МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки / специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль)/специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка методики диагностики токов утечки CAN-шины по току потребления драйвера»

Обучающийся А.А. Давыдова
(Инициалы Фамилия) (личная подпись)

Руководитель С.В. Митин.
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант к.п.н.,Т.С. Якушева
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2022

Аннотация

Бакалаврская работа включает: 60 страниц, 45 рисунков, 1 таблицу, 4 раздела, 20 использованных источников, 1 приложение.

Цель работы – разработка скрипта на основании проведенных исследований, который выполнит подготовку данных и создаст модель для определения токов (сопротивления) утечек по входным данным.

Объект исследования – выбор и проработка программных оптимальных решений для обработки входных данных и выбор модели для определения тока(сопротивления) утечек.

Предмет исследования – поиск подходов к обработке входных данных и способов построения модели для определения тока утечки или сопротивления утечки.

Полученные результаты – разработан и реализован алгоритм, определяющий состояние шины CAN, исходя из входных данных.

Рекомендация внедрения результатов работы – полученный алгоритм в дальнейшем может быть использован для внедрения его в различные диагностические приборы, которые используются для диагностики шины CAN, например на станциях технического обслуживания автомобилей.

Область применения – разработанный алгоритм возможно применять при проведении различных диагностических работ с шиной CAN.

Значимость работы – позволяет наиболее эффективно, диагностировать шину CAN в составе различных устройств или группы устройств.

Abstract

The title of the graduation work is «Development of a methodology for diagnosing leakage currents of the CAN bus by the current consumption of the driver».

The graduation work consists of an explanatory note on 60 pages, introduction, four parts including 45 illustrations, 1 table, the list of 20 references.

The aim of this work is development of a script based on the conducted research, which will prepare the data and create a model for determining the leakage currents (resistance) from the input data.

The object of the study is selection and development of software optimal solutions for processing input data and selection of a model for determining the leakage current (resistance).

The subject of the study is search for approaches to processing input data and ways to build a model to determine the leakage current or leakage resistance.

The result of the graduation work is the developed and implemented algorithm that determines the state of the CAN bus based on input data.

It is highly recommended to use this algorithm to implement it in various diagnostic devices that are used to diagnose the CAN bus, for example, at car service stations.

The results of this work can be applied when carrying out various diagnostic work with the CAN bus.

The relevance of this work lies in the most effective diagnostics of the CAN bus as part of various devices or a group of devices.

Содержание

Введение

Данная работа посвящена решению вопроса диагностики CAN шины. Необходимо определить сопротивления утечки каждой линии шины на общий провод, либо на шину питания бортовой сети автомобиля. Решение о значении сопротивления утечки принимается на основании токов вытекающего CurrentHigh и втекающего CurrentLow в драйвер CAN шины. Значения этих токов фиксируется в процессе измерения дифференциального сопротивления CAN шины. С помощью внешней цепи возможно имитировать различные варианты замыкания линий шины как на общий провод, так и на шину питания, имеются следующие варианты сопротивления утечки: 100 Ом, 1 кОм, 10 кОм, 100 кОм.

На основании полученных данных предполагается разработать алгоритм и ПО для определения сопротивления утечки линий шины если таковые имеются.

Целью работы является разработка скрипта на основании проведенных исследований, который выполнит подготовку данных и создаст модель для определения токов (сопротивления) утечек по входным данным.

Объектом исследования является выбор и проработка программных оптимальных решений для обработки входных данных и выбор модели для определения тока(сопротивления) утечек.

Предметом исследования работы является поиск подходов к обработке входных данных и способов построения модели для определения тока утечки или сопротивления утечки.

Данный алгоритм и ПО предполагается внедрить в устройство диагностики, что позволит снизить затраты времени на диагностику CAN шины и необходимость дополнительного измерительного оборудования для выполнения этой задачи.

1 Принцип работы шины CAN

Controller area network (CAN) — стандарт промышленной сети, ориентированный, прежде всего, на объединение в единую сеть различных исполнительных устройств и датчиков. Режим передачи — последовательный, широковещательный, пакетный. [1]

CAN разработан компанией Robert Bosch в середине 1980-х и в настоящее время широко распространён в промышленной автоматизации, технологиях домашней автоматизации («умного дома»), автомобильной промышленности и многих других областях. Стандарт для автомобильной автоматики. [1]

Данная шина обычно используется для обмена данными в тех системах, где быстродействие является важным условием при проектировании. Например, в легковых автомобилях, связывая контроллеры различных систем, таких как контроллер системы управления двигателем (КСУД), контроллер управления трансмиссией, контроллера антиблокировочной системы торможения (АБС) и т.п.

Конструктивно шина состоит из двух проводов, которые именуются как CAN_H (High) и CAN_L (Low) и представляют собой витую пару.

Шина имеет два устойчивых состояния: доминантное и рецессивное.

В доминантное состояние шина переходит, когда один из контроллеров, находящихся в сети, инициирует сеанс передачи данных. При этом потенциал на проводе CAN_H повышается на один вольт относительно рецессивного состояния, а на CAN_L уменьшается на один вольт.

В рецессивном состоянии CAN_H и CAN_L имеют примерно одинаковый потенциал (примерно 2,5 вольта). В данном состоянии шина находится всё время, до тех пор, пока один участников сети на начнёт

передачу данных, после окончания передачи данных, шина снова вернётся в рецессивное состояние. На рисунке 1 показаны оба состояния.

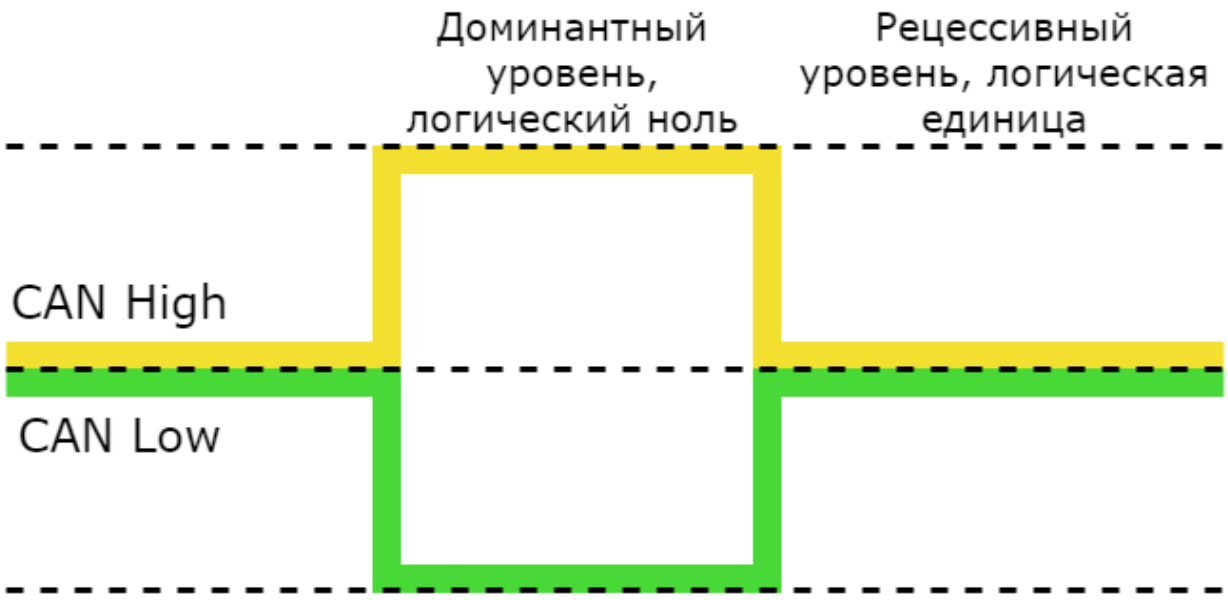

Рисунок 1 – Логические уровни шины CAN при доминантном и рецессивном состоянии.

Шина CAN так же является помехозащищённой, так как обрабатывается не сам сигнал с линий High и Low, а его разница. При рецессивном состоянии шины разница сигнала High и Low близка к нулю, а при доминантном максимальна.

При возникновении электромагнитных помех в шине CAN, всплеск напряжения на обоих проводах будет одинаковый, т.е. помеха будет синфазной. Получается, что на обоих проводах будет присутствовать наведённый импульс, но разность потенциалов между High и Low при этом не изменится. На рисунке 2 изображён сигнал, на котором присутствует электромагнитная помеха, разность потенциалов при ней остаётся постоянной.

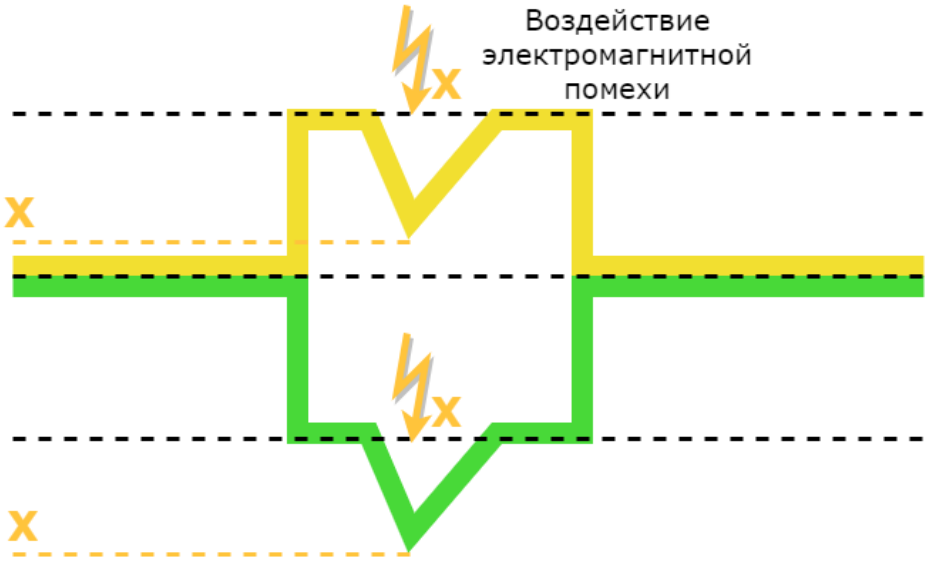

Рисунок 2 – Сигнал на линиях High (Желтый) и Low (Зелёный) при наличии электромагнитной помехи

Данная особенность является большим преимуществом шины CAN и позволяет эффективно подавлять электромагнитные помехи. Для обработки реального сигнала в схемах используется дифференциальный усилитель. На рисунке ниже показана обработка сигнала дифференциальными усилителем.

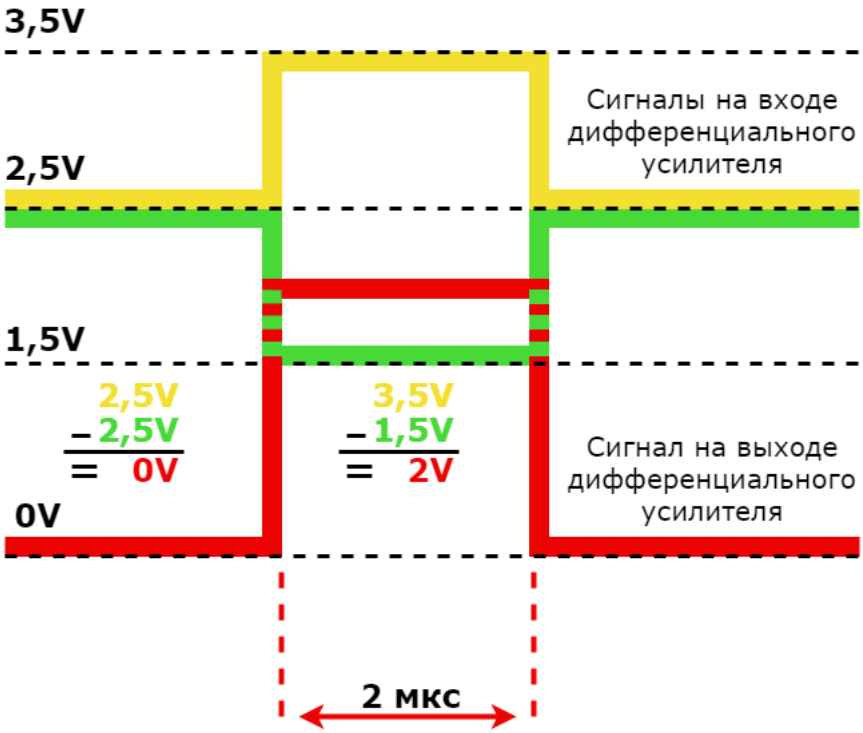

Рисунок 3 – Сигнал на выходе дифференциального усилителя (Красный)

Большинство разработчиков электронных устройств с шиной CAN придерживаются скорости 500 кбод, длительность импульса при этом равна 2 мкс.

Выводы по разделу 1.

В данном разделе мы описали принципы работы шины CAN, обозначили её особенности, которые понадобятся для описания аварийных реакций и разработки методики диагностирования.

2 Диагностика шины CAN

Для диагностирования шины CAN было разработано устройство, которое может имитировать различные аварийные ситуации на шине. Такими аварийными ситуациями являются:

- Замыкание на «+» питания
- Замыкание на землю «GND»
- Обрыв цепи.

Электрическая схема устройства для исследования токов утечки на шине CAN представлена на рисунке ниже.

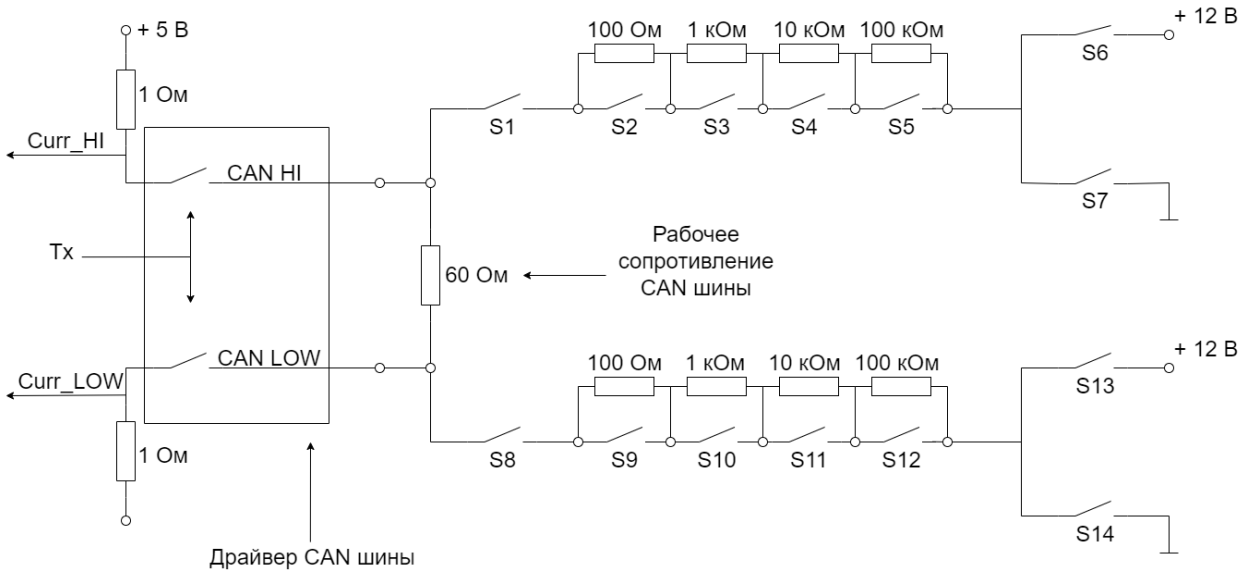

Рисунок 4 – Электрическая схема устройства для исследования токов утечки на шине CAN

Данное устройство в своей основе стоит из CAN трансивера (приёмо-передатчика), резисторов для измерения тока, а также из группы резисторов,

которые будут замыкать линии CAN_H и CAN_L на питание и землю соответственно для имитирования различных токов утечки.

В роли CAN трансивера выбрана микросхема TJA1020 которая представляет собой интерфейс между микроконтроллером, который будет непосредственно принимать и передавать сообщения по CAN и физической шиной CAN. В цепь питания данной микросхемы устанавливаются резисторы для измерения тока утечки, один на + питания, другой на землю. Ток утечки будет измеряться с помощью аналого-цифрового преобразователя (АЦП). Сырые данные с АЦП будут переведены в вольты, ток будет рассчитан по закону Ома, исходя из сопротивления резистора.

Для регулирования токов утечки, в цепь нагрузки установлена группа из 8 резисторов, которые коммутируются ключами. При исследовании токов утечки, пользователь может как переключать резисторы между собой, так и добавлять их в цепь, таким образом регулируя сопротивление и в конечном счёте ток утечек.

На линии CAN_H группа резисторов состоит из 4 штук и имеет номиналы 100 Ом, 1 кОм, 10 кОм, 100 кОм. Резисторы и ключи подключены по схеме аналогичной линии CAN_L.

Поскольку каждая линия шины CAN может иметь утечку как на питание, так и на землю, устройство для диагностики так же оборудовано ключами, которые замыкают линии CAN_H и CAN_L на питание или землю, через группу резисторов, изображенных на рисунке 4.

Испытания будем производить при кратковременном включении драйвера 200-300 мкс, в виду быстрого нарастания токов в исследуемых цепях. На вход драйвера TX, будем подавать логический 0 на время, равное 200-300 мкс, за этот период АЦП получит 16 выборок по току на линиях High

и Low. Нагрузкой на шину будет служить резистор 60 Ом, что является рабочим сопротивлением шины CAN.

2.1 Нормальное состояние шины CAN

Нормальным состоянием шины CAN считается такое состояние шины, при котором отсутствуют утечки тока (т.е. ток линии High и Low равны) на обеих линиях (CAN_H и CAN_L), подключена нагрузка и балансировочный резистор. Для проведения тестирования на стенде были отключены все резисторы, которые отвечают за регулирование утечек на линиях. На токоизмерительных резисторах был измерен ток. График токов на линиях High и Low изображен на рисунке ниже.

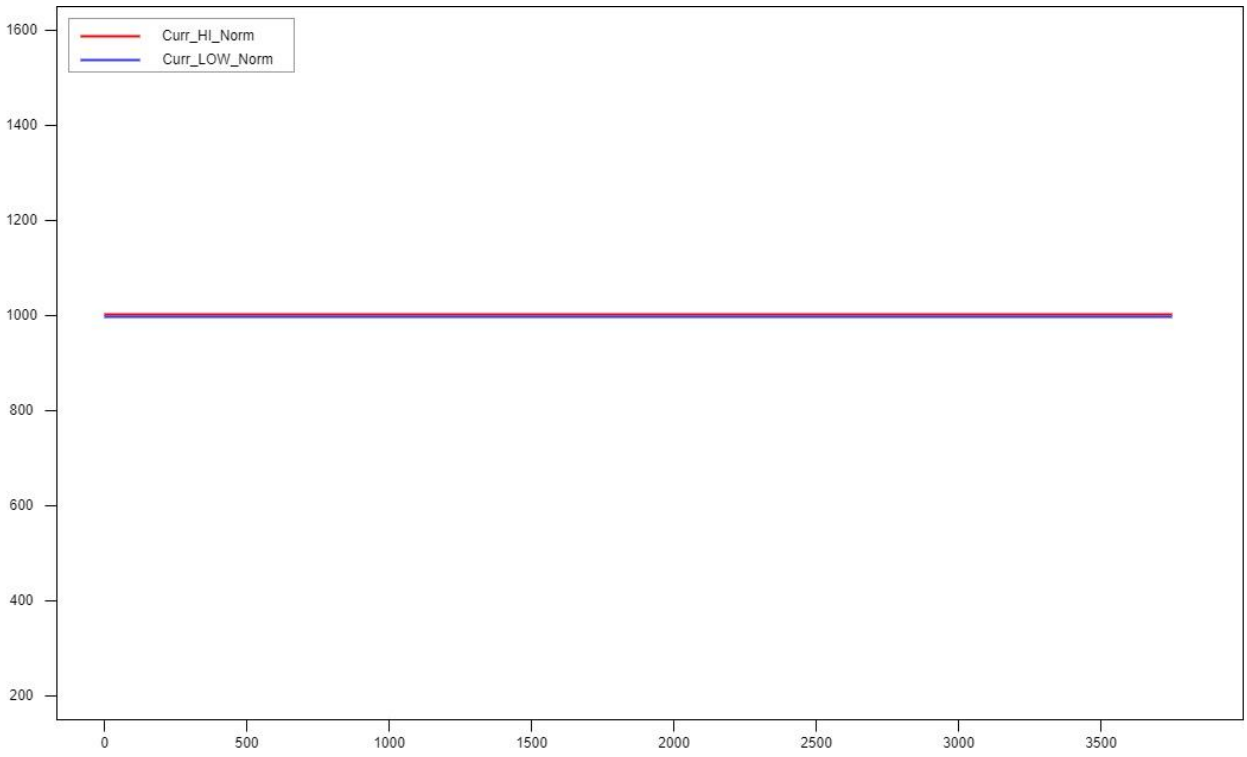

Рисунок 5 – Токи на линиях High и Low при нормальном состоянии шины CAN.

На рисунке 5 виден ток с линий High и Low, который при нормальном состоянии шины остаётся неизменным на протяжении всего периода испытаний.

2.2 Аварийные ситуации на шине CAN

Каждая линия шины CAN (CAN_H и CAN_L) может находиться в трёх состояниях:

- Замыкание на «+» питания
- Замыкание на землю «GND»
- Обрыв цепи.

Однако шина в целом может иметь множество различных состояний, которые складываются из совокупности состояний каждого плеча драйвера.

Наиболее ценными для исследования представляют собой те состояния, где обе линии находятся вне нормального состояния, а именно замкнуты на землю или питания.

Для испытания шины на различные аварийные реакции, будем замыкать линии High и Low на землю и питание через четыре резистора, чтобы увеличивать ток утечки. На рисунке 6 изображена диаграмма токов, на которой в интервале времени 500 – 1000 линии были замкнуты на землю с наибольшим сопротивлением (100 кОм), а после сопротивление уменьшали до 0. На интервале времени 1000 – 1500 линия Low была замкнута на 12 вольт, аналогично переключая сопротивления со 100 кОм до 0. На интервале 1500 – 2000 только линия High была замкнута на 12 вольт, так же уменьшая сопротивление. На следующем интервале линию Low замкнули на землю, а High на 12 вольт, затем наоборот Low на питание, а High на землю. На двух

последних интервалах линии Low и High замыкались либо на питание, либо на землю.

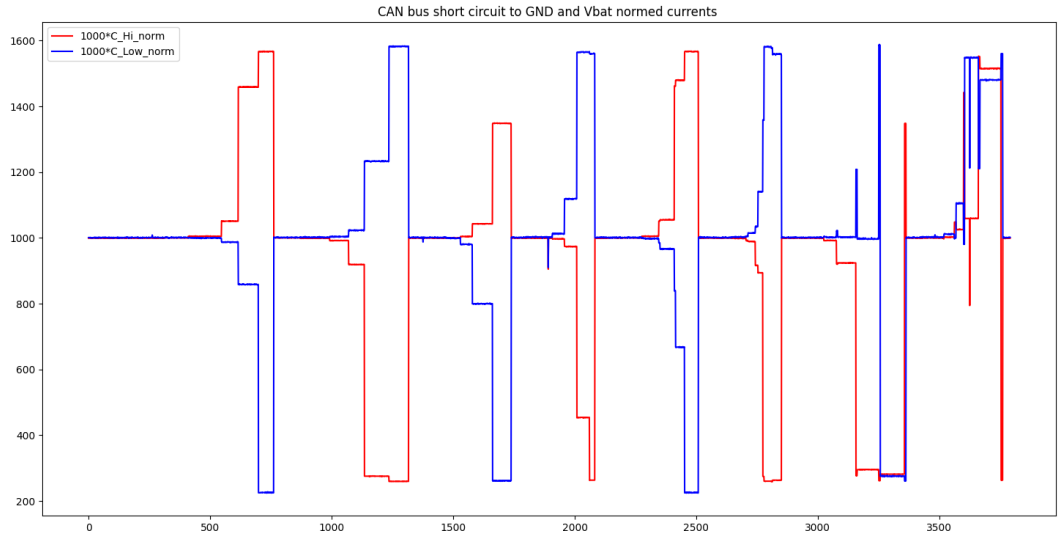

Рисунок 6 – Диаграмма токов при различных аварийных реакциях

Так как предполагается использовать результаты тестирования шины CAN как входные данные для машинного обучения, то рассмотрим аварийные ситуации описанные на рисунке 6 подробно. Далее на графиках будут изображены диаграммы токов при различных паразитных сопротивлениях. По оси абсцисс от 1 до 6 указывается порядковый номер резистора на стенде, который имитирует паразитное сопротивление в шине.

Рассмотрим ситуацию, когда, линия Low замкнута на землю. График токов изображен на рисунке ниже. График построен по 6 точкам (ось абсцисс), которые соответствуют подключаемым сопротивлениям.

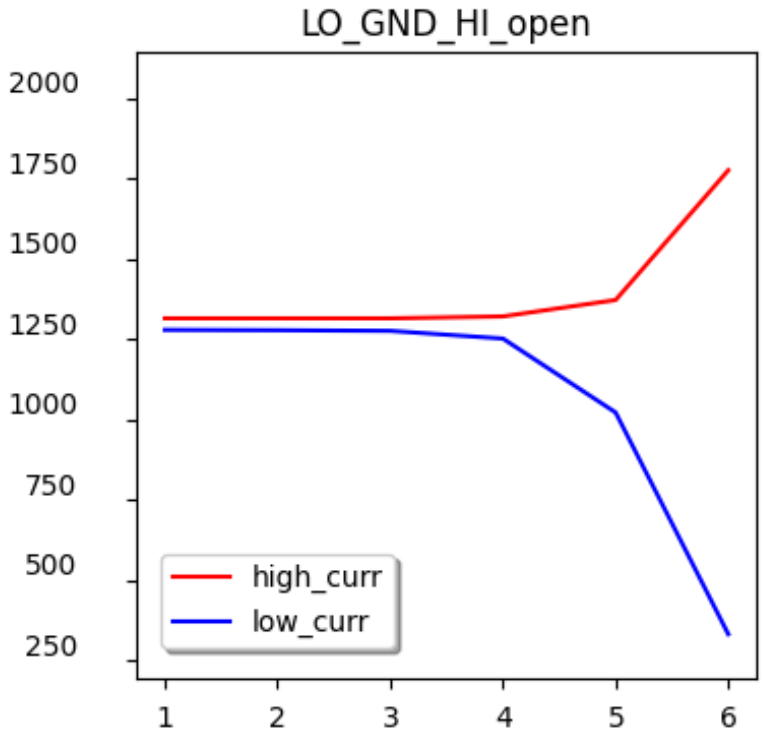

Рисунок 7 – Диаграмма токов при замыкании линии Low на землю

На данной диаграмме видно, что ток драйвера увеличивается в обоих плечах. Но ток в нижнем плече увеличивается с большей скоростью. Начальное значение нормировано, поэтому превышает ток нормального состояния и равно 1250, данное значение условное и является по сути значением выхода АЦП.

Рассмотрим так же случай, когда линия Low замкнута на +12 вольт. Диаграмма токов изображена на рисунке ниже, и аналогично рисунку 7 построена по 6 точкам, соответствующих сопротивлений.

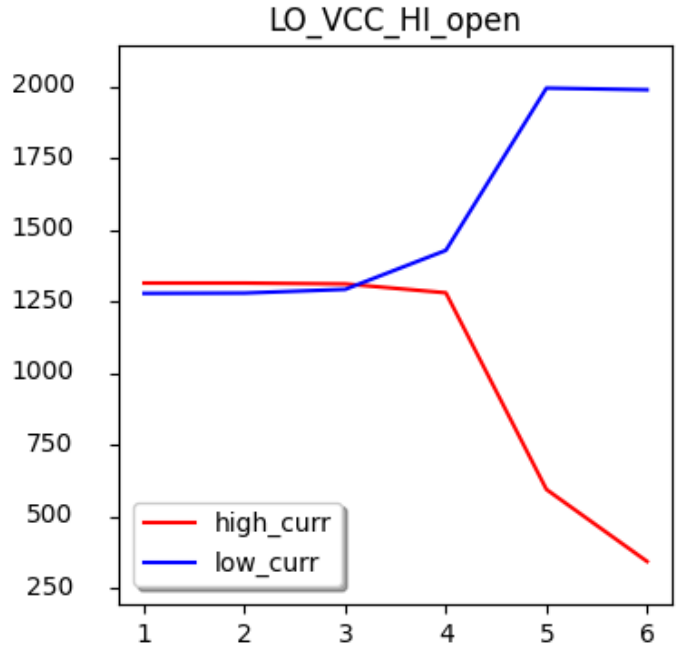

Рисунок 8 – Диаграмма токов при замкнутой на питание линии Low

На данной диаграмме видно, что ток драйвера по цепи земли лавинно увеличивается в сторону тока короткого замыкания.

Исходя из вышеописанных диаграмм можно сделать вывод, что замыкание одной линии шины (High или Low) на питание или землю приводит к возрастанию тока соответствующего плеча драйвера до тока короткого замыкания (КЗ). Поскольку для защиты устройства, в цепях нагрузки стоят резисторы, то ток КЗ не стремится к бесконечности, а равен току, проходящему через этот резистор.

Стоит так же обратить внимание на случаи, когда обе линии шины замкнуты либо на землю, либо на питание. Для этого на испытательном стенде замкнём обе линии сначала на питание, а потом на землю. Полученный результат в виде диаграммы токов, изображен на рисунках ниже.

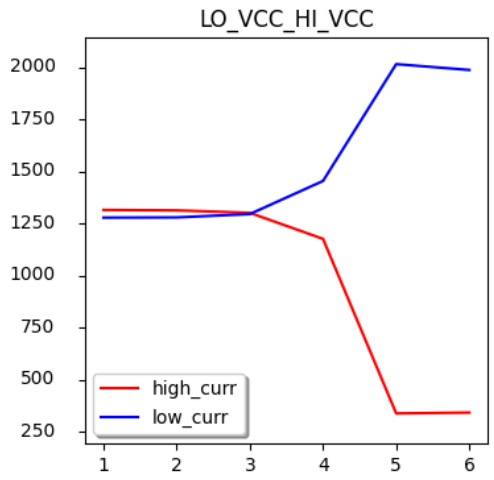

Рисунок 9 – Диаграмма токов при замыкании обеих линий шины CAN на питание

В данном случае видны отличия, оба тока нарастают лавинно. Проведём такой же эксперимент, но обе линии шины замкнём на землю. Диаграмма токов изображена на рисунке ниже.

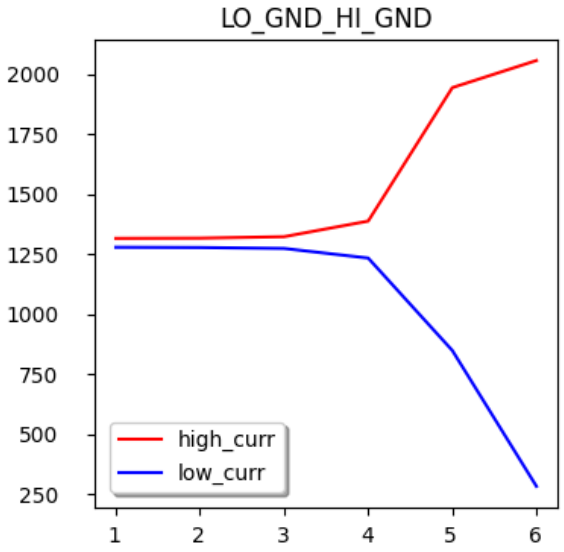

Рисунок 10 - Диаграмма токов при замыкании обеих линий шины CAN на землю

При замыкании на землю реакция токов противоположная, относительно замыкания на питание. Ток верхнего плеча увеличивается, а нижнего уменьшается. Скорость изменения токов ниже, чем при замыкании на питание.

Далее рассмотрим случаи, когда лини замкнуты противоположно, т.е. Low на землю, а High на питание и наоборот Low на питание, а High на землю. Диаграммы токов представлены на рисунках ниже.

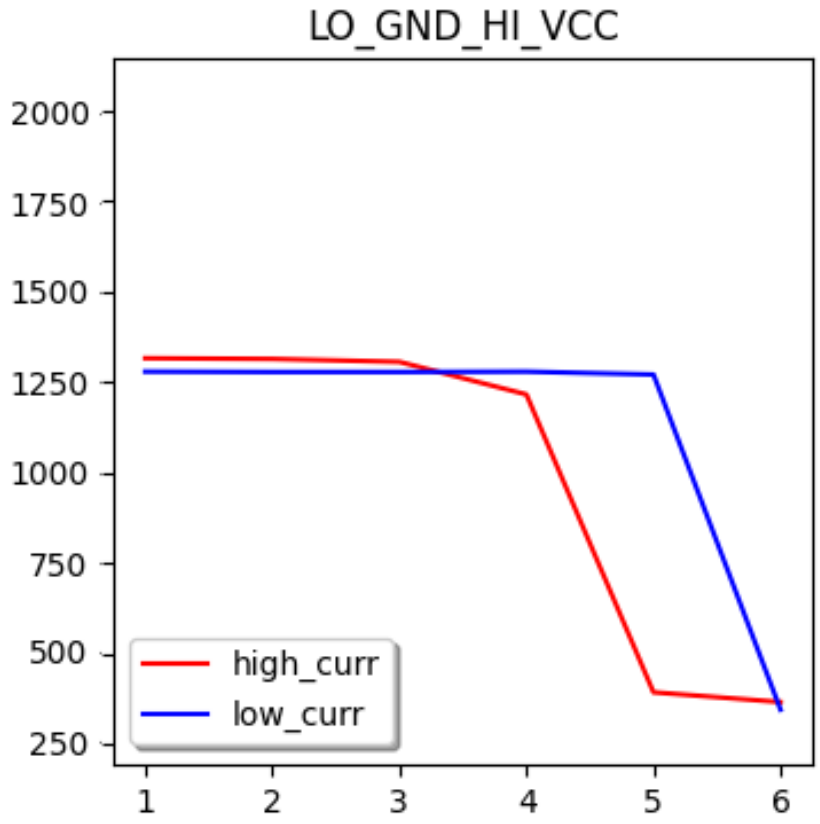

Рисунок 11 - Диаграмма токов при замыкании линии Low на землю, а линии High на питание

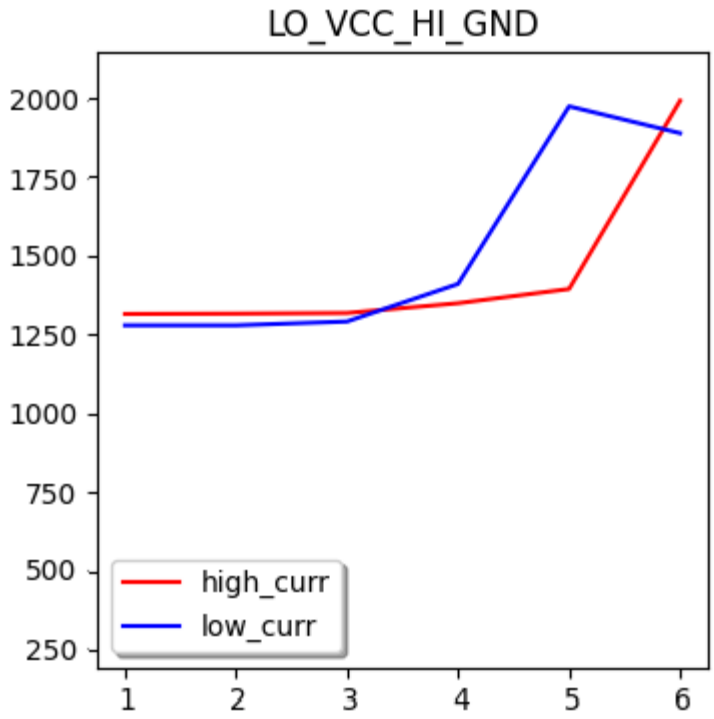

Рисунок 12 - Диаграмма токов при замыкании линии Low на питание, а линии High на землю

Токовые диаграммы в данных случаях имеют противоположный характер, на диаграмме 11 ток уменьшается по обеим линиям, а на диаграмме 12 увеличивается, так же по обеим линиям. Характер изменения тока в случае замыкания линии High на питание, а линии Low на землю лавинный, в отличии от токов на диаграмме 12.

Так же рассмотрим замыкания линии High на землю и питания, при нормальном состоянии линии Low. Диаграммы токов представлены на рисунках ниже.

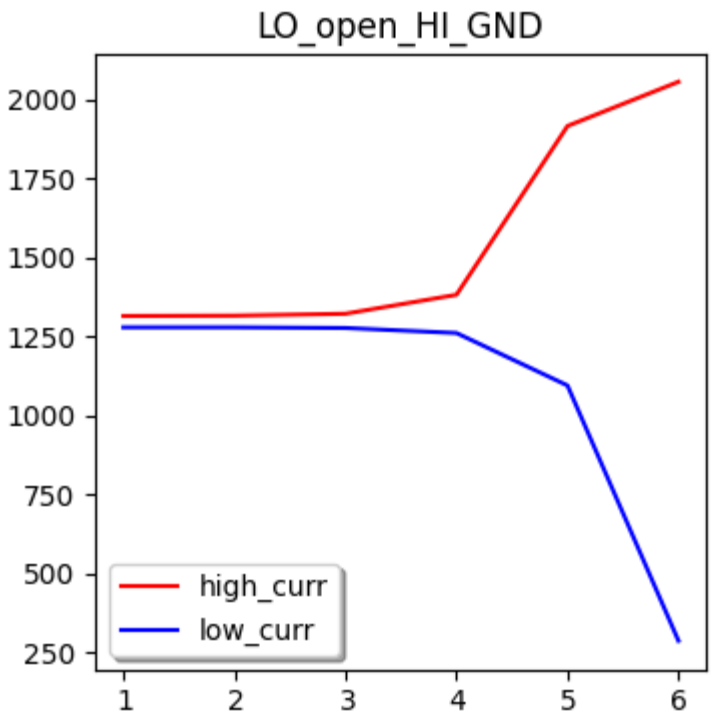

Рисунок 13 - Диаграмма токов при замыкании линии High на землю

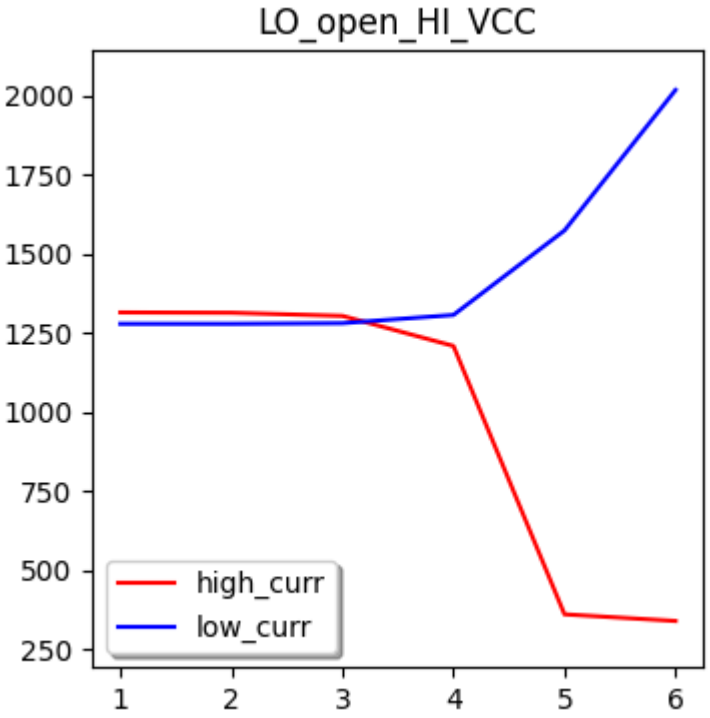

Рисунок 14 - Диаграмма токов при замыкании линии High на питание

В данных случаях ситуация аналогично замыканию на линии Low, токи возрастают лавинно, стремясь к токам КЗ.

Собранные данные позволяют построить математическую модель, и сгенерировать тестовый набор данных для проверки алгоритмов, затем подбирать алгоритмы для определения сопротивления утечки.

2.3 Определение состояния шины CAN и диагностирование неисправности с помощью алгоритмов машинного обучения

Для определения состояния шины необходимо подобрать наиболее подходящий алгоритм машинного обучения. Из огромного множества алгоритмов были выбраны наиболее популярные алгоритмы машинного обучения:

- Метод k-ближайших соседей (KNN)
- Логистическая регрессия или логит-модель (LR)
- Линейный дискриминантный анализ (LDA)
- Классификация и регрессия с помощью деревьев (CART)
- Метод опорных векторов (SVM)

Метод k-ближайших соседей удобен тем, что лёгок в реализации (нет необходимости выстраивать модель и настраивать большое количество параметров), не имеет чувствительности к выбросам загружаемых для обучения данных, а также является универсальным и может быть использован как для задачи классификации, так и для задачи регрессии.

При использовании данного метода для классификации объект присваивается тому классу, который является наиболее распространённым среди k соседей данного элемента, классы которых уже известны. В случае использования метода для регрессии, объекту присваивается среднее

значение по k ближайшим к нему объектам, значения которых уже известны. [2]

Алгоритм может быть применим к выборкам с большим количеством атрибутов (многомерным). Для этого перед применением нужно определить функцию расстояния. Функцией расстояния обычно принимают Евклидову метрику. [2]

Евклидова метрика – расстояние между двумя точками в евклидовом пространстве. Такое расстояние можно вычислить по теореме Пифагора. Получается, что Евклидова метрика — это кратчайшее расстояние между двумя точками. Формула для нахождения расстояние между двумя точками представлена ниже.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \quad (1)$$

Так же важным фактором метода является нормализация. Разные атрибуты обычно обладают разным диапазоном представленных значений в выборке. К примеру, атрибут А представлен в диапазоне от 0.01 до 0.05, а атрибут Б представлен в диапазоне от 500 до 1000. В таком случае значения дистанции могут сильно зависеть от атрибутов с большими диапазонами. Поэтому данные в большинстве случаев проходят через нормализацию. При кластерном анализе есть два основных способа нормализации данных: MinMax-нормализация и Z-нормализация. [2]

Формула MinMax-нормализации представлена ниже.

$$x' = \frac{x - min[X]}{max[X] - min[X]} \quad (2)$$

Формула Z-нормализации представлена ниже,

$$x' = \frac{x - M[X]}{\sigma[X]} \quad (3)$$

где σ - среднеквадратичное отклонение.

Чтобы определить состояния шины CAN и классифицировать её неисправности, сначала загрузим данные, полученные в ходе испытаний шины, описанных в параграфе выше. Число k будем выбирать из оптимального числа соседей.

Для каждого набора данных нужно будет вычислить евклидову метрику между примером запроса и примером из загруженных данных. Все данные будем хранить в коллекции, добавив порядковый номер.

Так же нужно будет провести сортировку коллекции от наименьшего до наибольшего, в порядке возрастания.

Для задачи регрессии будем возвращать среднее значение выбранных k меток, а для задачи классификации выберем наиболее встречающееся значение.

Логистическая регрессия прогнозирует только вероятность возникновения некоторого события сравнивая его с логистической кривой. Результатом работы такого алгоритма является бинарная переменная, которая показывает вероятность того, что задаваемое значение принадлежит к какому-то определённому классу.

Основной смысл логит-регрессии основывается на том, что все исходные значения разделяются некой границей на две, соответствующих классам, части. В случае двухмерного измерения – это линия, в случае трехмерного измерения это плоскость. Данная граница задаётся исходя из имеющихся исходных данных для обучения алгоритма.

Далее стоит рассмотреть один элемент из набора данных для обучения, и определить его местоположение. Результатов может три:

- Точка принадлежит классу «+»
- Точка принадлежит классу «-»
- Точка лежит на границе

С помощью функции отношения классов **OR**, можно преобразовать полученные значения в вероятность P.

При помощи метода линейного дискриминантного анализа (Linear Discriminant Analysis, LDA), выбирают проекцию пространства изображений на пространство признаков таким образом, чтобы минимизировать внутриклассовое и максимизировать межклассовое расстояние в пространстве признаков. В этих методах предполагается, что классы линейно разделимы.[3]

Алгоритм классификации и регрессии с помощью деревьев состоит, по сути, в самом построении дерева принятия решения. Дерево принятия решений — это дерево, в листьях которого стоят значения целевой функции, а в остальных узлах — условия перехода, определяющие по какому из ребер идти. Если для данного наблюдения условие истина, то осуществляется переход по левому ребру, если же ложь — по правому.[3]

Метод опорных векторов в своей основе имеет перевод исходных векторов в пространство более высокой размерности, и поиск разделяющей гиперплоскости с наибольшем зазоров в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости,

разделяющей классы. Разделяющей гиперплоскостью будет гиперплоскость, создающая наибольшее расстояние до двух параллельных гиперплоскостей. Алгоритм основан на допущении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.[4]

Опишем алгоритм работы всей программы блок схемой. Схема представлена на рисунке 15.

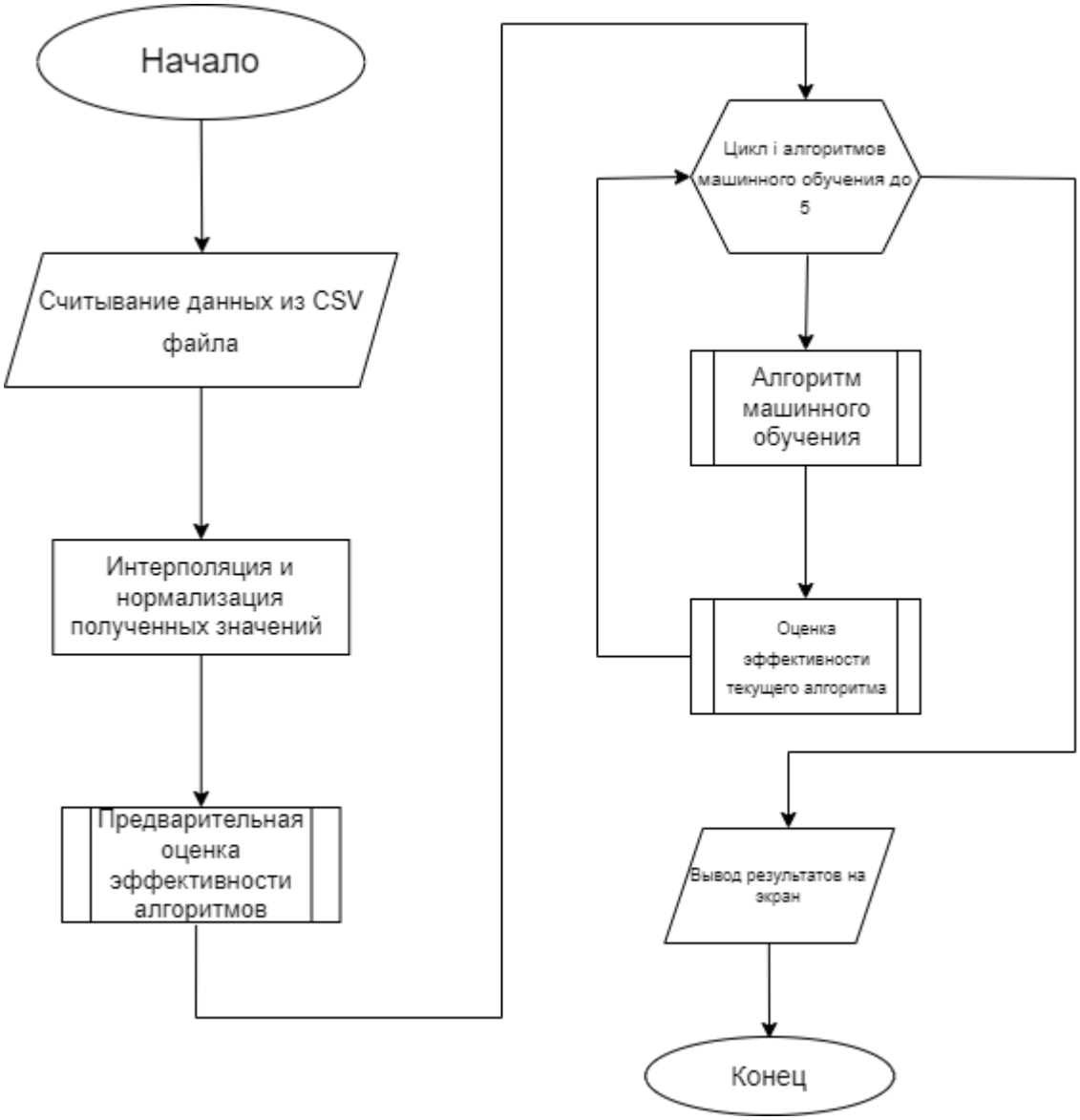

Рисунок 15 – Блок схема алгоритма программы

В начале алгоритма происходит считывания данный, полученных в результате эксперимента, из CSV файла. Далее производится подготовка этих данных для передачи в алгоритмы машинного обучения, а именно интерполяция и нормализация. Следующим шагом производится предварительная оценка эффективности алгоритмов машинного обучения, данная оценка производится методом кросс-валидации.

Подготовленные данные передаются каждому из описанных выше алгоритмов машинного обучения. Далее на основании экспериментальных данных производится предсказания аварийных реакций на шине. Полученные результаты предсказаний проходят оценку на эффективность по нескольким показателям.

Все результаты выводятся в консоль среды разработки.

На основании данной блок-схемы в последующих параграфах будет разработана программа на языке Python, для оценки аварийных реакция на шине CAN.

Вывод по разделу 2.

На данном этапе работы было произведено тестирование шины CAN на различные аварийные реакции. Получены зависимости токов от сопротивления утечки, построены графики этих зависимостей. Так же были выбраны типовые алгоритмы машинного обучения и выработана концепция разработки программного обеспечения, отраженная в блок схеме.

3 Разработка программы для диагностики шины CAN

3.1 Набор данных для обучения

Перед началом разработки программы для диагностики шины CAN, необходимо создать набор данных, который будет использоваться для машинного обучения. В параграфе 2.2 было описано тестирование шины CAN на различные аварийные реакции. Для удобства использования обучающего набора данных создадим файл в формате .CSV, в котором будут записаны токи линий High и Low, для каждого паразитного сопротивления. Формат CSV -текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделенных запятыми. В таблице будут присутствовать столбцы, в которых записываются токи линий High и Low драйвера, а также соответствующее этим токам паразитное сопротивление. В последнем столбце запишем фактическое состояние шины.

3.2 Подготовка набора данных для машинного обучения

В предыдущих параграфах мы описали данные, которые используются для обучения алгоритмов предсказания. Однако для оценки работы алгоритмов необходимо предоставить данные, которые отличаются от тех, что используются для обучения. Получить такие данные можно методом интерполяции уже имеющихся данных, полученных в результате тестирования шины CAN.

Интерполяция — в вычислительной математике нахождение неизвестных промежуточных значений некоторой функции, по имеющемуся дискретному набору её известных значений. [8]

Реализация таких математических операций как интерполяция в среде Python уже реализована в ряде библиотек. Импорт этих библиотек представлен на рисунке ниже.

```
from scipy.interpolate import Akima1DInterpolator, interp1d
```

Рисунок 16 - Код на языке Python с импортом библиотек для функций интерполяции

В среде Python для интерполяции кривой существует библиотека с открытым исходным кодом SciPy, включающий в себя функцию Akima1DInterpolator(), которая на вход принимает отсортированные значения функции.

После импорта библиотек нам необходимы тестовые данные для реализации интерполяции кривой. В этой работе будем использовать данные полученные в результате тестирования шины CAN, а именно токи утечки.

Рассмотрим случай, когда линия High замкнута на землю, при этом линия Low остаётся в нормальном состоянии. Используя стенд, описанный в параграфе 2, получим значения токов при каждом паразитном сопротивлении. Для удобства использования этих данных в дальнейшем запишем их в CSV файл.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot
```

```
resistance_arr = [ 0, 100, 1000, 10000, 100000, 1100, 11100, 111100 ]

show_graph = True
csv_fnm = '/home/svm/QT_PROJ/build-ESP32_WIFI_2_CAN-Desktop-Debug/CAN_diag/CAN_H_2_gnd_data.csv'
#csv_fnm = '/home/svm/QT_PROJ/build-ESP32_WIFI_2_CAN-Desktop-Debug/CAN_diag/CAN_L_2_gnd_data.csv'

file_name = csv_fnm.split('/')[-1].split('.')[0]
```

```
data = pd.read_csv(csv_fnm, sep='\t')
#data = pd.read_csv(csv_fnm, sep=',')

print( f' ===== type(data): {type(data)},\tdata.info:\n{data.info()}')
print( f' ===== data.shape: {data.shape} ')

curr_high = data[data.columns[0]]
curr_low = data[data.columns[1]]

if show_graph:
    pyplot.title(f'{file_name} test results')
    pyplot.plot(data.index, curr_low, 'g-', label='Curr_Low')
    pyplot.plot(data.index, curr_high, 'r-', label='Curr_High')
    pyplot.legend()
    pyplot.show()

data.columns
```

Рисунок 17 – Код на языке Python с реализацией считывания данных из CSV файла

Считанные результаты выведем на график.

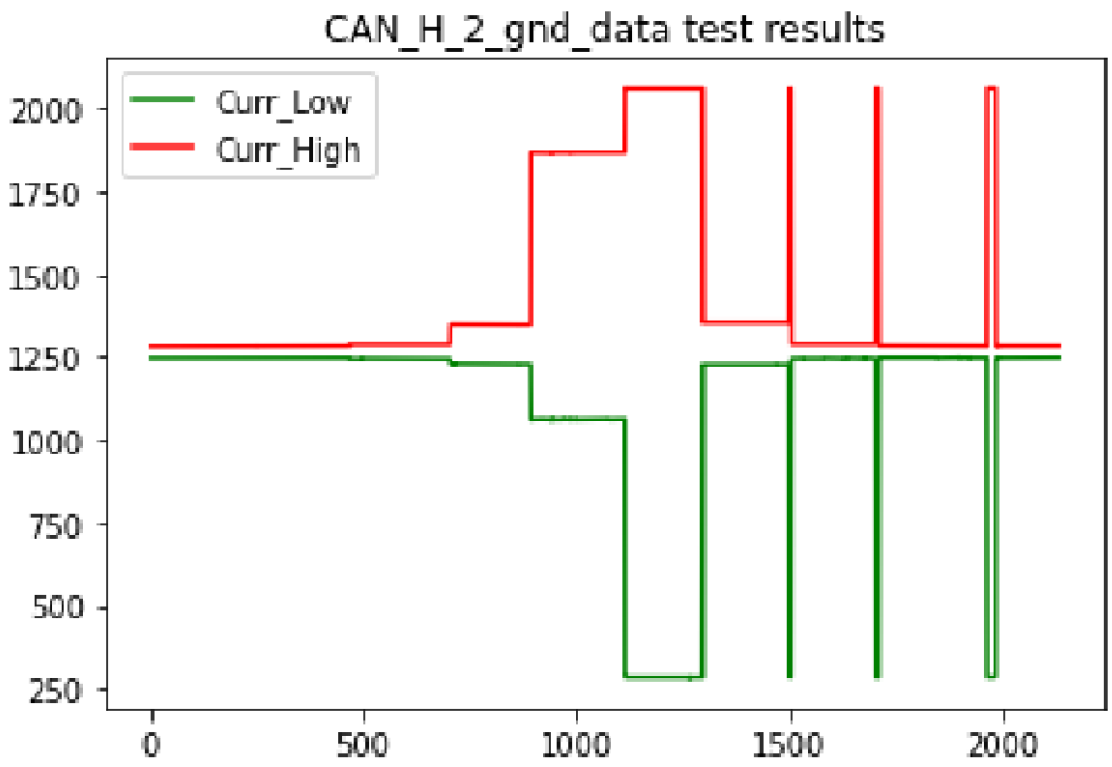

Рисунок 18 – График токов линий High и Low при замкнутой линии High на землю

Выделим ключевые точки соответствия замкнутому сопротивлению и тока драйвера и составим таблицы. Код на языке Python представлен ниже.

```
sorted(kmeans_Lo.cluster_centers_)
```

```
[array([283.20093458]),
 array([1061.51801802]),
 array([1224.94152047]),
 array([1226.47058824]),
 array([1243.88399072]),
 array([1245.]),
 array([1246.]),
 array([1247.02])]
```

```
def prn_calib_tbl(tbl):
    for r, c in tbl:
        print(f'{r}:\t{round(c[0],2)}')

calib_table_hi = list(zip( sorted(resistance_arr), sorted(kmeans_Hi.cluster_centers_,reverse=True)))
calib_table_lo = list(zip( sorted(resistance_arr), sorted(kmeans_Lo.cluster_centers_)))
print( f'{file_name}\n')

print( f'=== calib_table_hi, len: {len(calib_table_hi)}\n')
prn_calib_tbl(calib_table_hi)
print( f'=== calib_table_lo, len: {len(calib_table_lo)}\n')
prn_calib_tbl(calib_table_lo)
```

Рисунок 19 – Код на языке Python с созданием таблиц токов

Далее полученные табличные данные интерполируем, чтобы получить количество точек достаточное для обучения алгоритма предсказания. Код с реализацией интерполяции представлен ниже.

```
c_hi = np.array( sorted(kmeans_Hi.cluster_centers_,reverse=True) ).reshape(1,8)[0]
c_lo = np.array( sorted(kmeans_Lo.cluster_centers_)).reshape(1,8)[0]

x = np.arange(10, 100000, 1)

res_a = sorted(resistance_arr)
y_for_hi = Akima1DInterpolator( sorted(resistance_arr), c_hi )(x)
y_for_lo = Akima1DInterpolator( sorted(resistance_arr), c_lo )(x)

# y_for_hi = interp1d( res_a, c_hi )(x)
# y_for_lo = interp1d( res_a, c_lo )(x)

pyplot.title(f' Driver current for case {file_name} ')
pyplot.plot(resistance_arr, c_hi,'ro')
pyplot.plot(resistance_arr, c_lo,'go')
pyplot.plot(x, y_for_hi, 'r-', label='hi_cur')
pyplot.plot(x, y_for_lo, 'g-', label='lo_cur')
pyplot.xscale('log')
pyplot.legend()
pyplot.show()
```

Рисунок 20 – Код на языке Python с реализованной интерполяцией

Данные операции были проделаны для двух аварийных реакций, замыкания каждой из линий на землю. Интерполированные графики зависимости токов от сопротивлений утечки представлены ниже.

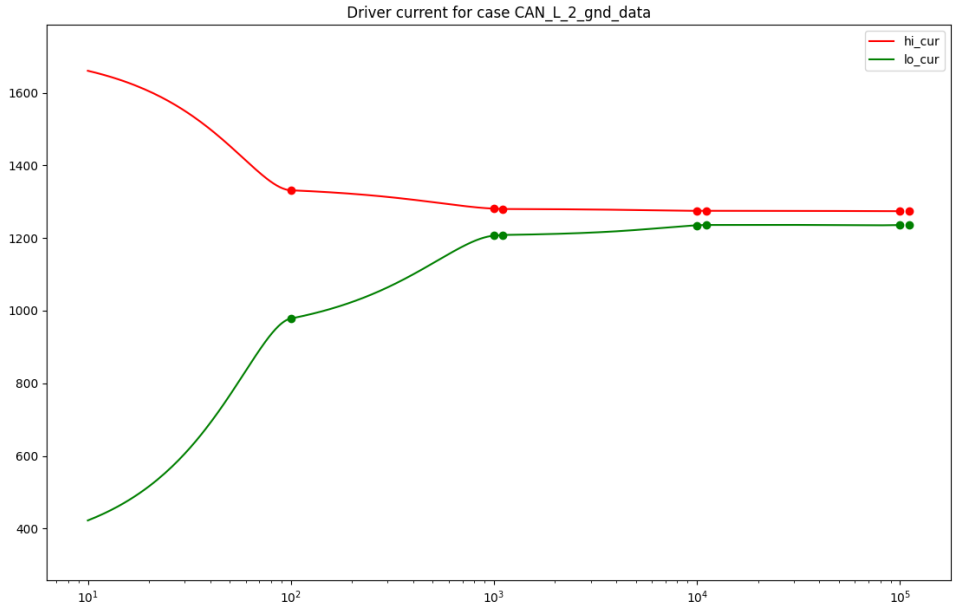

Рисунок 21 – Интерполированный график токов утечки CAN драйвера при замкнутой линии Low на землю

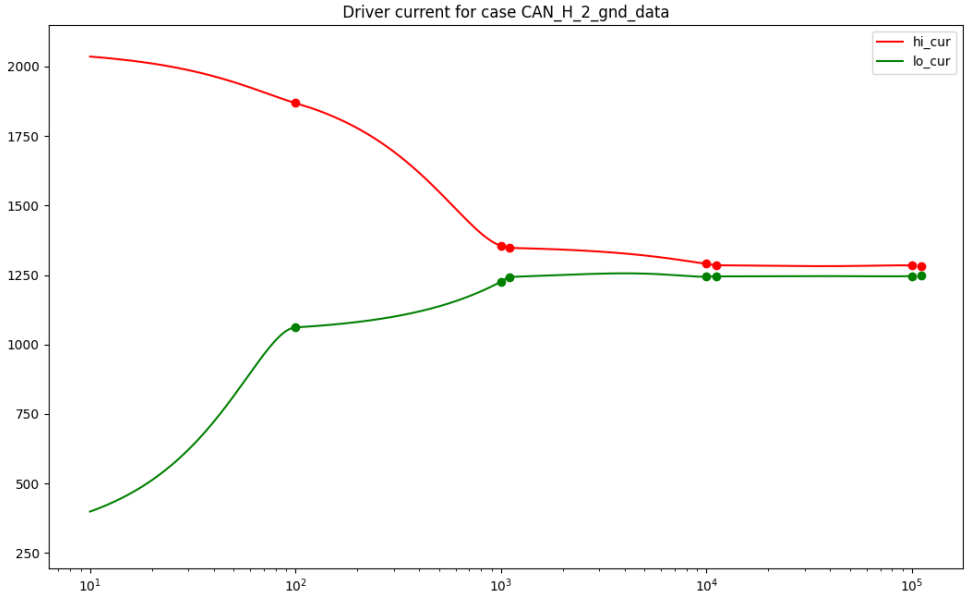

Рисунок 22 - Интерполированный график токов утечки CAN драйвера при замкнутой линии High на землю

Далее сформируем CSV файл, данные из которого будут использованы для машинного обучения. Для формирования CSV файла будем использовать библиотеку pandas. В результате получим файл со структурой, описанной на рисунке ниже.

	curr_hi	curr_lo	CAN_H_2_gnd_res	CAN_L_2_gnd_res	CAN_BUS_diag
99985	1284.507173	1245.999571	99995.0	100000.0	OK
99986	1284.507004	1245.999656	99996.0	100000.0	OK
99987	1284.506835	1245.999742	99997.0	100000.0	OK
99988	1284.506667	1245.999828	99998.0	100000.0	OK
99989	1284.506498	1245.999914	99999.0	100000.0	OK
99990	1720.884954	422.101634	100000.0	10.0	CAN_L_2_gnd
99991	1721.037982	431.472011	100000.0	11.0	CAN_L_2_gnd
99992	1721.186834	440.798290	100000.0	12.0	CAN_L_2_gnd
99993	1721.331512	450.079529	100000.0	13.0	CAN_L_2_gnd

Рисунок 23 – Структура CSV файла с данными для машинного обучения

В данной файле первый столбец – номер по порядку. Во втором столбце описана пара токов утечки с линий High и Low. В третьем и четвёртом столбце указано сопротивление утечки. В последнем тип аварийной реакции. В данном файле для апробации алгоритма машинного обучения описано только две аварийные реакции. В дальнейшем возможно дополнять данную программу данными и исследовать другие аварийные реакции.

3.3 Разработка программы на языке Python

В данной работе принято решение разрабатывать программу на языке Python, так как в данной среде существует множество библиотек с различными алгоритмами машинного обучения. Для работы будем использовать среду разработки PyCharm версии 2022.1. Создадим в данной среде разработки пустой проект.

Чтобы работать с библиотеками машинного обучения, нужно заранее их загрузить в среду, а также в коде программы с помощью команды import имплементировать их в проект. Все включенные в проект библиотеки изображены на рисунке 16.

```python
import numpy as np
import pandas as pd

from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

```

Рисунок 24 – Код на языке Python с импортом библиотек.

В данный проект включена такая библиотека как numpy, которая предоставляет поддержку математических функций для работы с массивами. Так же в проекте присутствует библиотека pandas, предназначенная для работы с числовыми таблицами и временными рядами.

Основной библиотекой для машинного обучения является sklearn. Данная библиотека включает в себя различные алгоритмы машинного обучения: классификации, регрессии, кластеризации и др. Метод k-ближайших соседей так же присутствует в данной библиотеке и импортирован в проект командой import KNeighboursClassifier. Sklearn полностью написана на языке Python, однако использует некоторые методы и процедуры из библиотеки numpy, по этому импортироваться в проект должна только после импорта numpy.

Так же в проект импортированы все библиотеки, которые содержат в себе реализацию методов машинного описанных в параграфе 2.3.

Библиотека matplotlib нужна для визуализации данных, а именно вывода графиков на экран.

Библиотека seaborn выполняет все необходимые преобразования для создания более информативных графиков. Данная библиотека основывается на библиотеке matplotlib.

Поскольку в данной работе было принято решение сформировать исходные для обучения данные в формате CSV, необходимо считать файл с данными и десереализировать его в массив данных. Для этого в библиотеке pandas присутствует метод read_csv, который принимает путь к файлу как входной параметр, с типом данных string. Для того, чтобы проверить, верно ли считан файл, после десереализации выведем на экран 10 строк. Вывод строк осуществляется с помощью метода head, который принимает число формата int, как входной параметр. Данное число указывает, сколько строк необходимо вывести на экран или в консоль. Код на языке Python изображен на рисунке ниже.

```
20    enter_data = read_csv('Enter.csv', delimiter=';')
```

Рисунок 25 – Код на языке Python с реализацией считывания CSV файла.

Вторым параметром зададим разделитель в виде точки с запятой, так в данной работе используется файл CSV в формате с разделителем “;”.

Результат считывания CSV файла приведён на рисунке ниже.

	curr_hi	curr_lo	CAN_H_2_gnd_res	CAN_L_2_gnd_res	CAN_BUS_diag
0	2035.576215	398.514051	10.0	100000.0	CAN_H_2_gnd
1	2033.063364	409.943423	11.0	100000.0	CAN_H_2_gnd
2	2030.559858	421.345529	12.0	100000.0	CAN_H_2_gnd
3	2028.065885	432.718187	13.0	100000.0	CAN_H_2_gnd
4	2025.581632	444.059213	14.0	100000.0	CAN_H_2_gnd
5	2023.107286	455.366424	15.0	100000.0	CAN_H_2_gnd
6	2020.643034	466.637639	16.0	100000.0	CAN_H_2_gnd
7	2018.189063	477.870673	17.0	100000.0	CAN_H_2_gnd
8	2015.745559	489.063345	18.0	100000.0	CAN_H_2_gnd
9	2013.312712	500.213470	19.0	100000.0	CAN_H_2_gnd
10	2010.890706	511.318868	20.0	100000.0	CAN_H_2_gnd
11	2008.479730	522.377354	21.0	100000.0	CAN_H_2_gnd
12	2006.079971	533.386745	22.0	100000.0	CAN_H_2_gnd
13	2003.691615	544.344860	23.0	100000.0	CAN_H_2_gnd
14	2001.314850	555.249514	24.0	100000.0	CAN_H_2_gnd
15	1998.949862	566.098526	25.0	100000.0	CAN_H_2_gnd
16	1996.596840	576.889713	26.0	100000.0	CAN_H_2_gnd
17	1994.255970	587.620891	27.0	100000.0	CAN_H_2_gnd
18	1991.927439	598.289877	28.0	100000.0	CAN_H_2_gnd
19	1989.611434	608.894490	29.0	100000.0	CAN_H_2_gnd

Рисунок 26 – Входные данные для машинного обучения, считанные из CSV Файла (первый 20 строк)

На данном этапе работы неизвестно какой из алгоритмов будет наиболее эффективным и точным. Поэтому протестируем все выбранные алгоритмы на эффективность.

В среде PyCharm будем загружать каждый алгоритм и в цикле проверять эффективность. Результаты выведем в консоль. Код на языке Python приведён на рисунке ниже.

```
# Выбор столбцов для обучения
X = array[:, 2:6]

# Выбор столбца с результатом
y = array[:, 6]

# Разделение X и y на обучающую и контрольную выборки
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=0.25, random_state=1)
```

Рисунок 27 – Код на языке Python с реализацией разделения столбцов

После считывания данных из CSV файла необходимо разбить данные по массивам. Создадим два массива X и Y. В массив X считаем все столбцы с токами для обеих линий High и Low, а также сопротивления утечки. В массив Y считаем состояния шины при этих токах утечки. Далее проведём разделение на обучающую выборку и контрольную с помощью метода train_test_split. Контрольная выборка нужна для проверки результатов работы машинного обучения, однако для полноценного тестирования мы будем задавать свои данные, полученные в ходе испытания шины CAN.

Далее загрузим алгоритмы машинного обучения. Создадим коллекцию models в которую с помощью метода append будем добавлять выбранные алгоритмы машинного обучения. Код на языке Python представлен на рисунке ниже.

```
# Загружаем алгоритмы модели
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('SVM', SVC(gamma='auto')))

```

Рисунок 28 - Код на языке Python с добавлением алгоритмов машинного обучения в коллекцию models

Для проверки эффективности алгоритмов, будем использовать перекрёстную проверку оценки производительности.

Перекрестная проверка (кросс-валидация или скользящий контроль) — это статистический метод, используемый для оценки модели машинного обучения на независимых данных.[5]

Данная проверка имеет один параметр kfold, который показывает число групп, на которые должна быть разделена выборка данных.

В число kfold будем возвращать метод StatifieldkFold, одним из параметров которого является число групп (в нашем случае 4).

В коллекцию result будем возвращать текущие результаты, а в коллекцию name текущее имя метода машинного обучения. Результаты выведем в консоль. Код на языке Python и результаты проверки на эффективность представлены на рисунках ниже.

```
52  for name, model in models:
53      kfold = StratifiedKFold(n_splits=4, random_state=1, shuffle=True)
54      cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy')
55      results.append(cv_results)
56      names.append(name)
57      print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))
```

Рисунок 29 – Код на языке Python с алгоритмом проверки эффективности методов машинного обучения

```
LR: 0.937500 (0.108253)
LDA: 0.791667 (0.125000)
KNN: 0.416667 (0.176777)
CART: 0.875000 (0.125000)
SVM: 0.875000 (0.125000)
```

Рисунок 30 – Результат проверки методов машинного обучения

Из всех выбранных методов, самым эффективным оказался метод логистической регрессии, его эффективность равна 93,75%. В дальнейшем будем использовать его для диагностирования шины CAN на аварийные реакции. Для наглядности построим график результатов оценки алгоритмов. Одним из способов сравнения результатов является диаграмма, на которой отражен размах атрибутов выходных данных и «усов» для каждого сравнения распределений и самого распределения.

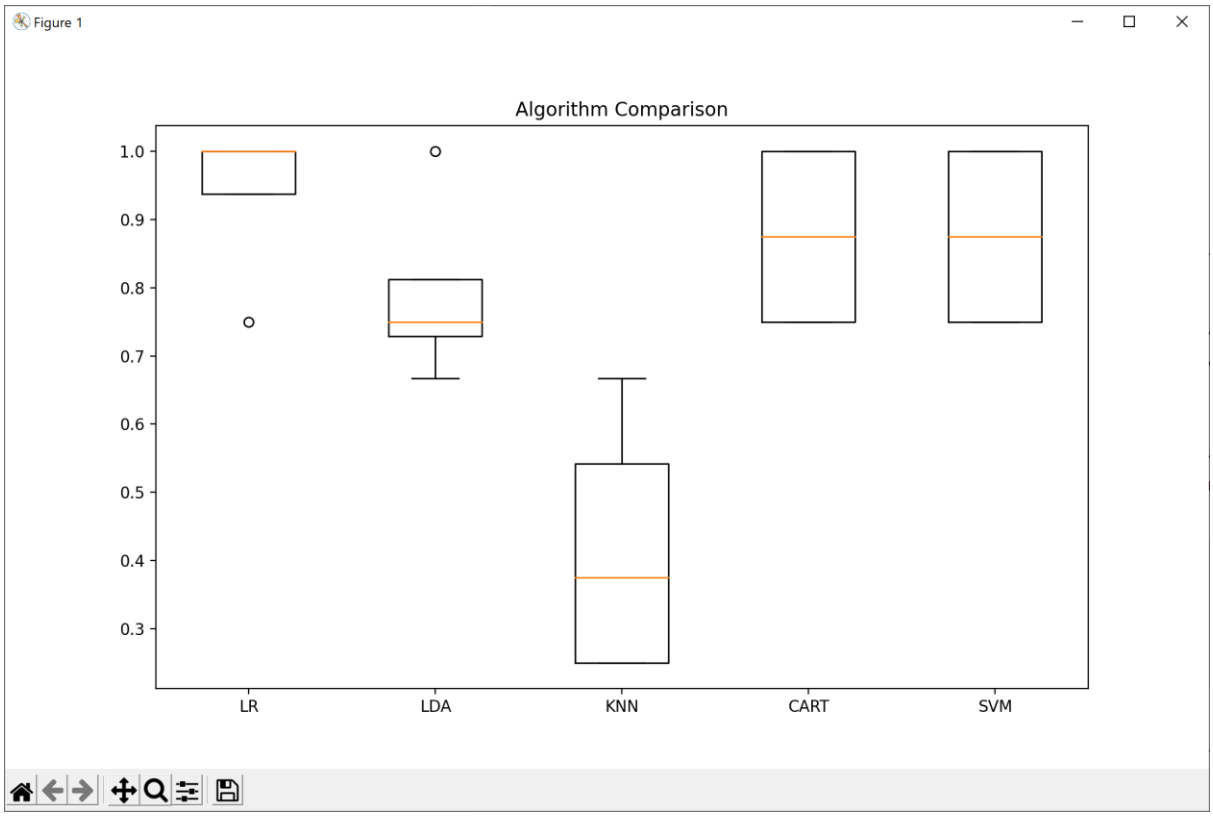

Рисунок 31 – Графики эффективности алгоритмов

На графике выше видно, что часть алгоритмов достигают эффективности 100%, однако некоторые алгоритмы имеют низкую эффективность близкую к 20%.

Для реализации метода логистической регрессии была подключена библиотека LogisticRegression из базы sklearn. В переменную model вернём значение метода LogisticRegression, который вызовем с параметрами solver = liblinear (т.к. он работает быстрее с небольшими наборами данных), а также с параметром multi_class = ‘ovr’. Код на языке Python представлен на рисунке ниже.

```
59  X_test =[[1250, 1250,   1280,   1550,   1900,   2000,   1250,   1250,   1250,   1099,   98
60  Y_test = [0, 2]
61  model = LogisticRegression(solver='liblinear', multi_class='ovr')
62  model.fit(X_train, Y_train)
63  predicted_y = model.predict(X_test)
64  print(predicted_y)
65  print('Accuracy: {:.2f}'.format(model.score(X_test, Y_test)))
```

Рисунок 32 – Код на языке Python с реализацией метода логистической регрессии

С помощью метода fit передадим в алгоритм данные для обучения, которые были сформированы раннее (Массивы X_train и Y_train). Для прогнозирования данных будем использовать метод predict. Данный метод принимает на вход параметры, исходя из которых надо предсказать результат.

Данные для прогнозирования были так же сняты со стенда, который описан в параграфах выше. Было промоделировано несколько аварийных реакций и сняты различные токи утечки драйвера. Данные представлены в таблице на рисунке ниже.

Создадим заранее массив X_test, в который запишем токи утечки для линий High и Low (рисунок 33). Выведем в консоль результат, рисунок 34. Так же оценим эффективность данного прогнозирования, записав ожидаемый результат в массив Y_test.

A	B
Curr_Hi	Curr_Low
1700	470
1350	1000
1300	1100
1280	1150
1220	1200
1220	1250
1220	1250
1220	1250
2100	360
1900	1100
1280	1200
1270	1240
1250	1250
1250	1250
1230	1250
1220	1250

Рисунок 33 – Таблица с данными для прогнозирования

```
LR
['CAN_L_2_gnd' 'CAN_L_2_gnd' 'CAN_L_2_gnd' 'CAN_L_2_gnd' 'OK' 'OK' 'OK'
 'OK' 'CAN_H_2_gnd' 'CAN_H_2_gnd' 'CAN_H_2_gnd' 'CAN_H_2_gnd' 'OK' 'OK'
 'OK' 'OK']
Accuracy: 1.00
1.0
[[4 0 0]
 [0 4 0]
 [0 0 8]]
              precision    recall  f1-score   support

 CAN_H_2_gnd       1.00      1.00      1.00         4
 CAN_L_2_gnd       1.00      1.00      1.00         4
          OK       1.00      1.00      1.00         8

    accuracy                           1.00        16
   macro avg       1.00      1.00      1.00        16
weighted avg       1.00      1.00      1.00        16
```

Рисунок 34 – Результат работы алгоритма машинного обучения

Как видно из рисунка 34, алгоритм сработал со 100% точностью.

Полный код программы на языке Python представлен в приложении 1.

Выводы по разделу 3.

На основе данных, полученных после тестирования, была разработана программа на языке Python. Так же полученные экспериментальные данные были модифицированы (интерполированы), для использования их в алгоритмах машинного обучения. Выбранные алгоритмы машинного обучения были предварительно проверены на эффективность и выбран наилучший.

4 Тестирование алгоритма и сравнительный анализ

4.1 Тестирование машинного обучения

В предыдущем параграфе оценка прогноза была только исходя из предполагаемых результатов. Однако, в среде Python есть возможность более тщательно оценить работу алгоритма. Для этого будем использовать библиотеку metrics из базы sklearn. В ней присутствуют методы, которые могут так же вычислить эффективность исходя из предполагаемых результатов, а затем вычислить точность классификации, а также матрицу ошибок и отчет о классификации.

Для этого вызовем последовательно методы accuracy_score, confusion_matrix и classification_report. В данные методы будем передавать данные, полученные в результате машинного обучения и матрицу предполагаемых результатов. Код на языке Python представлен ан рисунке ниже.

```
71  print(accuracy_score(Y_test, predicted_y))
72  print(confusion_matrix(Y_test, predicted_y))
73  print(classification_report(Y_test, predicted_y))
```

Рисунок 35 – Код на языке Python с реализацией тестирования результатов прогнозирования

Будем использовать данные для машинного обучения, полученные в предыдущем параграфе, методом интерполяции. Данные для предсказания остаются те же, что использованы ранее.

Результат работы тестовых методов приведён на рисунке ниже.

```
['CAN_L_2_gnd' 'CAN_L_2_gnd' 'CAN_L_2_gnd' 'CAN_L_2_gnd' 'OK' 'OK' 'OK'
 'OK' 'CAN_H_2_gnd' 'CAN_H_2_gnd' 'CAN_H_2_gnd' 'CAN_H_2_gnd' 'OK' 'OK'
 'OK' 'OK']
Accuracy: 1.00
1.0
[[4 0 0]
 [0 4 0]
 [0 0 8]]
              precision    recall  f1-score   support

 CAN_H_2_gnd       1.00      1.00      1.00         4
 CAN_L_2_gnd       1.00      1.00      1.00         4
          OK       1.00      1.00      1.00         8

    accuracy                           1.00        16
   macro avg       1.00      1.00      1.00        16
weighted avg       1.00      1.00      1.00        16
```

Рисунок 36 – Результат тестирования прогноза

По результатам тестирования видно, что точность равна 100% на контрольной выборке. Матрица ошибок даёт представление о количестве ошибок (сумма диагональных значений) Наконец, отчет о классификации предусматривает разбивку каждого класса по точности (precision), полнота (recall), f1-оценка, показывающим отличные результаты (при этом контрольная выборка была небольшая).

4.2 Сравнительный анализ алгоритмов машинного обучения

Поскольку для поставленной в данной работе задачи был отобран один алгоритм машинного обучения (метод логистической регрессии), стоит проверить другие алгоритмы, которые по результатам перекрёстной проверки показали меньший результат эффективности. Используя те же

данные, реализуем оставшиеся четыре алгоритма машинного обучения и сравним их эффективность предметно.

Первым апробируем алгоритм k – ближайших соседей.

```
84  model = KNeighborsClassifier(n_neighbors = 3)
85  model.fit(X_train, Y_train)
86  predicted_y = model.predict(X_test)
87  print (predicted_y)
88  print('Accuracy: {:.2f}'.format(model.score(X_test, Y_test)))
89  print(accuracy_score(Y_test, predicted_y))
90  print(confusion_matrix(Y_test, predicted_y))
91  print(classification_report(Y_test, predicted_y))
```

Рисунок 37 – Код программы на языке Python, с реализации KNN метода машинного обучения

На рисунке 37 видно, что в переменную models_knn возвращен классификатор метода KNN. С помощью метода fit ему переданы данные для обучения. Результат предсказания и эффективности представлены на рисунке ниже.

```
['CAN_L_2_gnd' 'CAN_L_2_gnd' 'CAN_L_2_gnd' 'CAN_L_2_gnd' 'OK' 'OK' 'OK'
 'OK' 'CAN_H_2_gnd' 'CAN_H_2_gnd' 'CAN_H_2_gnd' 'CAN_H_2_gnd' 'OK' 'OK'
 'OK' 'OK']
Accuracy: 1.00
1.0
[[4 0 0]
 [0 4 0]
 [0 0 8]]
              precision    recall  f1-score   support

 CAN_H_2_gnd       1.00      1.00      1.00         4
 CAN_L_2_gnd       1.00      1.00      1.00         4
          OK       1.00      1.00      1.00         8

    accuracy                           1.00        16
   macro avg       1.00      1.00      1.00        16
weighted avg       1.00      1.00      1.00        16
```

Рисунок 38 – Результат работы метода KNN

Как видно из результатов, данный метод не уступает методу логистической регрессии, его эффективность равна 100%, так же данный метод не допустил ошибок.

Далее реализуем метод линейного дискриминантного анализа. Для этого модифицируем код программы согласно рисунку 39.

```
81    models_lda = LinearDiscriminantAnalysis()
82    models_lda.fit(X_train, Y_train)
83    predictions_lda = models_lda.predict(X_test)
84    print('Accuracy: {:.2f}'.format(models_lda.score(X_test, Y_test)))
85    print(accuracy_score(Y_test, predictions_lda))
86    print(confusion_matrix(Y_test, predictions_lda))
87    print(classification_report(Y_test, predictions_lda))
```

Рисунок 39 – Код программы на языке Python, с реализацией LDA метода

Модификация программы произведена аналогично предыдущему случаю, в models_lda передан классификатор метода, методом fit загружены данные для обучения, а методом predict получены результаты предсказания.

Результат работы метода LDA представлен ан рисунке ниже.

```
LDA
['CAN_L_2_gnd' 'CAN_L_2_gnd' 'CAN_L_2_gnd' 'CAN_L_2_gnd' 'OK' 'OK' 'OK'
 'OK' 'CAN_L_2_gnd' 'CAN_H_2_gnd' 'OK' 'OK' 'OK' 'OK' 'OK' 'OK']
Accuracy: 0.81
0.8125
[[1 1 2]
 [0 4 0]
 [0 0 8]]
              precision    recall  f1-score   support

 CAN_H_2_gnd       1.00      0.25      0.40         4
 CAN_L_2_gnd       0.80      1.00      0.89         4
          OK       0.80      1.00      0.89         8

    accuracy                           0.81        16
   macro avg       0.87      0.75      0.73        16
weighted avg       0.85      0.81      0.77        16
```

Рисунок 40 – Результаты работы метода LDA

Как видно из результатов, метод LDA имеет 81% эффективности и допустил 3 ошибки.

Проверим эффективность метода классификации и регрессии с помощью деревьев. Модификация программы произведена аналогично предыдущему случаю, в models_CART передан классификатор метода, методом fit загружены данные для обучения, а методом predict получены результаты предсказания. Код на языке Python представлен на рисунке ниже.

```
models_CART = DecisionTreeClassifier()
models_CART.fit(X_train, Y_train)
predictions_CART = models_CART.predict(X_test)
print(predictions_CART)
print('Accuracy: {:.2f}'.format(models_CART.score(X_test, Y_test)))
print(accuracy_score(Y_test, predictions_CART))
print(confusion_matrix(Y_test, predictions_CART))
print(classification_report(Y_test, predictions_CART))
```

Рисунок 41 - Код программы на языке Python, с реализацией CART метода

Результат работы метода CART представлен на рисунке ниже.

```
CART
['CAN_L_2_gnd' 'CAN_L_2_gnd' 'CAN_L_2_gnd' 'CAN_L_2_gnd' 'CAN_L_2_gnd'
 'OK' 'OK' 'OK' 'CAN_H_2_gnd' 'CAN_H_2_gnd' 'OK' 'OK' 'OK' 'OK' 'OK' 'OK']
Accuracy: 0.81
0.8125
[[2 0 2]
 [0 4 0]
 [0 1 7]]
              precision    recall  f1-score   support

 CAN_H_2_gnd       1.00      0.50      0.67         4
 CAN_L_2_gnd       0.80      1.00      0.89         4
          OK       0.78      0.88      0.82         8

    accuracy                           0.81        16
   macro avg       0.86      0.79      0.79        16
weighted avg       0.84      0.81      0.80        16
```

Рисунок 42 – Результат работы метода CART

Исходя из полученных результатов, данный метод имеет эффективность 81,25%. Данный метод допустил 3 ошибки, аналогично предыдущему методу.

Последним методом испытаем метод опорных векторов. Модификация программы произведена аналогично предыдущему случаю, в models_svm передан классификатор метода, методом fit загружены данные для обучения, а методом predict получены результаты предсказания. Код на языке Python представлен на рисунке ниже.

```
models_svm = SVC(gamma='auto')
models_svm.fit(X_train, Y_train)
predictions_svm = models_svm.predict(X_test)
print(predictions_svm)
print('Accuracy: {:.2f}'.format(models_svm.score(X_test, Y_test)))
print(accuracy_score(Y_test, predictions_svm))
print(confusion_matrix(Y_test, predictions_svm))
print(classification_report(Y_test, predictions_svm))
```

Рисунок 43 - Код программы на языке Python, с реализацией SVM метода

Результат работы метода SVM представлен на рисунке ниже.

```
SVC
['CAN_L_2_gnd' 'CAN_L_2_gnd' 'CAN_L_2_gnd' 'CAN_L_2_gnd' 'CAN_L_2_gnd'
 'OK' 'OK' 'OK' 'CAN_H_2_gnd' 'CAN_H_2_gnd' 'CAN_L_2_gnd' 'OK' 'OK' 'OK'
 'OK' 'OK']
Accuracy: 0.81
0.8125
[[2 1 1]
 [0 4 0]
 [0 1 7]]
              precision    recall  f1-score   support

 CAN_H_2_gnd       1.00      0.50      0.67         4
 CAN_L_2_gnd       0.67      1.00      0.80         4
          OK       0.88      0.88      0.88         8

    accuracy                           0.81        16
   macro avg       0.85      0.79      0.78        16
weighted avg       0.85      0.81      0.80        16
```

Рисунок 44 – Результат работы метода SVM

Исходя из полученных результатов, данный метод имеет эффективность аналогичную методу CART. Данный метод допустил 3 ошибки, и имеет 81,25% эффективности.

Исходя из данных, полученных во время тестирования эффективности алгоритмов, составим таблицу, в которой укажем процент эффективности и количество допущенных ошибок предсказания, а также построим график.

Таблица 1 – Эффективность алгоритмов машинного обучения

	Эффективность, %	Количество ошибок предсказания
Метод k-ближайших соседей (KNN)	100	0
Логистическая регрессия или логит-модель (LR)	100	0
Линейный дискриминантный анализ (LDA)	81	3
Классификация и регрессия с помощью деревьев (CART)	81	3
Метод опорных векторов (SVM)	81	3

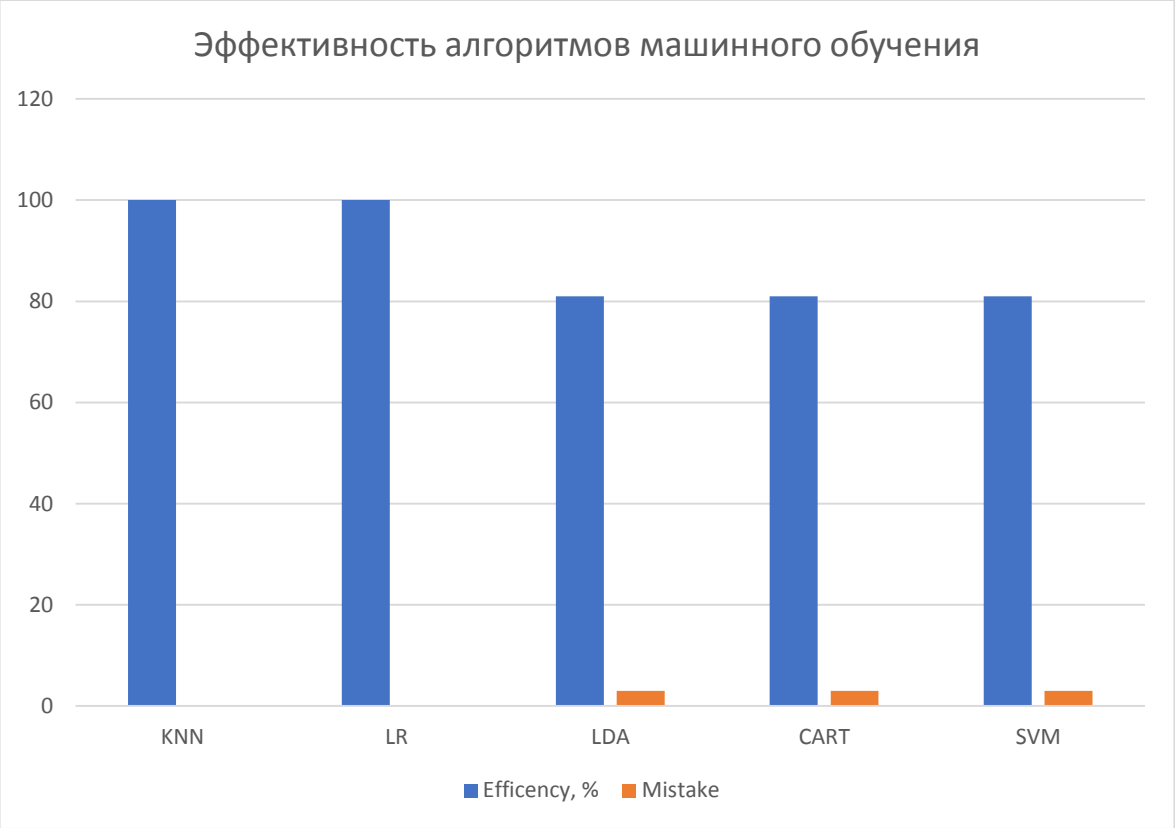

Рисунок 45 – График эффективности алгоритмов машинного обучения

Вывод по разделу 4.

В данном разделе выбранные алгоритмы машинного обучения были протестированы по различным параметрам. Исходя из результатов тестирования стало ясно, что выбранный алгоритм (логистическая регрессия) не единственный, показывающий высокую эффективность предсказания. Все данные сравнения были сведены в таблицу на основе которой построен график сравнения.

Заключение

В ходе выполнения выпускной квалификационной работы на тему «Разработка методики диагностики токов утечки CAN-шины по току потребления драйвера» были рассмотрены теоретические основы поиска подходов к обработке входных данных и способов построения модели для определения тока утечки или сопротивления утечки.

Во время выполнения данной работы было произведено тестирование шины CAN на различные аварийные реакции. Полученные данные были использованы как входные данные для машинного обучения.

Таким образом, были рассмотрены теоретические сведения по исследуемому объекту, рассмотрены и протестированы различные алгоритмы машинного обучения, выбран наиболее подходящий, а также разработано ПО для определения неисправностей на шине.

Для разработки программы с применением технологии машинного обучения был выбран язык Python и среда разработки PyCharm, которая имеет большой набор библиотек с реализацией алгоритмов машинного обучения. Этот выбор предоставляет возможность реализовать программу с наименьшими затратами.

Как показал анализ эффективности алгоритмов машинного обучения, наиболее эффективным являются алгоритм логистической регрессии. Однако после более глубокого анализа эффективности стало ясно, что существуют алгоритмы, которые, справляются с поставленной задачей с аналогичной эффективностью.

В ходе разработки была произведена предобработка данных методом к-средних, благодаря чему были найдены ключевые точки и по ним были построены кривые зависимости сопротивления утечки от входного тока. Далее по этим кривым были созданы данные для реакции модели -

результирующий столбец, благодаря которому были сопоставлены входной ток и реакция. Также была построена модель, которая по двум токам совершает прогноз о том, на какой из линий есть сопротивление утечки. После получения реакции по зависимости сопротивления утечки от входного тока было получено значение сопротивления утечки.

Таким образом, в данной выпускной квалификационной работе были отображены все основные моменты диагностики шины CAN, а также разработаны и программно реализованы алгоритмы определения неисправности с помощью методов машинного обучения.

Благодаря проведенным тестам была доказана высокая эффективность используемых алгоритмов для определения неисправностей на шине. И хотя в работе имеется всего две таблицы входных данных, разработанный подход позволяет расширить определение других подобных реакций.

Список используемых источников

1. B.Lubanovic. Introducing Python: Modern Computing in Simple Packages. O'Reilly Media, 2015. ISBN 978-1-449-35936-2.

2. J.VanderPlas. Python Data Science Handbook: Essential Tools for Working with Data. O'Reilly Media, 2016, 541 pp. ISBN 978-1-491-91205-8.

3. K.Funahashi. On the Approximate Realization of Continuous Mappings by Neural Networks. Neural Networks, 1989, v. 2, № 3, pp.183-191.

4. K.Hornick, M.Stinchcombe, H.White. Multilayer Feedforward Networks are Universal Approximators. Neural Networks, 1989, v. 2, № 5, pp.359-366.

5. M.Dawson. Python Programming for the Absolute Beginner. 3rd Edition. Course Technology, 2010. ISBN 978-1-435-45500-9.

6. M.Lutz. Python Pocket Reference: Python In Your Pocket. 5th Edition. O'Reilly Media, 2014. 258 pp. ISBN 978-1-449-35701-6.

7. Mohamad H.Hassoun. Fundamentals of Artificial Neural Networks. MIT Press, Cambridge, Massachusetts, 1995.

8. А.Н.Васильев, Д.А.Тархов. Нейростевое моделирование. Принципы. Алгоритмы. Приложения. СПб.: Изд-во Политехн. Ун-та, 2009. ISBN 978-5-7422-2272-9

9. Гэддис Т. Начинаем программировать на Python**.** – 4-е изд.: Пер. с англ. – СПб.: БХВ-Петербург, 2019. – 768 с.

10. Златопольский Д.М. Основы программирования на языке Python. – М.: ДМК Пресс, 2017. – 284 с.

11. Лутц М**.** Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с.

12. Лутц М. Программирование на Python, том I, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.

13. Лутц М. Программирование на Python, том II, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.

14. Лучано Рамальо Python. К вершинам мастерства. – М.: ДМК Пресс, 2016. – 768 с.

15. Любанович Билл Простой Python. Современный стиль программирования. – СПб.: Питер, 2016. – 480 с.: – (Серия «Бестсепперы O'Reilly»).

16. Рейтц К., Шлюссер Т. Автостопом по Python. – СПб.: Питер, 2017. – 336 с.: ил. – (Серия «Бестселлеры O'Reilly»).

17. Свейгарт, Эл. Автоматизация рутиных задач с помощью Python: практическое руководство для начинающих. Пер. с англ. — М.: Вильямс, 2016. – 592 с.

18. Федоров, Д. Ю. Программирование на языке высокого уровня Python: учебное пособие для прикладного бакалавриата / Д. Ю. Федоров. – 2-е изд., перераб. и доп. – Москва : Издательство Юрайт, 2019. – 161 с. – (Бакалавр. Прикладной курс). – ISBN 978-5-534-10971-9. – Текст: электронный // ЭБС Юрайт [сайт]. – URL: https://urait.ru/bcode/437489 (дата обращения: 13.02.2020).

19. Шелудько, В. М. Основы программирования на языке высокого уровня Python: учебное пособие / В. М. Шелудько. – Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2017. – 146 с. – ISBN 978-5-9275-2649-9. – Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. – URL: http://www.iprbookshop.ru/87461.html (дата обращения: 13.02.2020). – Режим доступа: для авторизир. Пользователей

20. Шелудько, *В. М.* Язык программирования высокого уровня Python. Функции, структуры данных, дополнительные модули: учебное пособие / В.

М. Шелудько. – Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2017. – 107 с. – ISBN 978-5-9275-2648-2. – Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. – URL: http://www.iprbookshop.ru/87530.html (дата обращения: 13.02.2020). – Режим доступа: для авторизир. Пользователей

Приложение А

Код программы на языке Python

```
from pandas import read_csv
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis


enter_data = read_csv('model_data_v2.csv', sep=',')
print(enter_data.shape)

print(enter_data.head(20))

# Разделение датасета на обучающую и контрольную выборки
array = enter_data.values


# Выбор столбцов для обучения
X = array[:, 2:6]

# Выбор столбца с результатом
y = array[:, 6]

# Разделение X и y на обучающую и контрольную выборки
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,
test_size=0.25, random_state=1)



# Загружаем алгоритмы модели
models = []
models.append(('LR', LogisticRegression(solver='liblinear',
multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))


# оцениваем модель на каждой итерации
results = []
names = []

for name, model in models:
   kfold = StratifiedKFold(n_splits=4, random_state=1, shuffle=True)
   cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
scoring='accuracy')
```

```
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

pyplot.boxplot(results, labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()


Test_data = read_csv('Enter2.csv', delimiter=';')
array = Test_data.values
Y_test = array[:,4]
X_test = array[:, 0:4]

print('LR')
model = LogisticRegression(solver='liblinear', multi_class='ovr')
model.fit(X_train, Y_train)
predicted_y = model.predict(X_test)
print (predicted_y)
print('Accuracy: {:.2f}'.format(model.score(X_test, Y_test)))
print(accuracy_score(Y_test, predicted_y))
print(confusion_matrix(Y_test, predicted_y))
print(classification_report(Y_test, predicted_y))

print('KNN')
model = KNeighborsClassifier(n_neighbors = 3)
model.fit(X_train, Y_train)
predicted_y = model.predict(X_test)
print (predicted_y)
print('Accuracy: {:.2f}'.format(model.score(X_test, Y_test)))
print(accuracy_score(Y_test, predicted_y))
print(confusion_matrix(Y_test, predicted_y))
print(classification_report(Y_test, predicted_y))

print('LDA')
model = LinearDiscriminantAnalysis()
model.fit(X_train, Y_train)
predicted_y = model.predict(X_test)
print (predicted_y)
print('Accuracy: {:.2f}'.format(model.score(X_test, Y_test)))
print(accuracy_score(Y_test, predicted_y))
print(confusion_matrix(Y_test, predicted_y))
print(classification_report(Y_test, predicted_y))

print('CART')
model = DecisionTreeClassifier()
model.fit(X_train, Y_train)
predicted_y = model.predict(X_test)
print (predicted_y)
print('Accuracy: {:.2f}'.format(model.score(X_test, Y_test)))
print(accuracy_score(Y_test, predicted_y))
print(confusion_matrix(Y_test, predicted_y))
print(classification_report(Y_test, predicted_y))
```