UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Android application for anomaly detection in car engine system

Author:
Victor YEZHOV

Supervisor:
Oles DOBOSEVCH

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2020

Declaration of Authorship

I, Victor YEZHOV, declare that this thesis titled, "Android application for anomaly detection in car engine system" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"Quality means doing it right when no one is looking."

Henry Ford

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Android application for anomaly detection in car engine system

by Victor YEZHOV

Abstract

Nowadays smart phones are playing major role in our everyday life. The computational power of flagman versions can be compared with some of the laptops. Having all that power in our pockets gives us an opportunity to create apps with capabilities to analyze large amount of data, perform computations for making predictions and in the end, increase life comfort of the user.In this work i would like to describe the process of creating Android application, which aims to detect anomalies in car engine system.

Acknowledgements

I would first like to thank my diploma supervisor Oles Dobosevych of the Faculty of Applied Sciences at Ukrainian Catholic University. Oles always helped me with an project ideas and was very supportive during studying at Ukrainian Catholic University

Also, I am thankful to the Ukrainian Catholic University for giving me all necessary knowledge for becoming a professional in Computer Science field ...

Contents

Declaration of Authorship i

Abstract iii

Acknowledgements iv

List of Figures vii

1 Introduction 1
- 1.1 Overview . . . 1
- 1.2 Problem . . . 1
- 1.3 Suggested solution . . . 1

2 Market analysis 3

3 Technical information 5
- 3.1 OBD . . . 5
  - 3.1.1 Overview . . . 5
  - 3.1.2 Available data . . . 5
  - 3.1.3 Request structure . . . 6
  - 3.1.4 Work modes . . . 7
  - 3.1.5 Adapter . . . 7
  - 3.1.6 Mobile . . . 7

4 Methods 9
- 4.1 Overview . . . 9
- 4.2 Multivariate statistical analysis . . . 10
  - PCA . . . 10
  - Anomaly detection . . . 10
  - Mahalanobis distance . . . 11
- 4.3 Artificial Neural Network . . . 11
  - 4.3.1 General . . . 11
  - 4.3.2 Autoencoder NN . . . 12
  - 4.3.3 Results . . . 13

5 Implementation 15
- 5.1 Technical side . . . 15
- 5.2 Internal architecture . . . 16
  - 5.2.1 General Technologies . . . 16
  - 5.2.2 App architecture . . . 16
  - 5.2.3 Database . . . 19
  - External . . . 19
  - Internal . . . 20
- 5.3 Design Prototypes . . . 20

5.3.1 Car screen . . . 21
5.3.2 Engine state screen . . . 22
5.3.3 Connections screen . . . 23
5.4 App versions . . . 23
5.4.1 POC . . . 23
5.4.2 MVP . . . 24
5.4.3 V0.1 . . . 24
5.4.4 V0.5 . . . 24
5.4.5 V.** Future Plans . . . 24

6 Results **26**
6.1 General . . . 26
6.2 Improvements to be made . . . 26
6.3 Weak points . . . 27
6.4 Summary . . . 27

Bibliography **28**

List of Figures

2.1 MotorData . . . 3
2.2 Torque Pro . . . 3

3.1 Common place for OBD-2 socket . . . 5
3.2 OBD-2 Request / Response structure . . . 6
3.3 OBD-2 possible working modes . . . 7
3.4 App energy efficiency . . . 8

4.1 Outlier visualization . . . 9
4.2 PCA example . . . 10
4.3 ANN schema . . . 11
4.4 Autoencoder NN schema . . . 12
4.5 Loss function . . . 13
4.6 RPM dynamics . . . 13
4.7 Throttle dynamics . . . 13
4.8 Mehalanobis results . . . 14
4.9 Autoencoder results . . . 14

5.1 MVVM . . . 17
5.2 Application architecture . . . 19
5.3 Document in Firebase . . . 19
5.4 Car info screen . . . 21
5.5 Engine state screen . . . 22
5.6 Connections screen . . . 23

List of Abbreviations

IVS	**I**nternal **C**ombustion **E**ngine
OBD	**O**n **B**oard **D**iagnostics
BLE	**B**leutooth **L**ow **E**nergy
ANN	**A**trificial **N**eural **N**etwork
DB	**D**ata **B**ase
POC	**P**roof **O**f **C**oncept
MVP	**M**inimum **V**aluable **P**roduct

Chapter 1

Introduction

1.1 Overview

In modern world personal car is becoming essential attribute of all people in developed countries. Together with it, cars itself have evolved from simple IVS with wheels to highly advanced vehicles with big variety of subsystems, sensors and abilities. On the other hand, having all this advanced technology inside an engine, makes is very hard to understand and check stability of work for people without special education. So, I come up with an idea of a "pocket mechanic" app. If car is able to give us in numeric form, all information about it, why just not to use computer to perform analysis over this data ? Because analyzing numbers is the thing which computers can do best, isn't it? What is more, all this big variety of sensorics data could also allow us to be more preventive in case of problem. Having analogy with human anatomy if our body would be armed with bunch of sensors which are continuously controlling all vital systems, we could have foreseen failures in our organism. Foreseeing issue in this case means a life of an individual, in the car case saving a lot of time money of the user. But How the program, or in our case, mobile app will know that something is wrong? Here comes the Anomaly Detection. Having all data stored in memory, we can clearly see the place and the time point when something went wrong. So, A.D.E.S project aim is to give people who are not very familiar with car engine construction a powerful tool, which can help in their everyday life by having your "pocket mechanic" which watches over your car.

1.2 Problem

As was described modern car users can no longer see problem by themselves. So, we can only react on failure when it's occurs. Which is not the most efficient way of problem solving. The easiest and economically effective way is to be able to prevent failures of the system.

1.3 Suggested solution

A.D.E.S system is a pairing of Android mobile app with software which allows analyze data, and special Bluetooth adapter for car which will be described in technical related parts of this work. Workflow of the app is the following:

- User connects adapter to car diagnostics socket
- For the first time app is in synchronization mode in which it is gathering and preparing vital data

– When initialization is done app is in working mode. While user drives it continually sends data to the user's phone. Data there is mapped to appropriate form for Anomaly Detection algorithm.

– If app mentions suspicious behavior of some systems it notifies the user about it, giving useful tips about what can cause the problem and what can be a solution.

Chapter 2

Market analysis

Cars apps are divided in two sections: OBD readers and Car mangers. First ones are used to read car data and display it to the user, second are just managing apps, which user can track his expanses, count days till service, etc. A.D.E.S will combine and improve this function bringing new level of service to the client. Currently on the market there are a few quite successful apps for interacting with car via phone and via Bluetooth. The best apps from this section have from 500.000 to 5.000.000 downloads. So, these are quite demanded tools in the car owner's world. But all of them has some elements in common. Old UI, not obvious UX and absentness of any innovations. On the other hand, still many people use it, because it brings a lot of additional, valuable information about their car. Here are some examples of apps this kind:

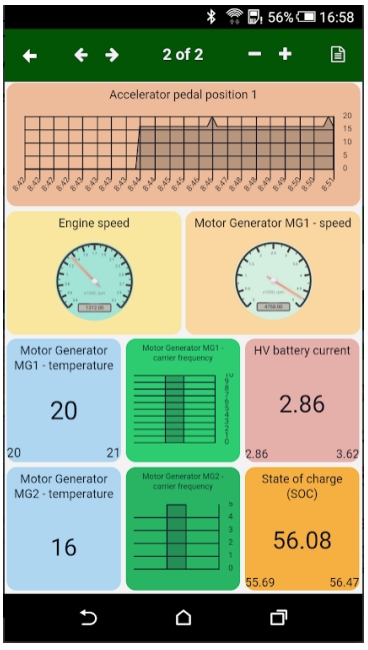

FIGURE 2.1: Motor-Data

FIGURE 2.2: Torque Pro

So here you can obviosly see that these apps are quite old fashined. But this is not main problem. Above I mentioned that these apps now, are missing some inovative part. Yes, they can read and display values that car system provides, but in modern world data is a key to something bigger, and as was mentioned in the first chapter – preventing failures – that is the feature that none of the current apps on the market has. And this is the feature which will be the core function of A.D.E.S system. Giving powerfull macine learning algorithms and neurall networks which

will be disscussed later in this article, A.D.E.S could become new state of the art solution in car industry with no real compatitors at the start.

Chapter 3

Technical information

3.1 OBD

3.1.1 Overview

OBD stands for on board diagnostics. This is term is referring to vehicles self-diagnostics and reporting capability. Since first introduced in early 1980s. This communication standard has dramatically improved and now is a mandatory option which car manufactures should include to their production. In this project I will be using OBD-2 because nowadays this is world-wide standard. OBD-2 provides a standardized hardware interface – female 16-pin J1962 connector. The big

FIGURE 3.1: Common place for OBD-2 socket

advantage of standardization is that car manufactures are obliged to place OBD-2 connector within 0.61m from steering wheel, so it's always accessible for driver.

3.1.2 Available data

OBD -2 can provide big amount of information about car status including:

- Engine load
- Engine coolant temperature
- Engine RPM
- Vehicle speed

- Mass air flow sensor air flow rate
- Throttle position
- Run time since engine start
- Engine oil temperature
- Fuel condition
- Fuel Pressure
- Intake manifold pressure
- Fuel Rail Pressure
- Fuel Tank Level Input
- Car fuel type

And many more other params. Full list could be found here.

3.1.3 Request structure

Communication is built using simple request – response structure. Each request is array of bytes divided by section

Each request contains:

- Header
  - Header information is sending to tell receiver what kind of information is to be transmitted, priority of data and whether or not a response from sender is expected
- Body
  - Contains data from censors
- Control sum
  - Helps to ensure that Body data is valid
- Other
  - SOF – starts of frame – marks start of message
  - EOD – marks end of data
  - EOF – end of frame – marks end of message etc

Overall structure looks like the following :

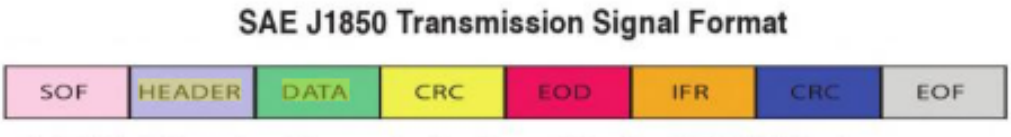

FIGURE 3.2: OBD-2 Request / Response structure

For example, request for getting current speed will look like the following:

68 6C F1 01 0D A6, where:
68 6C F1 – Header
01 0D – Body (01 – work mode 0D pid number which is responsible for current speed)
A6 – CRC

3.1.4 Work modes

OBD-2 can send data in 10 working modes.

Service (hex)	Description
01	Show current data
02	Show freeze frame data
03	Show stored Diagnostic Trouble Codes
04	Clear Diagnostic Trouble Codes and stored values
05	Test results, oxygen sensor monitoring (non CAN only)
06	Test results, other component/system monitoring (Test results, oxygen sensor monitoring for CAN only)
07	Show pending Diagnostic Trouble Codes (detected during current or last driving cycle)
08	Control operation of on-board component/system
09	Request vehicle information
0A	Permanent Diagnostic Trouble Codes (DTCs) (Cleared DTCs)

FIGURE 3.3: OBD-2 possible working modes

In my case I'm interested only in 01 and 02 work modes which are for sending live data and data in specific time frame correspondingly.

3.1.5 Adapter

Adapters for OBD-2 are based on ELM327 microcontroller

So, usage of this adapter is quite simple, it is sold in every car shop for a price around 10$. After purchasing user just needs to stick to female adapter in his car and forget that it is there. The sizing on new adapters is quite small so they will be not noticeable people inside of the vehicle Basically, consists of microcontroller itself and Bluetooth module. There are some models which are allowing communication via WIFI by I stopped my choice on Bluetooth one, because of its simplicity. Moreover, Bluetooth Low Energy API is far more effective that WIFI in terms of battery and phone resources consumption.

3.1.6 Mobile

Mobile platform to which application is aiming for Is Android. It gives full access for Bluetooth API, that's why it's perfect choice for an application of this kind.

Later in this article I will describe in detail app architecture, it's internal logic and functionalities.

For accessing Bluetooth I'm using RxAndroidBle. Powerful library based on RxJava, which is best java library for asynchronous apps for now. So, knowing that

all Bluetooth operations are asynchronous this library provides very high level of comfort while using it. It also does thread management which is very important in Android development because mobile device has limited resources and has specified treads for different actions.

On the part of actual communication with OBD connector things are more complicated. There is java library obd-java-api which provides some most common precoded requests like reading the RPM or engine temperature. But the other side of medal is that this software tool is quite outdated and has poor error handling mechanism that's why my usage of it was limited to these basic functions. More complex data requests required native query implementation and direct communication with device Overall efficiency on the app you can see on this graph:

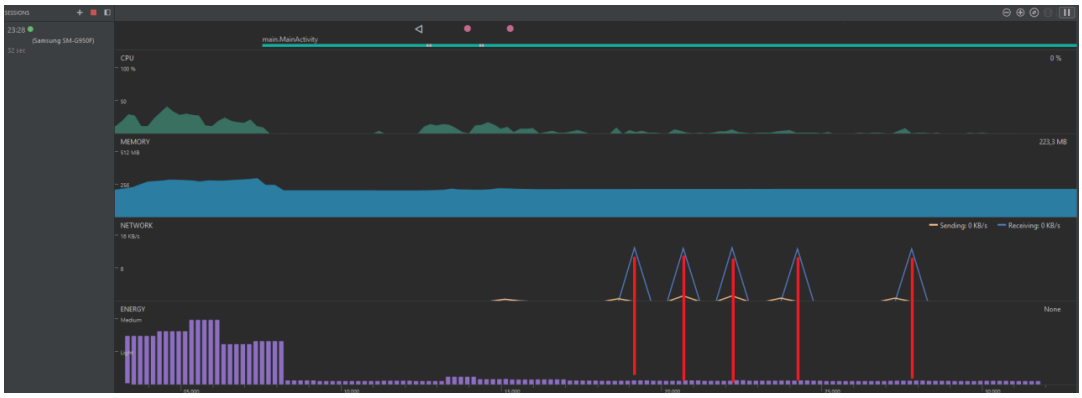

FIGURE 3.4: App energy efficiency

So, on start on the application when it is performing the search of devices the energy usage is high for a small period of time. Then it drops almost to stand – by power mode. With red lines I marked the moments of data reading, as you can see it slightly affects the CPU usage and courses very little change in the energy consumption. How app works in detail I will describe in dedicated article.

Chapter 4

Methods

4.1 Overview

Core of the application is the prediction module and anomaly detection. Anomaly detection can be explained as the identification of rare events or observations which significantly differs from the main part of data. Usually these suspicions cases are signals for the problem. These points could not be an actual failure, but they will signalize that this particular censor or equipment is no longer in normal condition. Based on founded outliers, system can predict future failures. This approach is also well known as condition monitoring.

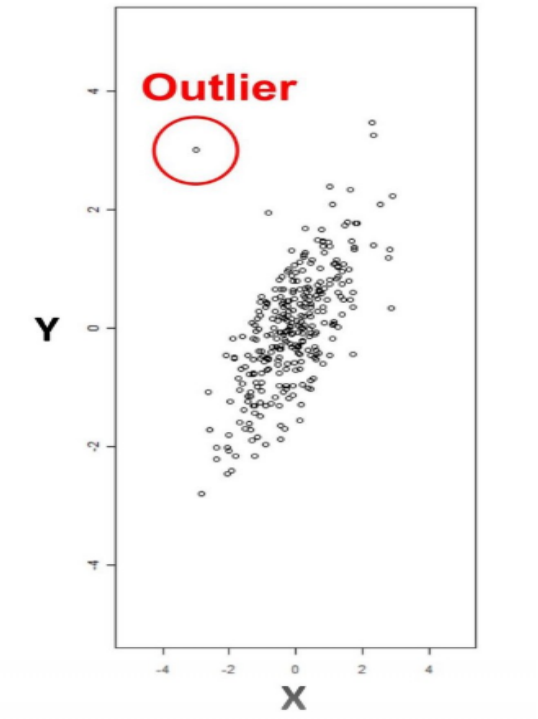

FIGURE 4.1: Outlier visualization

There are a lot of ways to do so, but they are divided into 2 main approaches. Machine Learning and Deep Learning. For this project I'm still in the process of finding best way and algorithm for it, so in this chapter I would like to write which methods were used their pros and cons.

4.2 Multivariate statistical analysis

In our case we could not build our decisions on data from only one censor, there will be always combination of different circumstances which in total could bring the real indication of the situation. So first of all, dimensionality reduction techniques need to be applied, in order to compress data to less dimensional one without big data loss.

PCA

One of the most wide -spread methods is PCA – principal component analysis. Main idea of this method to create a linear mapping from original data to lower-dimensional space, in such way that the variance of the data in this space is maximized. This algorithm is quite straightforward. Firstly the covariance matrix of initial data is constructed. From this matrix we can compute it's eigenvectors. That the eigenvectors which correspond to largest eigenvalues can be used to construct matrix, with a help of which the original matrix could be transformed to a lower-dimensional space formed by eigenvectors.

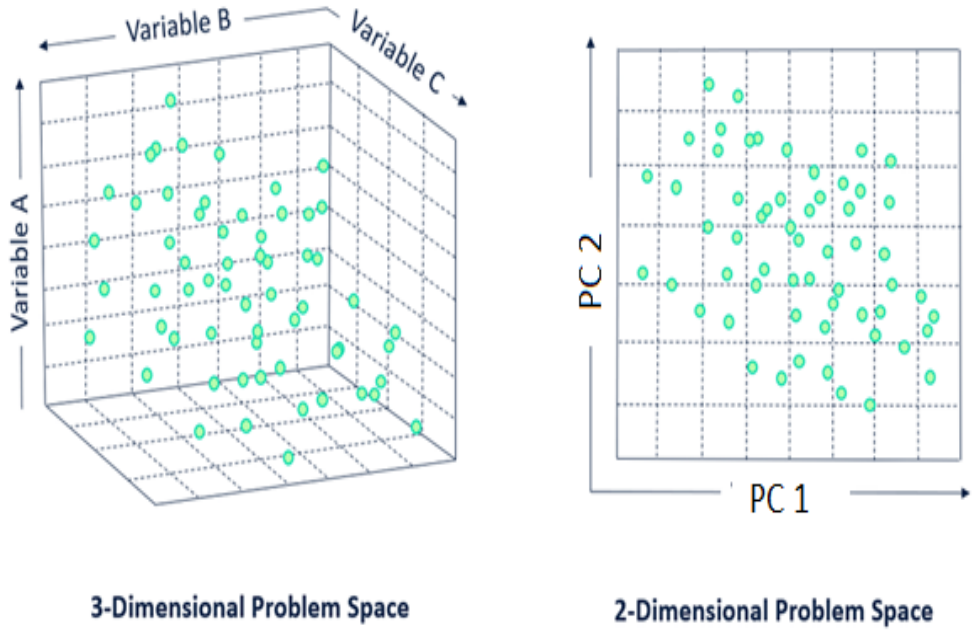

FIGURE 4.2: PCA example

Anomaly detection

Basically, problem of finding anomaly boils down to finding probability if point consider to be unnormal or normal. For doing so fist step is to calculate probability distribution $p(x)$ from our data points. Than when we will be analyzing next point x_1, we will need to compare $p(x_1)$ to some threshold - t. The anomaly points usually would have small $p(x)$, so comparing $p(x_1)$ to t would give us an answer. If $p(x_1) < t$ point will be considered as anomaly.

Mahalanobis distance

The Mahalanobis distance measures distance relative to the centroid — a base or central point which can be thought of as an overall mean for multivariate data. The centroid is a point in multivariate space where all means from all variables intersect. The larger the MD, the further away from the centroid the data point is. The most common use for the Mahalanobis distance is to find multivariate outliers, which indicates unusual combinations of two or more variables. The Mahalanobis distance between two objects is defined (Varmuza Filzmoser, 2016, p.46) as:

$d(Mahalanobis) = [(x_B - x_A)^T * C^{-1} * (x_B \check{} x_A)]^{0.5}$ Where: x_A and x_B is a pair of objects, and C is the sample covariance matrix.

4.3 Artificial Neural Network

4.3.1 General

Artificial neural networks, further – ANN, is one of the fastest growing areas in machine learning. From word "neural" obviously comes that this system inner logic was inspired by brain structure. The main idea of it was to recreate way how our brain works, learns and understands. Main structural part of these are input, hidden and output layers. Their roles are: input of the information, processing info, and output the result correspondingly.

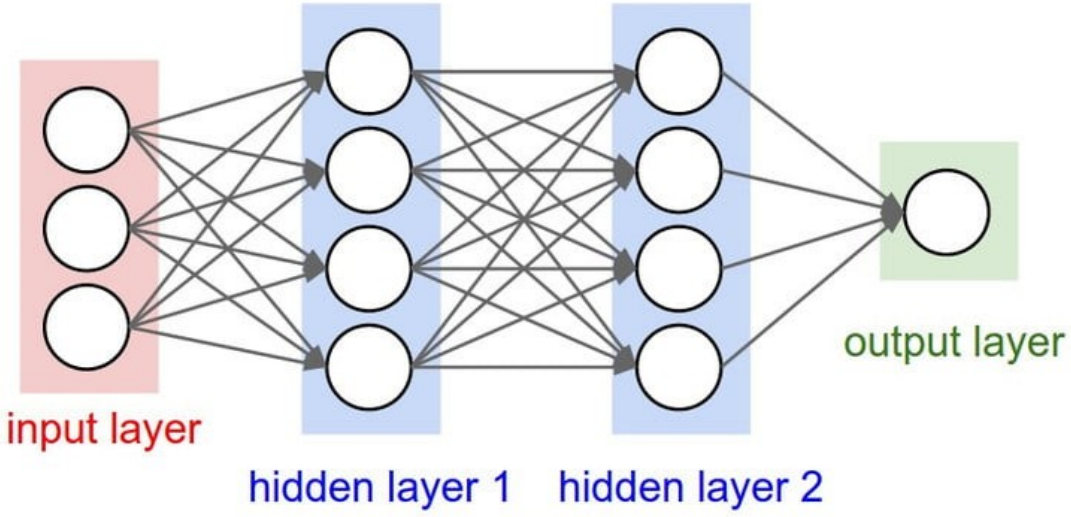

FIGURE 4.3: ANN schema

The idea is the following: every cell in hidden layer represent a single neuron in a brain. In ANN it is represented by a differentiable function, which is called – activation. It is connected to a neurons in a previous layer and accepts their output signals as inputs, as shown in a figure below. Based on these signals it compiles it's own output signal which will be transmitted to next layer of neurons. This compilation is based on special weights which are assigned to each input signal. Adjusting these weights, the main goal of training process. This training process is based on gradient descent, and it's aim to minimize difference with output which is given by ANN and real result. For doing this we need to define loss function which will give as a measurement of how far our output is from real result, this process is called forward propagation. Now, when we have our measurement of error we need try

to minimize it, using gradient descent, having the fact that all neurons are differentiable functions, we can compute gradients to determine how much and to what direction we should adjust our weights, this is called back propagation. Repeating this process N times, will leave us with weights that guaranties the minimal possible error between ANN output and real result with current network params.

4.3.2 Autoencoder NN

An Autoencoder is a type of ANN which is used to learn data dependencies in unsupervised way. The main goal of this type of network is to learn to reconstruct its own input from its reduced encoding representation. Sound's similar to dimensionality reduction, isn't it? Simplest form and autoencoder can be illustrated as following:

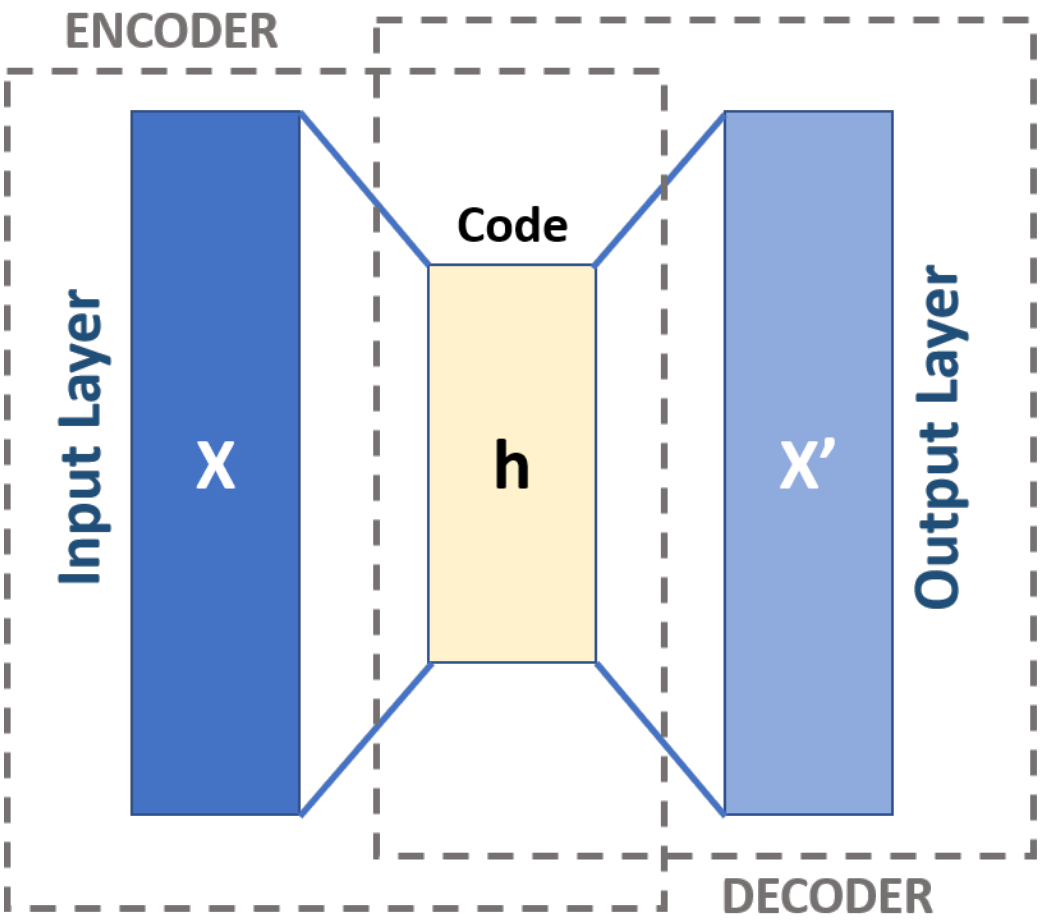

FIGURE 4.4: Autoencoder NN schema

I our case, the goal is to use autoencoder to transform data from sensors to a lower-dimensional encoded state, where using features of this ANN type, it will detect correlations and bindings between given variables. The main advantage of this approach over the PCA, that it cal also detect non-linear dependencies.

But how it can be used for anomaly detection? The main idea is the that as the target sensor will be giving inaccurate data, this will affect the correlations in the variables. Giving this we will see the increasing error in network output layer as it represents the reconstruction of giving data. By watching over this error, we can see the "health" of an sensor.

In my case I used the following configuration:

3-layer network: first layer 10 nodes, middle layer 2 nodes, and 10 nodes in third layer.

Mean square error was used as loss function

Model trained with ADAM optimizer, with 10 epochs with batch size of 10.

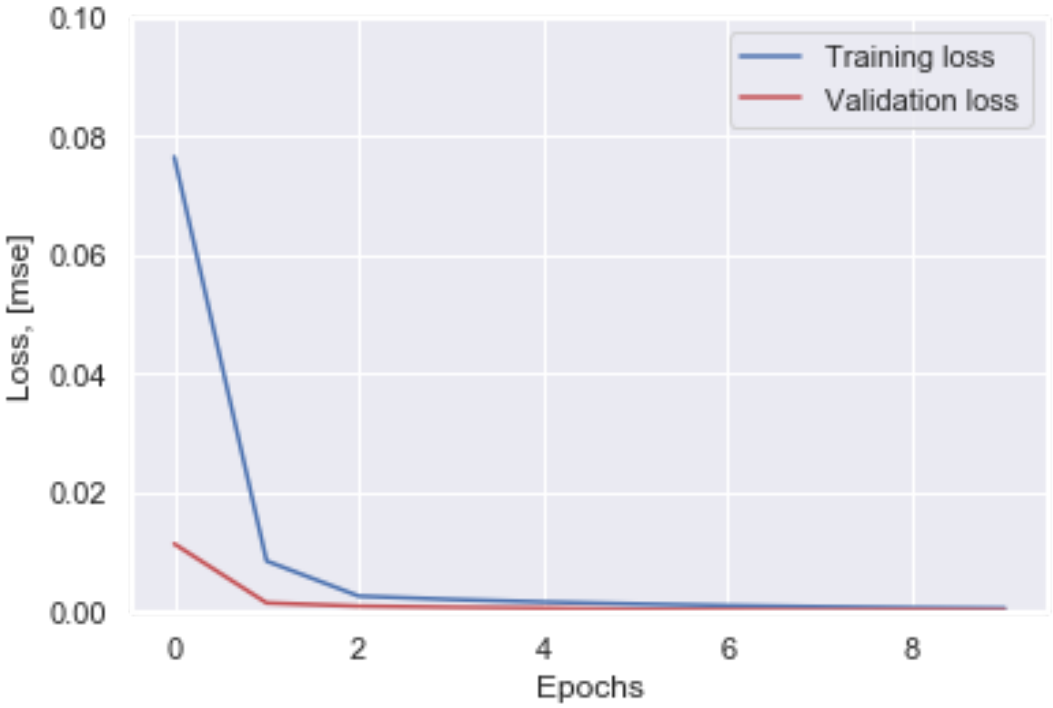

FIGURE 4.5: Loss function

As you can see, training and validation losses after fifth epoch are tending to zero.

4.3.3 Results

During testing runs, I was using datapoints from engine RPM and corresponding throttle position. As you can see from the data visualization below, RPM is increasing and decreasing correspondingly to throttle position. For simulated anomaly in last records, where RPM goes significantly higher, throttle values were set to 0.

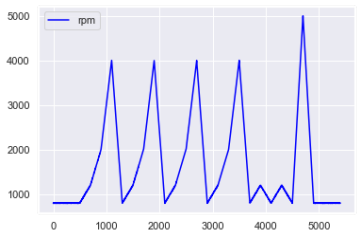

FIGURE 4.6: RPM dynamics

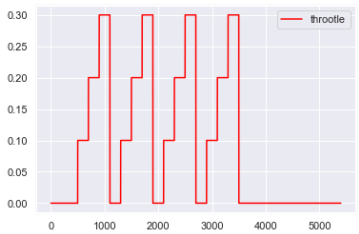

FIGURE 4.7: Throttle dynamics

As was described above, Mahalanobis distance is the evaluator of error in the model. Once it's exceeds the threshold value, we can consider this as an anomaly

point. As you can see below, once RMP is starting to increase without changing of throttle position, this value is increasing significantly, which signalize about error.

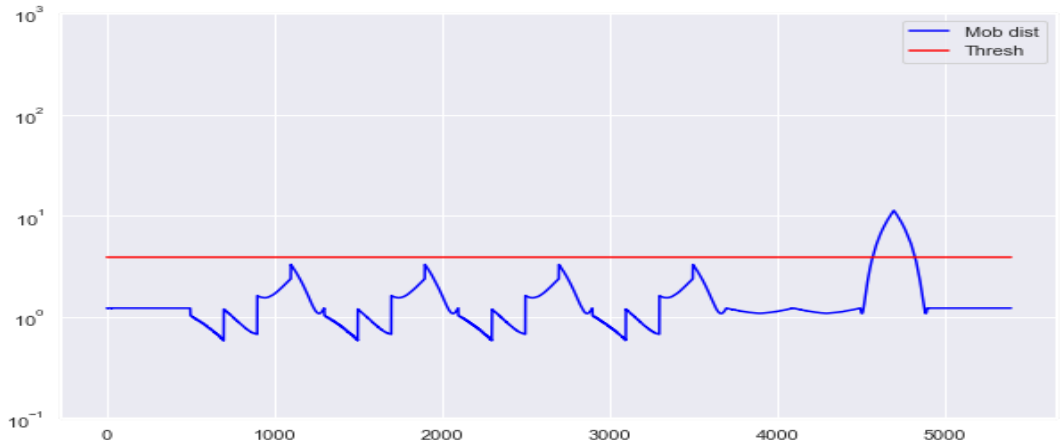

FIGURE 4.8: Mehalanobis results

In the case on ANN, the situation is almost same, the value of error is increasing in the anomaly area.

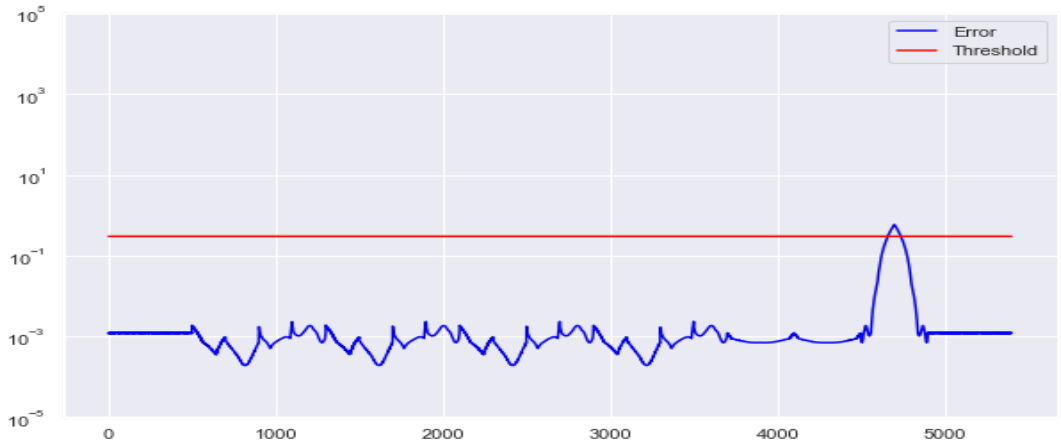

FIGURE 4.9: Autoencoder results

Both models performed similarly. The difference is in training time. Training ANN takes many times longer, but numbers are not very big around 3 sec, with 10 epochs. On the other hand I can assume that ANN will perform much better when I will include more engine parameters with non-linear relations.

The problem now is that models are performing good on relatively large anomalies but does not react pretty well on smaller. Because changing the RPM from 800 to 850 without changing acceleration pedal position should already be considered as anomaly. So the next steps in this process is to improve model "perception", teach it to react on smaller changes.

Chapter 5

Implementation

5.1 Technical side

In this section I would like to describe implementation process. The very first thing that I needed to collect many technical data about car, which functions are depending on which sensors, which sensors rely on each other, and which sensorics data will change accordingly to driver actions.

Data which is now collected by app and is used in algorithms:

- Engine temperature
- Engine RPM
- Selected gear
- Vehicle speed
- Throttle position
- Fuel condition

Engine temperature is very important for detecting work mode, which will be discussed further in this section. RPM one the best way to rate the health level on engine. When something is wrong in engine system, RPM will be the first who will fell it. In normal conditions engine RPM remains stable over time, in case of problem it's value can start jumping from higher to lower over time without changing throttle position or changing throttle position will lead to wrong (lower higher) RPM values. So, this number is target value for anomaly detection algorithm.

For a Petrol engine normal value of RPM when car is idle is – 700 - 800r/min, for diesel engine 500-600 r/min. But it makes no sense to analyze RPM itself, because cars are not just standing still, which results in continuously changing RPM value. That why algorithm is taking into consideration also throttle position and selected gear (if available).

The Logic is the following if RPM and throttle position values corresponds than this is not considered as an anomaly, but if RMP changes without change of throttle position, this will signalize about failure in engine system. Then app can propose to user possible solutions of the problem based on data. To make prediction more accurate in future other values that affects RPM could be added to the input data. Like Fuel condition or intake air pressure.

As was mentioned above car engine basically has 2 main working modes: - Warming up (-further on cold), and Normal operation (-further on hot). Censor data during these two modes can differ significantly. So for analyzing data I'm using two models trained on 2 different sets of data: obtained on cold, and obtained on hot. The difference is in idling state. During this time car CPU artificially increases

RPM in idling state, for engine to reach working temperature faster. And many all other paraments could be changes in order to warm engine faster. That's why engine temperature is quite important parameter.

Other parameters are not used in Machine learning, but still are used to detect anomalies, in more simplistic way of general algorithms:

- Run time since engine start
  - If temperature could not reach 90 degrees from 5 till 15 mins from starting of engine It could signalize about problem with engine coolant circle.
- Fuel Pressure
  - If pressure goes down when driver is pressing on acceleration pedal that could mean problems with fuel pump
- Low air flow rate
  - Could be the consequence of dirty air filte

I mentioned only a few and most common problem which system can detect. Giving all data, which car is providing system can predict very wide variety of failures.

Next step in this direction will a be a consultation with professional car mechanic, in order to list all dependencies between censor data and information which it gives.

5.2 Internal architecture

5.2.1 General Technologies

App is supposed to be backendless solution, so for storage of the user data Firebase can be used. It provides great service for backendless applications. Firebase-Auth is used for user authentication. In this app, authentication is not necessary, only by user wish, if he wants his data to be synchronized among different devices.

App Auth is made by phone number. For user's comfort users' number is auto populated from his google account. Firebase send a sms code to client device, this code is auto populated, so user don't have to enter it himself. This code is sent back to firebase for verification, if code is verified, user account gets created and he will have access to the DB to save his data.

But where is DB if A.D.E.S is backendless solution? And once again Firebase is helping with that. Firebase Cloud Firestore saves our day. It also has a security layer, so I can cleary define how user will access the DB and what data will be visible for them.

Firebase Cloud Firestore is used to store trained models for users for them not to re-train them once again if they will change a device and some general anonymous data for client analysis. Such as location, car, time in app, time of training etc. This data in future will help to better understand clients, and improve application accordingly to their needs

5.2.2 App architecture

Main language for an application was chosen to be Kotlin. This is relatively new language for Android development with many cool features which simplifies development process. Code on it usually has 25% less lines that Java. What is more, it

allows direct calls to native Java code, which means full support of Java -libraries. I was trying to use latest technologies and best practices presented on Google IO.

App is based on MVVM (Model – View – ViewModel) architecture, and is using RxJava / RxKotlin for handling asynchronous operations.

To explain what it is, firstly I need to explain the meaning of Android Lifecycle. Every UI screen in Android system has it's own living time, and during this time it goes through different stages. So the biggest pain point in all of this, is that if you will leave some business logic on UI screen, you could not be sure that it will not be killed by system and your computations would be stopped.

For ex: When you rotate your Android phone, it not just changes the way it looks, under the hood portrait version of screen is destroyed, and recreated in an landscape version. That's why all important data and business logic should be separated from it to some lifecycle-aware components.

On of them is ViewModel. It's is relatively new architecture component, firstly introduced in 2017. The biggest advantage of it, that it is UI-independent and if used in right way, can ensure you that your business logic is save from any lifecycle changes. So View (UI part) is only responsible for displaying data which ViewModel send to it. And if it will be destroyed ViewModel will just wait for next screen to be attached to it.

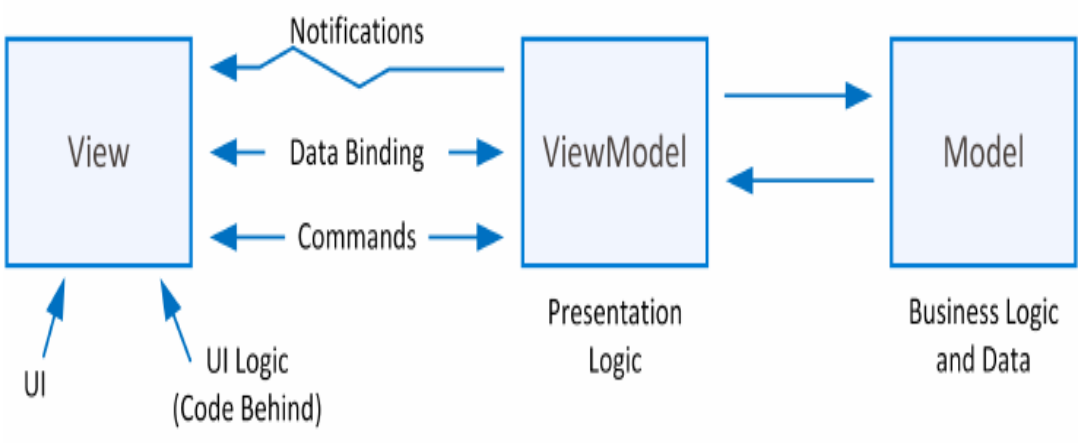

FIGURE 5.1: MVVM

Now I will try to describe technical part of the application, how relations are build inside of it, but not going to deep, for general understanding. In this section the most important components will be mentioned

Device manager

This Singleton class is fully responsible for actual communication with OBD-2 adapter. It's responsibilities include:

- Sending commands to data stream obtained from Connection manager
- Retrieving data from data stream and decoding data from bytes that it gets.
  - Request structure was explained in this section
- Delegating commands to Connection manger to scan, connect, dispose devices

Data Manager
Singleton class responsible for transforming raw data obtained from Device Manager to a form that it will be understandable for ML Models and Car Control system. It combines corelative params into packages with which next analyzes module would be comfortable to work with.

Fox example data for engine current RPM and throttle position are obtained from two different requests which are executed in two separate threads, Data Manager should combine them in one data package and send it for further analysis.

Car State Manager
Singleton class aimed to monitor car state, it detects changes of work mode of an engine (cold hot), which will trigger changes in ML Models behavior. Also it's responsible for storing current vehicle state. Here params which does not play role in ML algorithms are analyzed.

ML Model Manager
Singleton class responsible for ML Model management. It manages the state of ML Model, responsible for training and saving to internal memory. Main working params are work mode (on Hot on Cold) received from Car State Manager and data from Data Manager. Accordingly to work mode it creates instance of Actual Model and is responsible for training and saving it's trained state to memory.

ML Model Manager is also responsible for getting predictions from trained models accordingly to current work mode and transmitting results to a Control Center.

All computations are made in separate working threads, so this does not affect the main UI thread.

Control ViewModel
ViewModel class which represents the core of the system. It's responsibilities include:

- Sending data from Car State Manager to UI
- Managing state on ML Model Manager
- Transmitting user commands from UI to business logic part
- Delegating commands to corresponding manager Simplified version of these relations you can see at the following graph

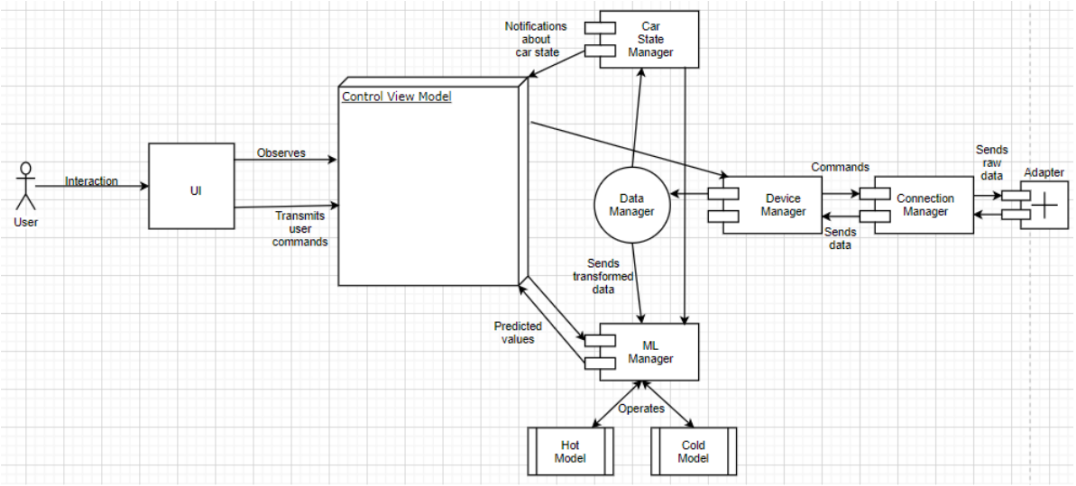

FIGURE 5.2: Application architecture

5.2.3 Database

External

For an external database Firebase Cloud Firestore was used. Here, in case user went through authentication flow, app saves necessary data for data synchronization among multiple devises.

Firebase Cloud Firestore is NoSQL database, unlike a SQL database, there are no tables or rows instead it stores data in a form of documents, which are grouped in collections for data organization and query building. They are organized in hierarchical model, from main collection to sub collections and with a simple document in the end. Each document is lightweight object, in fact a collection of key – value pairs, some kind of JSON object, with restriction of size (< 1MB) Here is an example of data record:

FIGURE 5.3: Document in Firebase

The communication with database is made by subscription model. Unlike common SQL databases, where data is obtained via queries, Firestore is following some sort of Observer pattern. Client in this case – mobile app, is subscribed on data updates, that means that every time data field is updated, client is notified about this change and will receive updated document. This feature becomes very useful in case if user is using different devices. The data obtained from one device, will automatically synchronize with second one.

Other big advantage of this database is offline support. As we know that A.D.E.S will be used in car, we can assume that there could be problems with network along

the way of traveling, which can lead to data loss, but Firestore solves this problem for us. During connection losses data is stored in local storage and when internet connection will be available it will synchronize data with remote database.

Internal

In A.D.E.S application local storage is also used. For this reason ROOM library is used. This is official Google library which provides and abstraction level over SQLite DB which lies under the hood. Local storage is stored for the same reasons as Firestore, which was described above, to ensure no data loss, and to save data of the non-authenticated client. Also all necessary data obtained from sensors is stored locally in this DB.

5.3 Design Prototypes

As A.D.E.S is a mobile application, it requires professional designs to be attractive for customers from UI / UX point of view.

Some designs has already been made, so in this section I would like to share them with you

5.3.1 Car screen

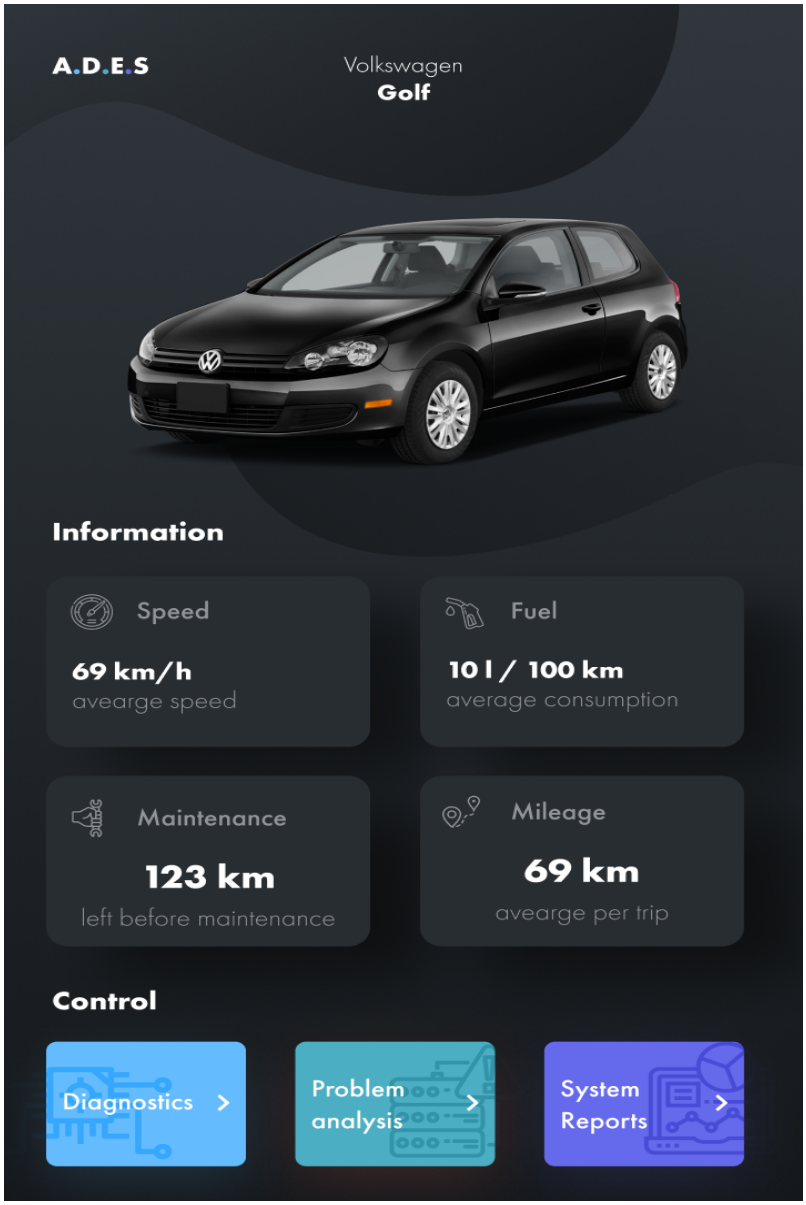

FIGURE 5.4: Car info screen

This screen is responsible to show main statistical data about user's car. Values like average speed, fuel consumption. From here user can open inner app features. The main car image will change accordingly to the information acquired from the car itself. Images will be stored in Firebase with public read rights.

5.3.2 Engine state screen

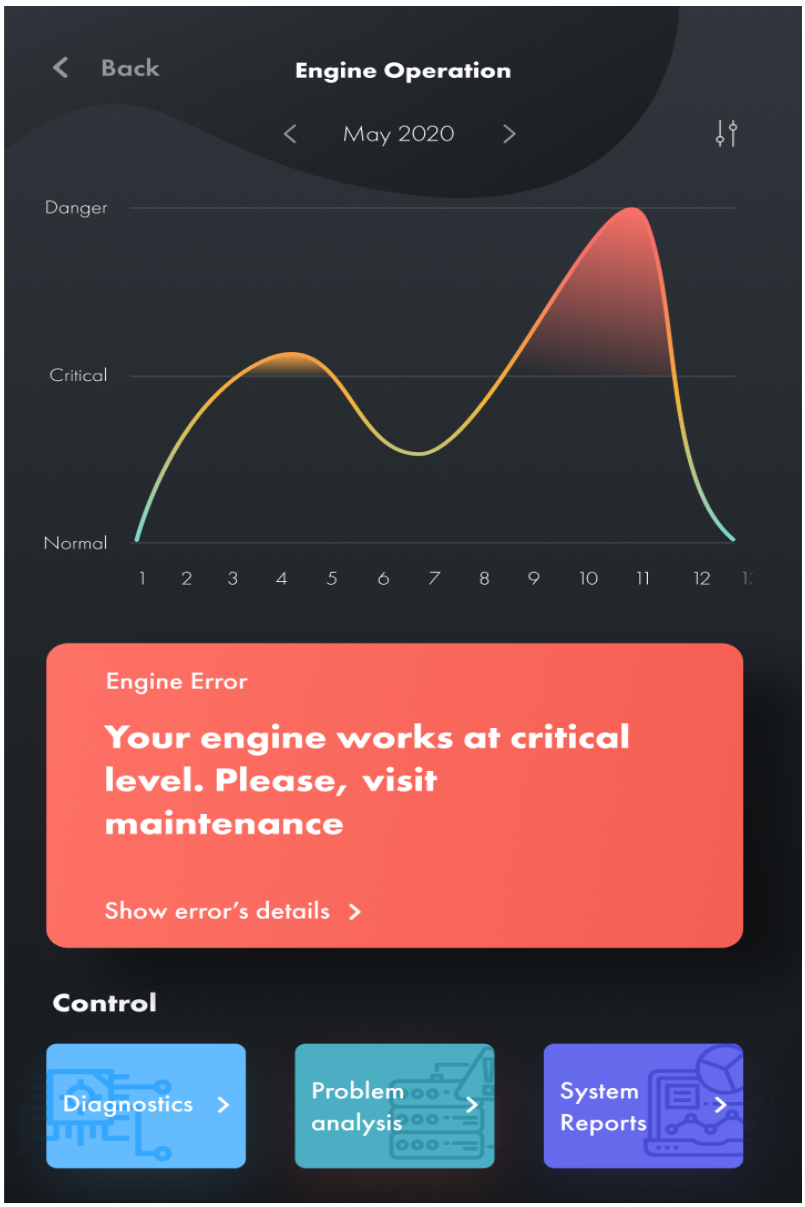

FIGURE 5.5: Engine state screen

The engine statistics screen. This graphic will be compiled from numbers of detected anomaly states inside if the application. So, this screen does not show dynamics about concrete sensor, because for most of the car users nowadays don't know which sensors is responsible from which function, but instead it shows general system dynamics. If number of error exceeds "normal" threshold, system with notify user about critical engine state, and will advise to visit maintenance station and will try to help with usefull tips and error details

5.3.3 Connections screen

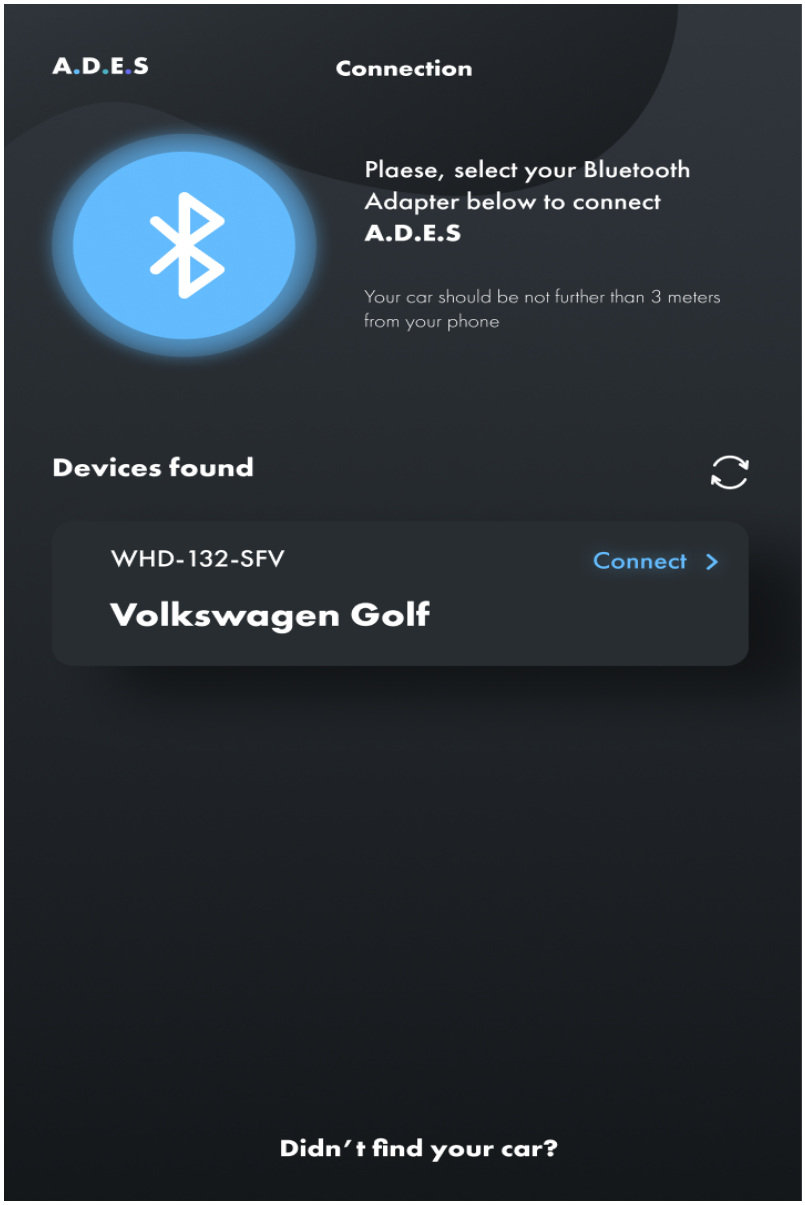

FIGURE 5.6: Connections screen

Simple connections screen, which will allow user to choose device to connect.

5.4 App versions

As every software product, A.D.E.S app development process was divided in separate versions to ensure smooth development process. Versions roadmap:

5.4.1 POC

At this stage the basement of the application was formed. Added Bluetooth functions, ability to connect to devices and read basic data.

Functions:

- Bluetooth scanning / connecting
- Data reading from adapter

5.4.2 MVP

At this stage basic valuable functions appear. Having functions from POC version. App now has abilities to communicate with main sensors such as: RPM, Speed, Car info, Throttle position, etc. Store this data for further analysis. Authentication methods are included as well as data base connections. ML side of the project is being developed. Acquired first results from both algorithms.

Functions:

- Read real data from car
- Decoding this data from byte format to real numbers
- User Authentication
- Firebase connection and data storage
- First attempts on training ML algorithm and ANN (separately using Python)
- Internal logic reacts on car state changes (on cold on hot)

5.4.3 V0.1

In this version all functions from an MVP are improved, first designs are getting implemented. ML algorithms are under investigation and improvement. New data points are added to the training process such as selected gear, fuel condition and data from air sensors.

Functions:

- First designs implemented
- Improved ML algorithms
- Full analysis of car data

5.4.4 V0.5

The rest of designs will be ready for implementation. ML models will be imported from Java to Kotlin. Start of testing in real driving scenarios.

5.4.5 V.** Future Plans

5. V.** Future Plans In next app versions will be added the following functions:

- Car profile
- Tracking car overall health
- Maintenance profile
- After consulting with car mechanic specialist, add ability to detect most common failures:

  - Fuel rail problems
  - Fuel pump degradation
  - Gearbox problems
  - Failures in ignition system
  - Alternator degradation
  -

- Data synchronization among multiple devices

Chapter 6

Results

6.1 General

In this paper I tried to cover all topics of development cycle around A.D.E.S app, from technical details to inner software architecture. As a result, now I have a working project, which already has all vital parts to be finished till going live state.

The whole development process was divided into 3 stages. At first stage I was investigating technical side of the problem. As workflow of IVS engine is the target is the application, I needed to understand the basic principles of its work. Which parameters are connected and how censors data is connected to engine health state. A lot of work was done to become acquainted with OBD-2 technology and to understand communication protocols.

Second stage was dedicated for investigation of possible ML and DL techniques. In the end two were chosen: Mahalanobis distance and Autoencoder Neural Networks. By this time both models performed relatively similar, so work on improving them is going on.

The third stage was dedicated to software development process, during it app architecture, was made and main service features was implemented. During development time I was using the following technologies:

- Android SDK along with Kotlin programming language
- RxJava / RxKotlin for Handling asynchronous app flow.
- Android Architecture components
- RxAndroidBle for establishing Bluetooth connection with device
- obd-java-api for actual communication with device
- Firebase Auth for user authentication
- Firebase Firestore as remote NoSQL database
- SQLite with ROOM adapter for internal database

6.2 Improvements to be made

By the time of writing this article app is in development, current version - MVP. The main focus for now is improving ML algorithms, testing it with different data combinations and internal architecture(in case of ANN). Having current results model could recognize anomaly points, when difference is significant and fails in a smaller errors. Solution to this will be enlarging training datasets, for models to better group the normal points.

6.3 Weak points

The main, and for now unsolved problem is false positive data readings. App is self-thought, that means that I'm not using one pre-trained model for all devises. Every device train it's own model based on data from a specific vehicle. That means if at the moment app is gathering first training data, one of the target sensor is already giving wrong data, or engine is already in a poor health. Algorithm will learn on this "unhealthy data". Which means he will treat normal working conditions as anomaly. The solution could be, before training model, check general statistics with some reference values obtained from data base. But this requires investigation.

6.4 Summary

In the end I would like to say, that A.D.E.S project demanded from me all my knowledge from different areas. Starting from IVS engine structure to Deep Learning with ANN. The road to this point was hard, but a lot of work still needs to be done. Thanks for your attention.

Git: Android App
Git: ANN
Git: Mahalanobis
Video: Demo

Bibliography

[1] OBD-2 Java

[2] ELM Electronics - Understanding of OBD

[3] Car Doctor. Histroy of creating OBD-2 Android app

[4] ELM Electronics - OBD commands

[5] ELM327 Microcontroller

[6] On-board diagnostics

[7] OBD-2 PIDs

[8] SAE International - J1962 adapter

[9] SAE Mobilius J1962 specification

[10] Autoencoder

[11] Condition monitoring

[12] Principal component analysis

[13] Bluetooth low energy

[14] RxAndroidBLE

[15] Understanding Principle Component Analysis(PCA)

[16] Why is the Mahalanobis Distance Effective for Anomaly Detection?

[17] *A Cluster-based Algorithm for Anomaly Detection in Time Series Using Mahalanobis Distance* Ryo Kamoi July 2015

[18] *Why is the Mahalanobis Distance Effective for Anomaly Detection?* Erick Giovani Sperandio Nascimento1a, Orivaldo de Lira Tavares1, and Alberto Ferreira De Souza February 2020

[19] *Introduction to Multivariate Statistical Analysis in Chemometrics. CRC Press* Varmuza, K. & Filzmoser, 2003

[20] Introduction to Artificial Neural Networks