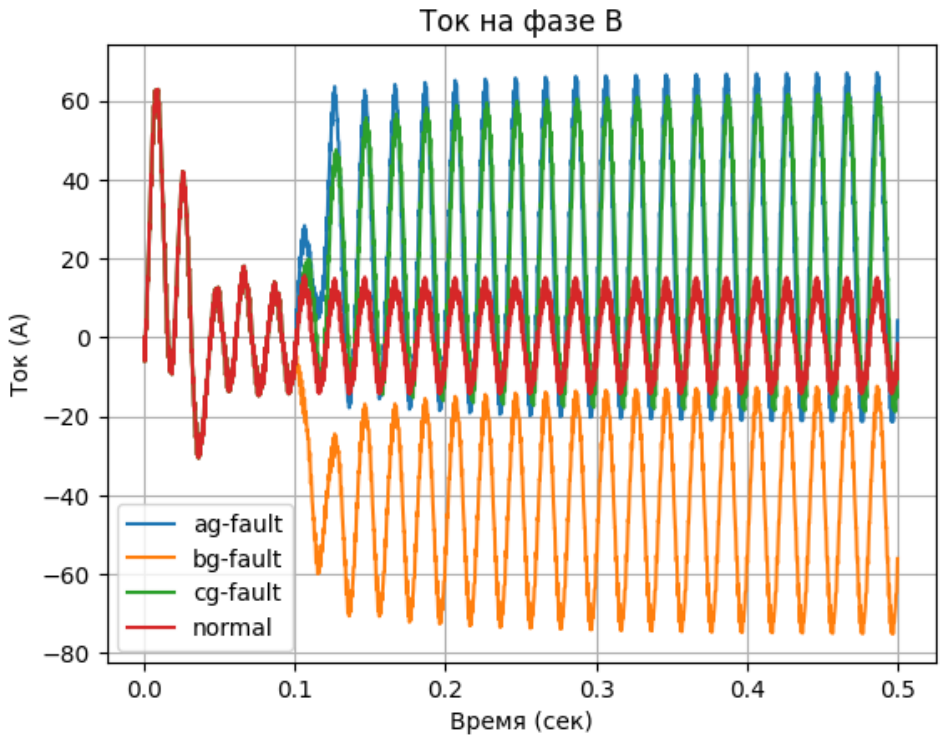

Рисунок 3. График тока второй фазы (B).

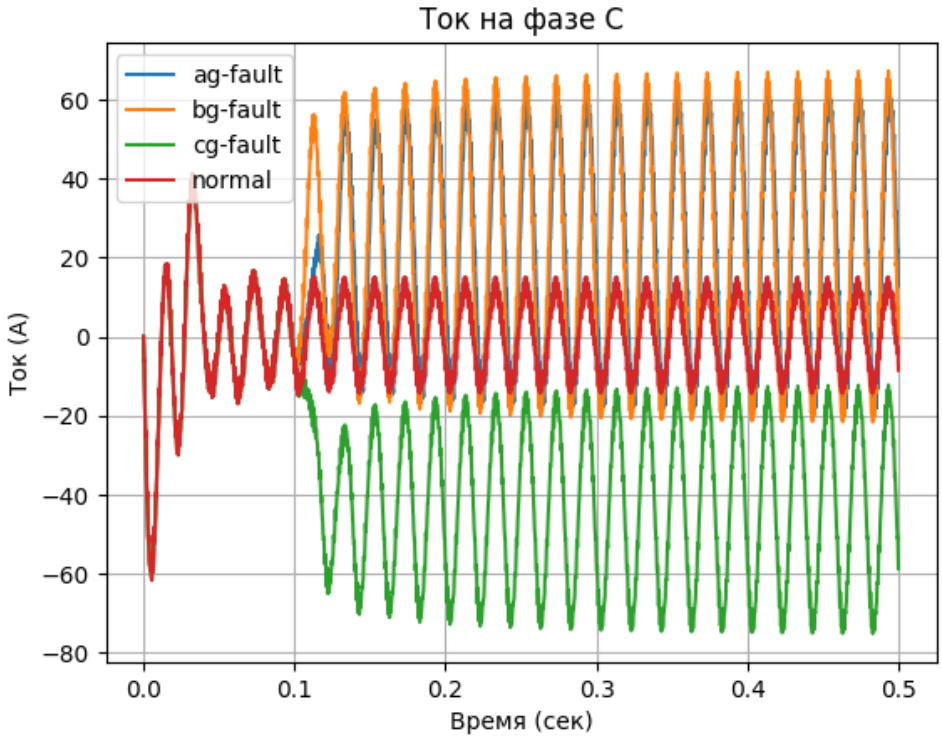

Рисунок 4. График тока третьей фазы (C).

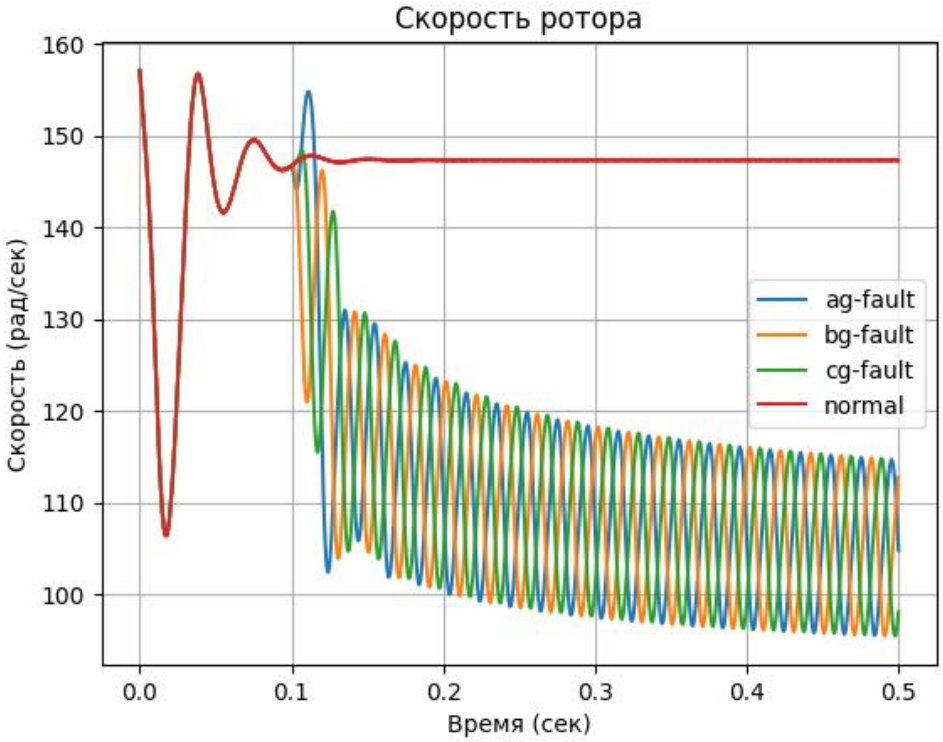

Рисунок 5. График скорости ротора асинхронного двигателя.

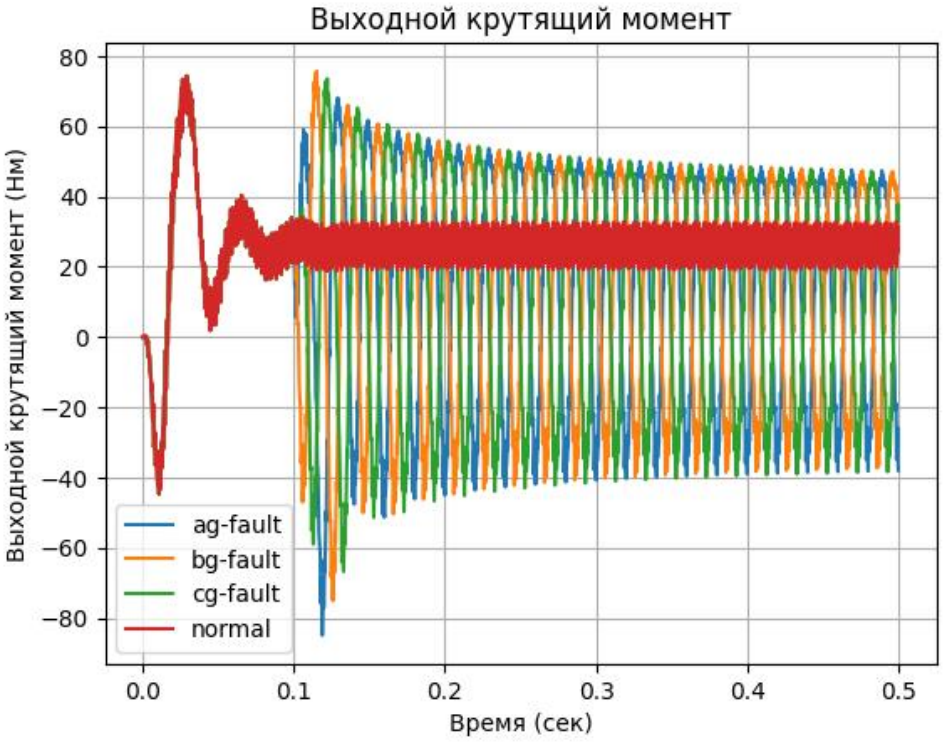

Рисунок 6. График выходного крутящего момента асинхронного двигателя.

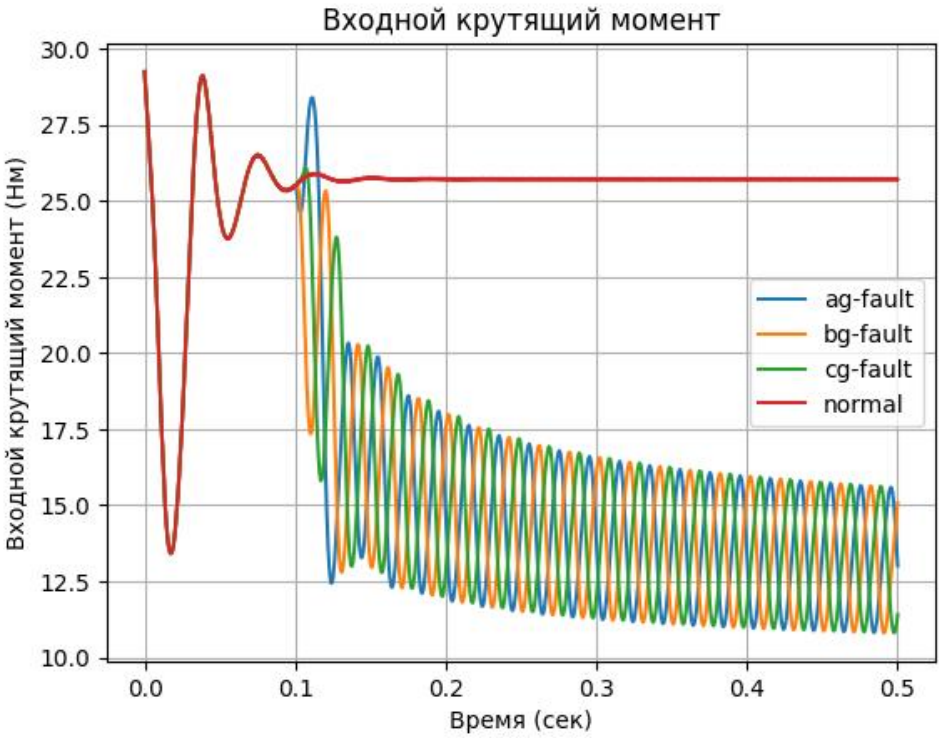

Рисунок 7. График входного крутящего момента асинхронного двигателя.

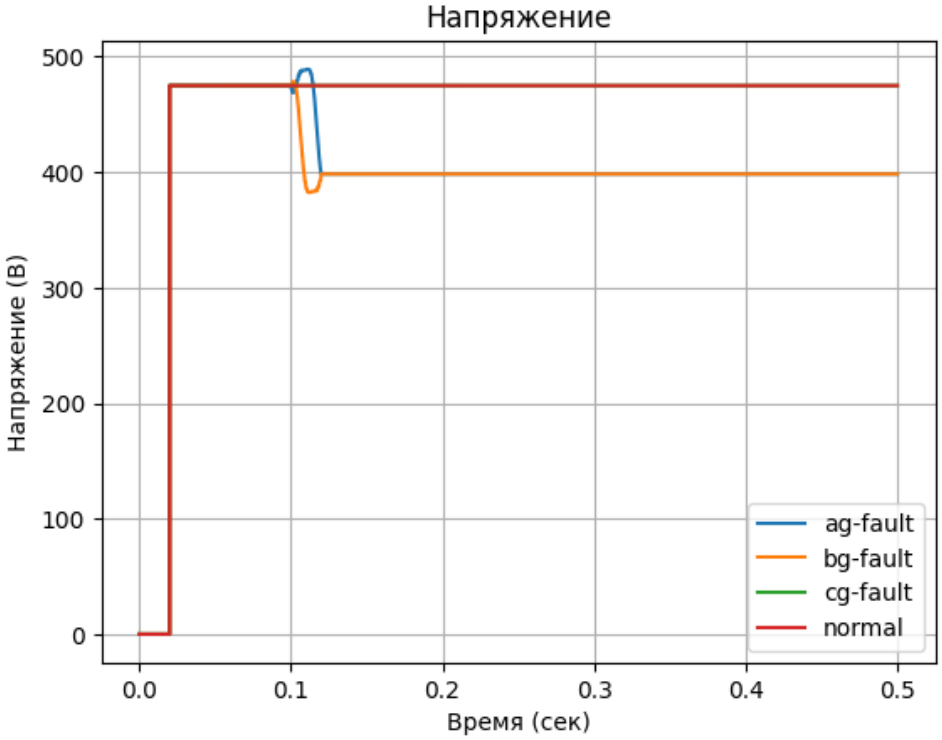

Рисунок 8. График напряжения асинхронного двигателя.

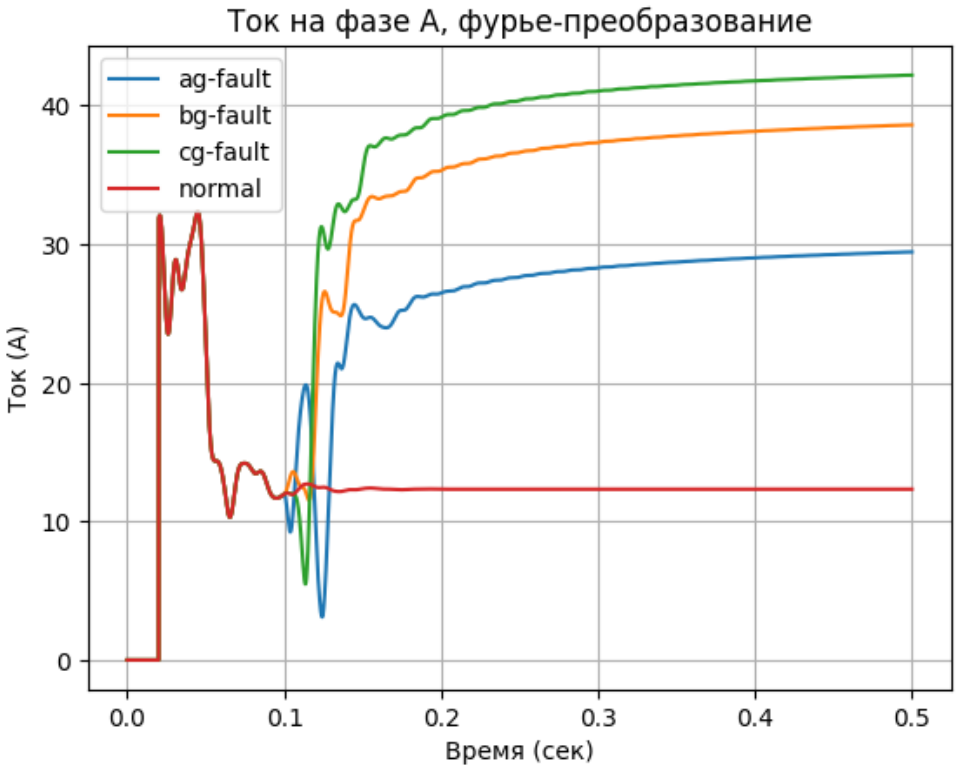

Рисунок 9. График тока первой фазы (A), частотный анализ.

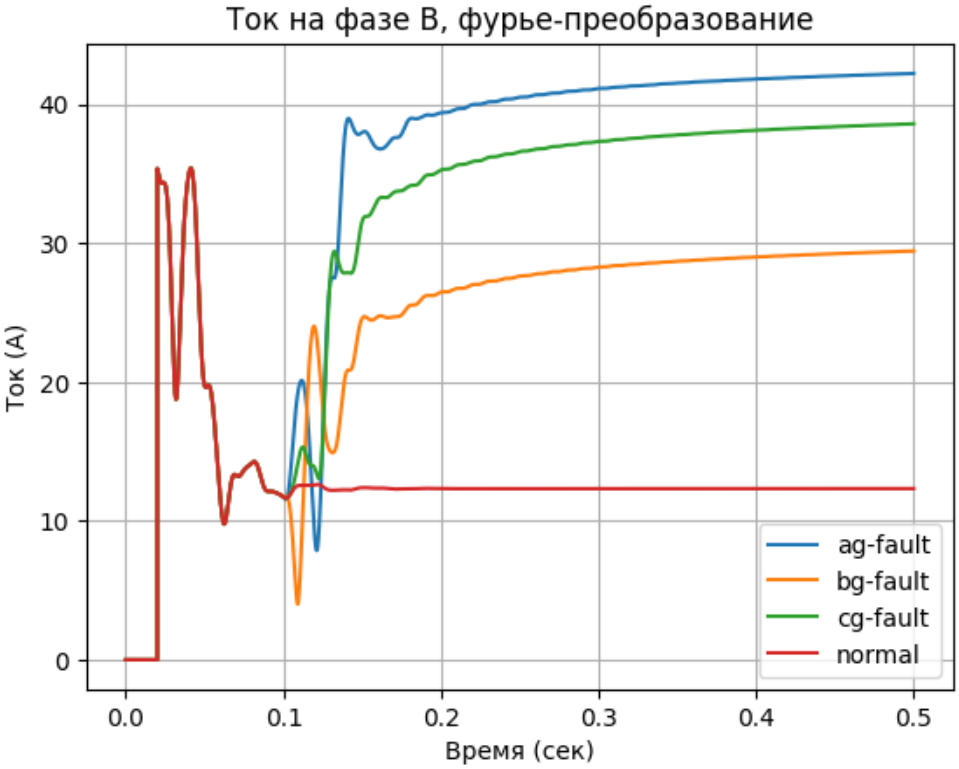

Рисунок 10. График тока второй фазы (B), частотный анализ.

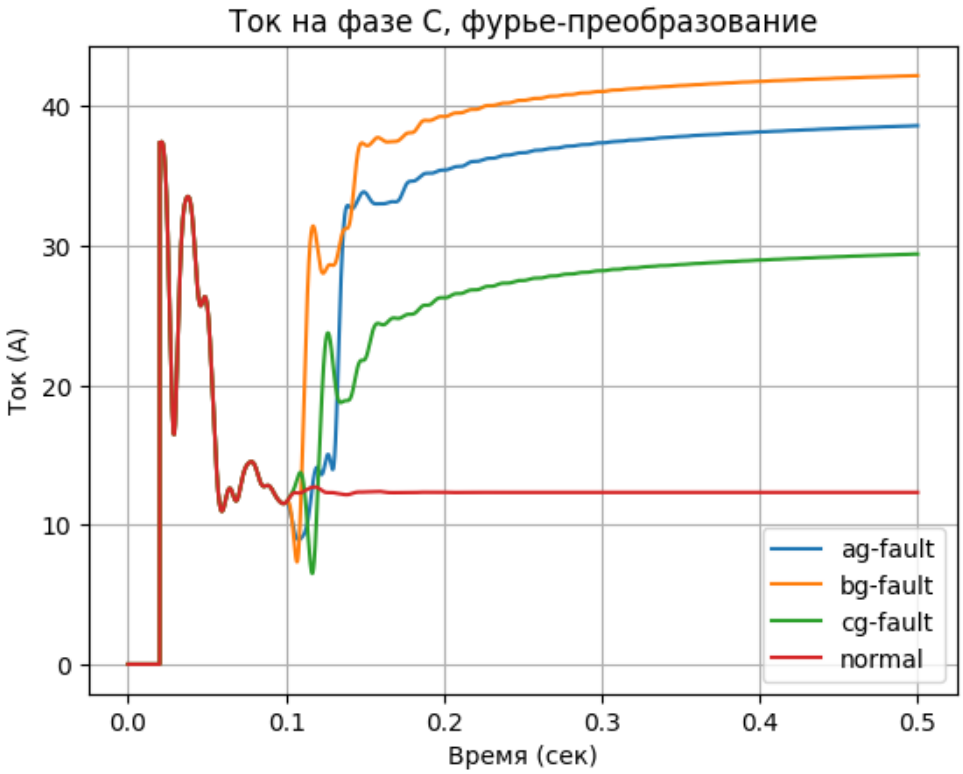

Рисунок 11. График тока третьей фазы (C), частотный анализ.

Глава 2. Модели машинного обучения.

Исходя из полученных графиков, мы выбрали следующие модели машинного обучения:

KNeighborsClassifier (KNN) – он же метод ближайших соседей. Один из самых простых методов классификации в машинном обучении. Алгоритм относит каждый новый объект к тому классу, к которому принадлежит большинство из k его ближайших *соседей*.

Алгоритм работы:

1. Загрузка данных:

   Существует набор данных. Для каждого объекта заранее известны признаки и метки классов.

2. Выборка кол-ва соседей:

   Задаем нечетное число k, это и есть количество наших соседей.

3. Расчёт расстояния:

   Для новоиспеченного объекта, который нам и необходимо классифицировать, вычисляется расстояние до всех уже известных объектов. Чаще всего используют “Евклидово расстояние”.

4. Поиск:

   Находим наиболее близких соседей (k - штук) к новому объекту.

5. Распределение:

   Новый объект приписывается к классу, который чаще всего встречается среди этих ближайших соседей.

Random Forest – случайный лес. Это ансамблевый алгоритм, он строит множество (ансамблей) независимых наборов решений (деревьев). После чего, объединяет их результаты для получения более точного и устойчивого предсказания, нежели одно надуманное дерево.

Алгоритм работы:

1. Бутстрэппинг:

   Звучит непонятно, но на самом деле довольно простой этап. Из нашего исходного обучающего датасета случайным образом создается n новых обучающих наборов (возможны повторы элементов). Что же это значит, мы разбиваем наши данные на разные наборы, они обычно такого же размера как оригинал, но некоторые объекты могут попадать в него несколько раз, а некоторые – ни разу.

2. Построение деревьев:

   Для каждого набора строим свое дерево. При построении каждого узла каждого из деревьев выбирается случайный набор признаков, например, у двигателя 5 характеристик: сила тока, напряжение, вибрация, скорость, мощность, а для одного узла (разбиения) мы возьмем 3 из них. И для каждого узла этот набор случайный. Это позволяет разнообразить деревья.

3. Голосование:

   После чего, подаем новый объект с его “характеристиками” в наш лес. Каждое из деревьев как бы сравнивает его характеристики и свои. После чего смотрим на количество голосов наших деревьев, а именно к какому классу наш элемент больше подходит.

Support Vector Machine (SVM, Метод Опорных Векторов). Нам необходимо найти гиперплоскость (линию/поверхность), которая максимально далеко отодвинута от ближайших точек разных классов (опорных векторов). Чем шире этот "буфер" (зазор), тем лучше.

Алгоритм работы:

1. Подготовка данных:
   Необходимо первым делом масштабировать признаки.

2. Выбор типа границ:
   - **linear** – если классы можно разделить прямой линией (редко).
   - **rbf** – если граница сложная, кривая (используется чаще всего).
   - **poly** – если граница полиномиальная кривая.

3. Граница:
   Алгоритм ищет разделяющую линию/поверхность (гиперплоскость). Критерий: эта граница должна быть максимально удалена от ближайших точек обоих классов.

Глава 3. Моделирование.

Определимся с количеством неисправностей (классов) для нашей модели:

- Короткое замыкание между фазой A и землей – Phase A to Ground Short Circuit Fault
- Короткое замыкание между фазой B и землей – Phase B to Ground Short Circuit Fault
- Короткое замыкание между фазой C и землей – Phase C to Ground Short Circuit Fault

Разумеется, имеется так же и нормальный режим работы двигателя, он имеет следующее обозначение: “NOM”. Итого 4 классов.

У нас огромный датасет, с определенным количеством признаков матрица корреляции признаков будет выглядеть следующим образом:

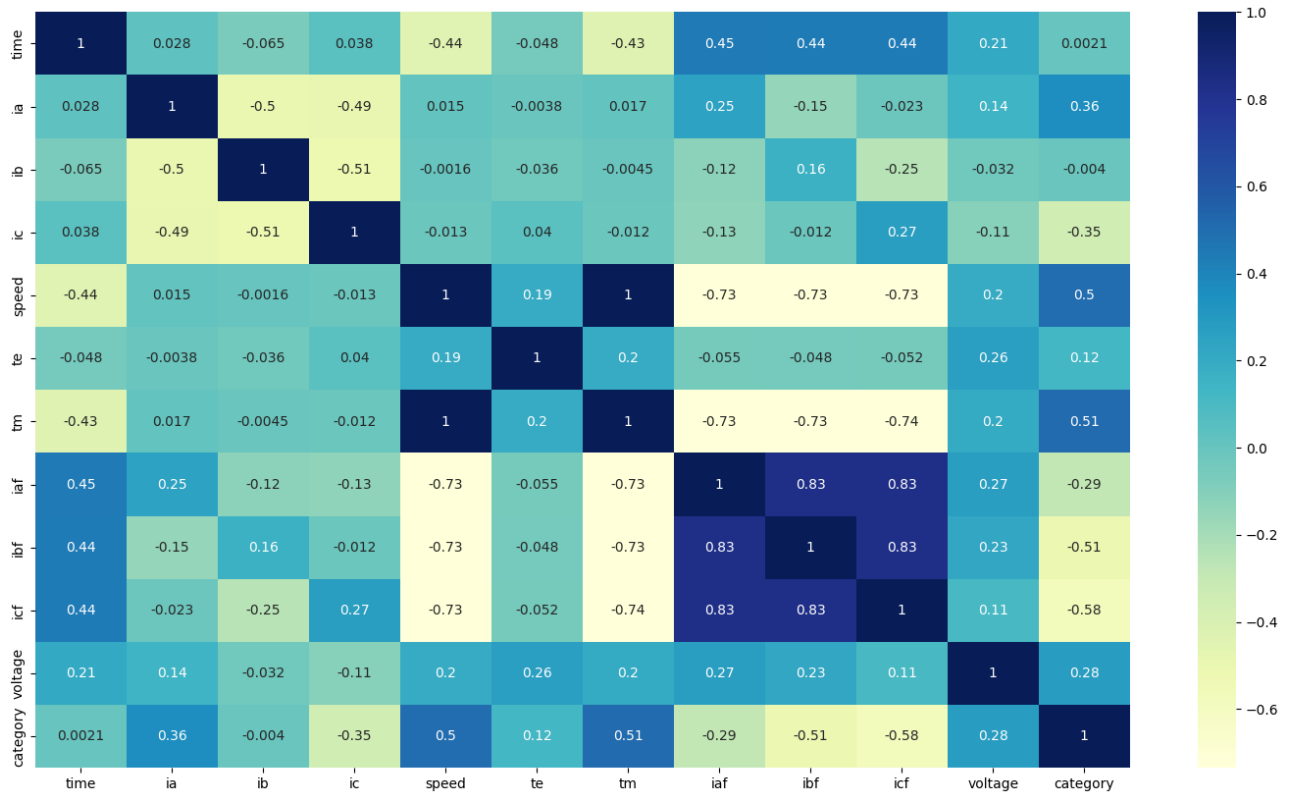

Рисунок 12. Матрица корреляции признаков.

Делим наш начальный датасет на тренировочную и тестовую выборки:

Листинг 1. Разделение на тренировочную и тестовую выборки.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y, shuffle=True
)
```

Далее инициируем модели:

Листинг 2. Инициализация KNN.

```
cv = KFold(n_splits=4)
params = [{'n_neighbors': np.arange(1, 50, 6),
           'weights': ['uniform', 'distance'],
           'p': [1,2],
           'n_jobs': [-1]
          }]
knn_clf = KNeighborsClassifier()
clf_knn = GridSearchCV(knn_clf,
                       param_grid=params,
                       scoring='f1_macro',
                       cv=cv)
clf_knn.fit(X_train,y_train)
```

Гиперпараметры:

n_neighbors - количество соседей для классификации.

Weights - как учитывать веса соседей:

- 'uniform' — все соседи равнозначны,
- 'distance' — чем ближе сосед, тем больше его "голос" весит.

Результаты KNN:

	precision	recall	f1-score	support
0	0.79	0.83	0.81	12093
1	0.83	0.82	0.82	12055
2	0.84	0.81	0.82	12080
3	0.82	0.83	0.83	12140
accuracy			0.82	48368
macro avg	0.82	0.82	0.82	48368
weighted avg	0.82	0.82	0.82	48368

Листинг 3. Инициализация RF.

```
params_forest = [{
    'n_estimators': [5, 20, 25, 30],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'bootstrap': [True, False],
    'n_jobs': [-1],
    'random_state': [42]
}]
# Инициализация моделимммм
rf_clf = RandomForestClassifier()

# GridSearch
clf_forest = GridSearchCV(
    rf_clf,
    param_grid=params_forest,
    scoring='f1_macro',
    cv=cv,
    n_jobs=-1,
    verbose=0
)
# Обучение
clf_forest.fit(X_train, y_train)м
```

Гиперпараметры:

n_estimators - количество деревьев в лесу.

max_depth - макс. глубина деревьев. none — без ограничений (деревья растут, пока могут),

bootstrap - использовать ли случайные подвыборки (бутстреп) для построения деревьев.

Результаты RF:

	precision	recall	f1-score	support
0	0.87	0.82	0.84	12093
1	0.79	0.86	0.82	12055
2	0.83	0.83	0.83	12080
3	0.87	0.83	0.85	12140
accuracy			0.84	48368
macro avg	0.84	0.84	0.84	48368
weighted avg	0.84	0.84	0.84	48368

Листинг 4. Инициализация SVM (rbf).

```
svm_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', SVC(kernel='rbf', gamma=0.1))
])

# Обучение
svm_pipeline.fit(X_train, y_train)
```

Результаты SVM (rbf):

	precision	recall	f1-score	support
0	0.86	0.83	0.85	12093
1	0.90	0.82	0.86	12055
2	0.86	0.82	0.84	12080
3	0.75	0.89	0.81	12140
accuracy			0.84	48368
macro avg	0.84	0.84	0.84	48368
weighted avg	0.84	0.84	0.84	48368

Листинг 5. Инициализация SVM (linear).

```
svm_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', SVC(kernel='linear'))
])

# Обучение
svm_pipeline.fit(X_train, y_train)
```

Результаты SVM (linear):

	precision	recall	f1-score	support
0	0.88	0.81	0.84	12093
1	0.92	0.81	0.86	12055
2	0.88	0.81	0.84	12080
3	0.71	0.91	0.80	12140
accuracy			0.83	48368
macro avg	0.85	0.83	0.84	48368
weighted avg	0.85	0.83	0.84	48368

Листинг 6. Инициализация SVM (poly).

```
svm_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', SVC(kernel='poly', degree=5))
])

# Обучение
svm_pipeline.fit(X_train, y_train)
```

Результаты SVM (poly):

	precision	recall	f1-score	support
0	0.88	0.83	0.85	12093
1	0.88	0.82	0.85	12055
2	0.86	0.82	0.84	12080
3	0.76	0.88	0.81	12140
accuracy			0.84	48368
macro avg	0.84	0.84	0.84	48368
weighted avg	0.84	0.84	0.84	48368

Также мы решили попробовать инициировать самодельную нейронную сеть:

Листинг 7. Инициализация Франкенштейна.

```
# Метки классов
label_map = {
    "input_normal.csv": 0,
    "input_ag.csv": 1,
    "input_bg.csv": 2,
    "input_cg.csv": 3
}

# Загрузка данных
dfs = []
for fname, label in label_map.items():
    df = pd.read_csv(fname)
    df["label"] = label
    dfs.append(df)

data = pd.concat(dfs, ignore_index=True)

# Выбор признаков и меток
features = ['ia', 'ib', 'ic', 'speed', 'te', 'tm', 'iaf', 'ibf', 'icf',
'voltage']
X = data[features].values
y = data["label"].values

# Нормализация
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

# Преобразование в тензоры
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.long)

# DataLoader
train_dataset = torch.utils.data.TensorDataset(X_train, y_train)
test_dataset = torch.utils.data.TensorDataset(X_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64)
```

Листинг 8. Инициализация самой модели.

```
class FaultClassifier(nn.Module):
    def __init__(self, input_dim, hidden_dim=64, output_dim=4):
        super(FaultClassifier, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, output_dim)
        )
    def forward(self, x):
        return self.model(x)

model = FaultClassifier(input_dim=X_train.shape[1])
```

Результат:

	precision	recall	f1-score	support
0	0.98	0.80	0.88	12140
1	0.96	0.81	0.87	12093
2	0.64	0.98	0.77	12055
3	0.97	0.80	0.88	12080
accuracy			0.84	48368
macro avg	0.89	0.84	0.85	48368
weighted avg	0.89	0.84	0.85	48368

Вывод.

Выбор наилучшей модели основывается на основных метриках:

- Accuracy
- F1-score

Итак, заметим, что все используемые модели показали нормальную эффективность (accuracy от 0.82 до 0.84), однако наилучший показатель f1-score принадлежит нашей самодельной нейронной сети, что чуть выше, чем у RF (0.85 и 0.84 соответственно).

Несмотря на это, наша нейронная сеть все же имеет свои недостатки, она сложнее в инициализации и настройке, а также слегка нестабильна. Что мы можем наблюдать на показателе precision (точности), для одного из классов (фаза B - земля).

Нейронная сеть из двух скрытых слоёв (ReLu) способна аппроксимировать сложные зависимости между признаками, тогда как в RF и SVM эти зависимости моделируются жёстче (линейно или через ядро). В дальнейшем можно рассмотреть гибридные нейронные сети или оптимизацию архитектуры сети (добавление регуляризации), чтобы повысить сбалансированность точности и полноты предсказаний.