VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SKUTEČNĚ CHYTRÁ CHYTRÁ ZÁSUVKA

TRULY SMART SMART SOCKET

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. ONDŘEJ VALUŠEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ZDENĚK MATERNA, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání diplomové práce

Student: **Valušek Ondřej, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Počítačové vidění
Název: **Skutečně chytrá chytrá zásuvka**
Trully Smart Smart Socket
Kategorie: Vestavěné systémy
Zadání:

1. Vyberte vhodnou chytrou zásuvku, která umožní lokální ovládání a zejména pak vyčítání aktuální spotřeby s dostatečným časovým rozlišením.
2. Nasbírejte data z vybrané zásuvky pro různé spotřebiče a jejich kombinace.
3. Navrhněte způsob identifikace jednotlivých spotřebičů na základě zaznamenané spotřeby.
4. Navržené řešení implementujte.
5. Řešení otestujte s různými typy spotřebičů.
6. Diskutujte dosažené výsledky a zvažte případná rozšíření či vylepšení.
7. Vytvořte video prezentující vaši diplomovou práci, její cíle a výsledky.

Literatura:

- Raspopov, Dmitriy, and Pavel Belousov. "Development of methods and algorithms for identification of a type of electric energy consumers using artificial intelligence and machine learning models for Smart Grid Systems." *Procedia Computer Science* 169 (2020): 597-605.
- Tekler, Zeynep Duygu, et al. "Near-real-time plug load identification using low-frequency power data in office spaces: Experiments and applications." *Applied Energy* 275 (2020): 115391.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Materna Zdeněk, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 18. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Na trhu je dnes k dispozici mnoho takzvaně chytrých zásuvek. Jejich využití je však značně omezené. Typicky umí měřit spotřebu a lze je spínat na dálku pomocí mobilní aplikace nebo časovače. Tato práce řeší, jak využít spínací modul s měřením spotřeby k tomu, aby se ze zásuvky stala skutečně chytrá zásuvka, která umí rozpoznat co je do ní aktuálně připojeno pouze na základě krátkého časového okna, a to až pro tři spotřebiče najednou. Spotřeba je měřena chytrým relé Shelly 1PM společně pro tři zástrčky. Extrakcí příznaků z časové řady, detekcí neznámých spotřebičů pomocí SVM a poté klasifikaci neuronovou sítí se podařilo dosáhnout přesnosti přes 99 % na datasetu obsahující různé kombinace zapojení chytré televize, stolní lampičky a notebooku. Informace o aktuálně připojených spotřebičích jsou přehledně zobrazeny ve webovém rozhraní a také jsou průběžně zapisovány do databáze pro zpětné zobrazení statistik. Informaci o připojení a odpojení spotřebičů je také dále možné poslat do systému pro správu chytré domácnosti.

Abstract

There is a large selection of so called smart sockets available on the market today. The possibilities of these sockets are sadly very limited. Typically, they can measure power consumption, be turned off and on remotely by mobile application and timer. This thesis deals with this problem by showing how a smart relay can be used to create a truly smart smart socket that can classify currently connected appliances using just short time window for up to three devices combined. The power consumption is measured using Shelly 1PM together for three plugs. Using time series feature extraction, unknown device detection with SVM and neural network classification, the accuracy was over 99%. on a dataset containing combinations of smart TV, lamp and a laptop consumption. Information about currently connected devices is displayed on a webpage and written to a database to be viewed later. The information about connecting and disconnecting a device can be further sent to a system for smart home management.

Klíčová slova

měření spotřeby, chytrá domácnost, chytrá zásuvka, časová řada, klasifikace časových řad, klasifikace signálu, klasifikace spotřebičů, extrakce příznaků, neuronová síť, SVM, Node-RED, Flask

Keywords

power consumption measurement, smart home, smart socket, time series, time series classification, signal classification, appliance classification, feature extraction, neural network, SVM, Node-RED, Flask

Citace

VALUŠEK, Ondřej. *Skutečně chytrá chytrá zásuvka.* Brno, 2022. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Zdeněk Materna, Ph.D.

Skutečně chytrá chytrá zásuvka

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doktora Materny. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Ondřej Valušek
15. května 2022

Poděkování

Děkuji Ing. Zdeňku Maternovi Ph.D. za poskytnutí cenných rad při řešení této diplomové práce.

Obsah

Kapitola 1

Úvod

V poslední době je možno sledovat rychle se zvyšující počet chytrých zařízení v domě. Podle statistik [1] z konce roku 2021 lidé na světě za chytrá zařízení utratili přes 39 miliard dolarů. Autoři předpovídají, že toto číslo se má každý další rok zvyšovat o přibližně 15 %. Na internet jsou dnes připojeny pračky, ledničky, televize a další spotřebiče.
Přesto, že počet těchto chytrých zařízení v poslední době rychle roste, nezřídka kdy je část spotřebičů v domě chytrá a část nikoliv, například proto, že se ani žádná chytrá varianta nevyrábí. Jistě by bylo zajímavé pomocí modulu do zásuvky za pár set korun udělat z několika „hloupých“ zařízení „chytrá“. Fungující zařízení, jistě nemá smysl měnit za nové jen proto, že se neumí připojit na internet. Zásuvky dostupné na trhu toto však typicky neumožňují. To, co výrobci prezentují jako chytrou zásuvku většinou znamená měření spotřeby a zapnutí/vypnutí v aplikaci mobilního telefonu.

Pokud by byla k dispozici skutečně chytrá chytrá zásuvka, konec pracího cyklu by mohla rozpoznat zásuvka a nebyla by potřeba chytrá pračka. Zásuvku by také bylo možné použít například k rodičovské kontrole, zda a kdy má dítě zapnutý počítač nebo televizi. Také by bylo možné nadefinovat složitější pravidla. Ta by jako další vstupní informaci mohla mít třeba aktuální čas nebo stav jiných spotřebičů v domě. Taková pravidla by pak mohla sloužit k ovládání dalších prvků domácnosti. Kromě tohoto by také bylo možné zobrazovat přehled o spotřebě zařízení a to zpětně, nikoliv jen vypisovat aktuální spotřebu.
Právě postupem, jak udělat pomocí jednoduchého měřáku za pár set korun ze zásuvky opravdu chytrou, takovou, která by umožňovala vše výše uvedené se tato práce zabývá.

V kapitole 2 jsou popsána jak komerční řešení, tak dosavadní přístupy některých podobných prací, které se však zaměřují především na pomalou detekci na velkých datasetech. Tedy především detekci pro účely statistik používání a optimalizaci spotřeby. V kapitole 3 je popsán nový přístup, který umožňuje detekci v řádu několika sekund.
Nejdříve je popsán výběr chytrého relé v sekci 3.1, poté v 3.2 architektura databáze. Dále je v sekci 3.4 představeno, jak probíhá klasifikace a detekce neznámých spotřebičů. Aby bylo možné data dále využívat, v sekci 3.6 je ukázán návrh webového rozhraní, které obsahuje informace o momentální spotřebě, aktuálním stavu připojených spotřebičů, zpětné statistiky o spotřebě a možnost konfigurace systému.

V 3.7 je pak ukázáno, jak lze chytrou zásuvku začlenit do chytré domácnosti a využívat tak komplexnější logiku v rámci nástroje pro řízení chytré domácnosti.

V kapitole 4 je dle výše uvedeného návrhu rozebrána implementace za použití konkrétních nástrojů a knihoven. Na závěr je pak v kapitole 5 přehledně zobrazen dataset, použitá metodika pro vyhodnocení a dosažené výsledky.

Kapitola 2

Chytrá domácnost

Tato kapitola představuje základní dále používané pojmy a zasazuje představovanou práci do kontextu aktuálního výzkumu v oblasti. Budou uvedeny pojmy jako chytrý dům a domácnost. Poté budou představeny práce z podobné oblasti, a to jak komerční řešení, tak otevřený software. Nakonec budou uvedeny i řešení vzdálenější od chytrých zásuvek, která se však zabývají podobnými problémy.

2.1 Internet věcí a chytrá zařízení

Původní definice chytré domácnosti jako takové je z roku 2003 [8]. Lze ji volně přeložit jako „Domácnost obsahující komunikační síť, do které jsou zapojeny klíčové elektrické spotřebiče, které mohou být na dálku ovládány a monitorovány“.

V článku [7] je popsáno několik málo kategorií a pro každou z nich nejznámější zařízení. Protože se tehdy jednalo o skutečné počátky, objevovaly se i věci jako chytrý polštář pro správu knih, chytrá rohožka, která měla ukazovat kdo je v domě a podobně. Šlo tedy o první nastínění toho, jakým způsobem si tehdy lidi představovali chytrou domácnost.

Protože byly v té době chytré spotřebiče drahé a na trhu jich bylo málo, začalo se více výzkumu [6, 11] ubírat směrem k optimalizaci využití energie než tomu, jak by mohla probíhat komunikace mezi spotřebiči. V těchto pracích je tedy především rozebráno kolik procent elektrické energie by se mohlo uspořit třeba zapínáním některých spotřebičů v různou dobu a podobně. Dnes se již velká část chytrých zařízení a výzkumu kolem nich ubírá také zájmovým směrem.

O pár let později v roce 2013 byla publikována práce [4], která představila myšlenku většího počtu levných relativně chytrých zařízení propojených v cloudu, tzv. internet věcí. Autoři procházejí historii, současný stav, možný stav v budoucnu a také potenciální bezpečnostní problémy. Naznačují, že vývoj se pravděpodobně nebude ubírat směrem pár chytrých a drahých zařízení, ale že nějakým způsobem chytré budou i levné spotřebiče a hlavní „mozek“ by mohl být právě v cloudu, zatímco levná chytrá zařízení by jen měřila a posílala data.

V současnosti uvádí mnoho zdrojů velmi různé počty aktuálně připojených zařízení do internetové sítě. Jedno ale mají společné. Jde o desítky miliard zařízení.

Navzdory rychle rostoucím číslům těchto zařízení je však málo kdo ochoten vyměnit relativně novou televizi, pračku nebo lampu za novou, jen proto že ta stará není chytrá a neumí se připojit na internet. Například v průzkumu z roku 2021 [16] uvedla většina (přes 51 %) respondentů, že novou televizi kupuje jednou za více než 8 let.

V současné době se tak řada lidí nachází někde na půli cesty [17], kdy je část spotřebičů chytrá a část nikoliv. Zde je tedy prostor pro zařízení, které určitým způsobem dělá z hloupého zařízení chytré při minimální investici času a peněz.

2.2 Existující komerční řešení

Chytrých zásuvek je dnes na trhu mnoho. Ceny se pochopitelně liší podle výrobce, kvality zpracování a možností, které zásuvka nabízí. Chytrost těchto zásuvek je však velmi diskutabilní. To, co je běžně vydáváno za chytrou zásuvku typicky znamená schopnost měření spotřeby, odeslání dat do cloudu a zobrazení grafu s průběhem naměřených hodnot. K tomu zapnutí a vypnutí na dálku přes aplikaci na mobilním telefonu nebo dle časovače.

Lepší zásuvky k výše uvedenému podporují ještě časové rozvrhy s pravidly. Tedy například pravidla typu, v 18 hodin zapnout světla v kuchyni apod. To již působí o mnoho lépe než pouhé měření spotřeby. Problém je zde ale cena zásuvky. Aby takové pravidlo fungovalo, musí být do zásuvky připojen jen jeden typ spotřebiče (například pouze světlo), které zásuvka svým zapnutím rozsvítí. Každý spotřebič by tak potřeboval svou chytrou zásuvku, což se při ceně cca 1000kč za jednu dost prodraží, a navíc je uživatel limitován tím, co mu umožní výrobce, obtěžován složitější montáží a podobně. Několik existujících komerčních řešení bude nyní představeno.

Loxone

Loxone je výrobce systémů pro chytrou domácnost. Narozdíl od dále představovaných služeb se jedná o kompletní řešení chytré domácnosti. Loxone nabízí v podstatě vše, co by uživatel mohl v chytré domácnosti chtít v jednom ekosystému. V nabídce má reprosoustavy, audioservery, prvky pro zabezpečení a intercomy, osvětlení nebo naopak stínění. Kromě těchto běžnějších zařízení však lze pomocí výrobků této firmy spravovat i klima v domě (teplota, filtrace vzduchu) nebo filtraci bazénu. Výrobce klade velký důraz i na vzhled svých výrobků a na svém webu se například chlubí spoluprací s interiérovými designéry, nebo zahradními architekty.

Jeho největší nevýhodou je však cena. Z e-shopu firmy je patrné, že necílí na hobby nadšence, ale spíše na nové drahé domy, kterým nabízí kompletní řešení. Například pro přehrávání hudby v domácnosti je nutné koupit audioserver za přibližně 11 tisíc korun. Reproduktory pak začínají na ceně asi 20 tisíc za pár. Kromě audioserveru, který funguje samostatně je ostatní funkcionalita zajištěna pomocí tzv. miniserveru za 10 tisíc korun, ke kterému se pak připojují moduly pro ovládání jednotlivých prvků. Ty jsou opět poměrně drahé a zařízení pro ovládání čtyř světel stojí přes 10 tisíc korun.[1]. Velkou výhodou však je, že při použití těchto spínačů již nejsou potřeba chytrá světla, o plynulé zhasnutí se postará modul sám.

Druhou výhodou je pak poměrně pestrá nabídka zařízení. Pokud tak uživateli nevadí vyšší pořizovací cena, může být chytrá domácnost od firmy Loxone dobrým řešením, například při stavbě nového domu. Výrobky této firmy je také možné propojit se zařízeními jiných výrobců. Je možné připojit například baterii Tesla Powerwall, nebo spotřebiče značky Miele, tedy třeba troubu a podobně. Ovládat lze celý systém pomocí aplikace, nebo připojit další asistenty jako třeba Apple home kit.

[1] Všechny ceny bez DPH a ceny montáže.

(a) Miniserver (b) Modul pro řízení světel

Obrázek 2.1: Výrobky firmy Loxone.

Apple HomeKit

Apple HomeKit je protokol vyvinutý společností Apple. Narozdíl od zbytku ekosystému této značky, který je poměrně uzavřený, mohou s tímto API pracovat různí výrobci. Tento protokol podporuje široká škála výrobců jako jsou konkurenční Samsung, Xiaomi, Philips a mnoho dalších. Na Iphone je pak možné všechna připojená zařízení ovládat v aplikaci „Domácnost“, nebo použít hlasového asistenta Siri.

Drobnou, avšak pochopitelnou nevýhodou je pak nutnost přítomnosti některého ze zařízení Apple v domácnosti, pokud ji uživatel chce ovládat i z vnější sítě. Pokud se uživatel spokojí s ovládáním pouze tehdy, pokud je zařízení ve stejné sítí, není nic takového vyžadováno. V opačném případě se dá pořídit Apple HomePod, který lze koupit za přibližně 2500 korun, nebo pro tento účel použít třeba AppleTV.

Značným omezením však je, že ač mohou výrobci používat API pro integraci jejich výrobků do HomeKitu, aplikace pro samotné ovládání je k dispozici pouze pro zařízení Apple.

Google home

Google Home je další z řady chytrých asistentů. Narozdíl od HomeKitu společnosti Apple jej lze využívat jak v telefonech s operačním systémem Android, tak s iOS. Stejně jako Apple HomeKit je možné jej integrovat se zařízeními mnoha výrobců. Lze nastavovat připomenutí, hledat telefon a všechny ostatní akce, které by uživatel mohl od asistenta požadovat. Také je možné posílat HTTP požadavky na libovolnou adresu, čímž je možná další integrace.

Amazon Alexa

Alexa je virtuální asistent vyvíjený společností Amazon. Proslavena byla především díky hlasovému ovládání a zabudování do reproduktorů Amazon Echo. Tohoto asistenta lze však také použít k ovládání chytré domácnosti. Funkčností je přibližně srovnatelný s Google Home.

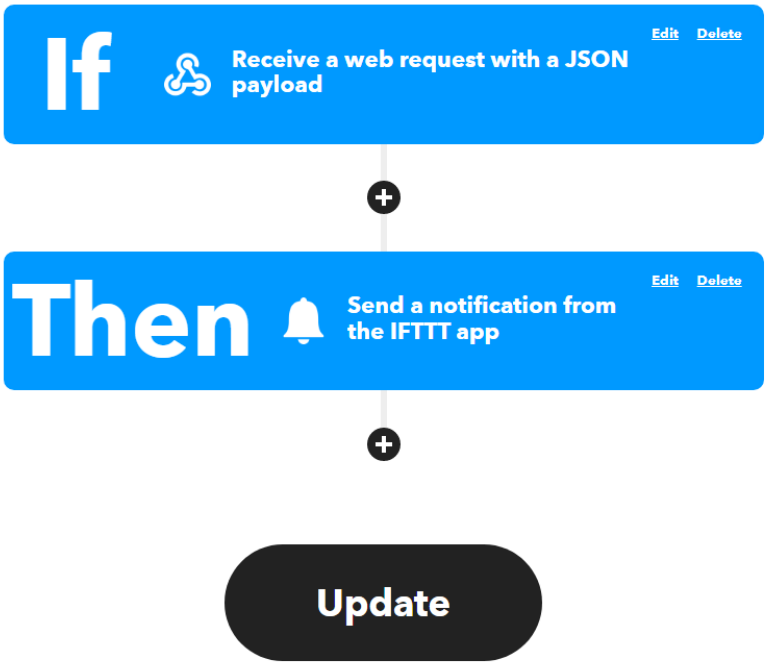
IFTTT

IFTTT je zkratka ze sousloví „if this, then that“, která už napovídá, k čemu služba slouží. Služba umožňuje vytvářet pravidla pro ovládání spotřebičů nebo dalších služeb. IFTTT

umožňuje jak ovládání prvků domácnosti, tak vytváření pravidel naprosto mimo bydlení a domácnost. Pravidla jsou zde nazývána jako tzv. Applety. Mezi nejoblíbenější[2] applety patří třeba:

- vyhledání a uložení písničky do oblíbených ve službě Spotify po označení to se mi líbí na serveru Youtube,
- automatické zveřejnění fotografie na Twitteru po jejím přidání na Instagram,
- automatická synchronizace kontaktů mezi Android zařízením a Apple zařízením,
- upozornění na telefon v případě, že má další den pršet.

Z výše uvedeného je patrné, že se nejedná primárně o aplikaci pro ovládání domácnosti. Díky možností použití webhooku lze ale provést integraci jakéhokoliv zařízení umožňující odeslání jednoduchého HTTP požadavku. Na adresu nastavenou v konfiguraci webhooku je možné posílat zařízením požadavky a ty dále zpracovávat a začleňovat do pravidel. Malou nevýhodou je zpoždění webhooku, které činí asi 5-10 sekund. To sice není dlouhá doba, ale pro některé aplikace může být komplikací.



Obrázek 2.2: Editace appletu v IFTTT.

Jak vypadá editace appletu používající webhook a zaslání notifikace je ilustrováno na obrázku 2.2.

Na obrázku 2.2 jsou kromě dlaždic vidět také malé plusy v černém kolečku. Ty značí že jak podmínek, tak akcí při jejich splnění může být více. Do podmínek je také možné přidat informace o aktuálním času, počasí a podobně. To však bohužel platí pouze pro placenou verzi. V základní bezplatné verzi jsou možná pouze velmi jednoduchá pravidla s jednou podmínkou a také jednou akcí při jejím splnění. Ceny předplatného na druhou stranu nejsou

[2]Podle počtu uživatelů.

nijak vysoké (v dubnu 2022 asi 110 korun měsíčně)[3] a uživatel dostane v podstatě hotové řešení umožňující integraci s mnoha službami a velmi jednoduchým rozhraním.

2.3 Existující algoritmy a datasety

Kromě toho, že je uživatel u komerčních řešení omezen tím, co nabízí výrobce může být problémem i cena. Pokud by někdo chtěl chytrými zásuvkami vybavit celý dům, cena začne dost rychle naskakovat. Nabízí se tedy otázka, zda by nebylo možné měřit několik zásuvek dohromady a disagregovat spotřebu tak, aby bylo možné v každém časovém úseku určit, které spotřebiče jsou aktuálně připojeny. Existuje mnoho prací z této oblasti, jejich společnou vlastností však je, že slouží především k získávání statistik o spotřebě a její následné optimalizaci, nikoliv ke komunikaci s dalšími prvky domácnosti.

K měření spotřeby existují dva opačné přístupy. První z nich je tzv. ILM (Intrusive load monitoring), zde je měřák umístěn přímo v zásuvce. Druhým je tzv. NILM (Non-intrusive load monitoring), kde je měřák umístěn pro všechny zásuvky v domě nebo místnosti dohromady. Zde je typicky vyšší počet simultánně připojených spotřebičů. V některých článcích [12] je klasifikace ILM vs NILM klasifikována ještě dále, tak jak je uvedeno v tabulce 2.1.

Označení	význam
NILM	jeden měřák pro celý dům
ILM I	měřák pro každý jistič
ILM II	měřák pro rozdvojku/roztrojku
ILM III	měřák pro každou zástrčku

Tabulka 2.1: Granularita umístění měřáku z článku [12].

Přístup NILM byl populární zejména v minulosti. Důvodem byla hlavně dříve velmi vysoká cena chytrých měřáků. Umístil se tedy jeden pro celý dům a algoritmy musely být dostatečně dobré, aby zvládly klasifikaci spolehlivě.

V práci [13] z roku 2015 byl pro toto představen dataset obsahující 15 kategorií spotřebičů jako například pračky, televize, holící strojky, kávovary, ledničky a podobně. Úlohou bylo z jednoho měřáku, kde může být až 15 běžících spotřebičů zároveň, rozpoznat, které jsou aktuálně zapojeny. Protože byl v domě jen jeden měřák, mohl být dražší a chytřejší a měřit více veličin než jen okamžitou spotřebu, což na druhou stranu klasifikaci ulehčovalo. Kromě okamžité spotřeby tak byly k dispozici i časové řady pro:

- reaktivní výkon,
- RMS Proud,
- RMS Napětí,
- fáze napětí relativní k proudu.

Také frekvence měření byla poměrně vysoká a to 10Hz. Autoři porovnávali různé algoritmy pro klasifikaci (skryté Markovovy modely, Gaussovský model směsi, shlukování nejbližších sousedů). Přesnost uvádějí až 93 %. Se stejným datasetem pracuje i článek [5], kde bylo dosaženo přesnosti přes 99 % s neuronovou sítí.

[3] Po registraci je nabídnuta 7denní zkušební verze zdarma.

Tento přístup sice fungoval a klasifikace byla spolehlivá i pro vyšší počet spotřebičů, kvůli použití velmi dlouhého okna však klasifikace probíhala se značným zpožděním. Postup tedy lze bez problému použít například pro statistiky spotřeby a času zapnutí jednotlivých spotřebičů apod. Pro použití tam, kde je třeba rychle reagovat na změnu připojených spotřebičů je však nevhodný.

S klesající cenou chytrých zařízení v posledních letech stoupá obliba přístupu ILM. Klasifikace může být o mnoho rychlejší a spolehlivější. Nevýhodou je nutnost montáže více měřáků, pokud je cílem pokrytí např. celého domu a také nižší frekvence odečtu spotřeby.

Tímto přístupem se zabývá méně prací. Jednou z nich je například [10] z roku 2020. Autoři zde vyzdvihují klesající ceny a možnost nejen mít přehled o spotřebě, ale také možnost ovládání jednotlivých zásuvek. Pro klasifikaci je zde použito kombinace neuronové sítě a SVM. Bylo dosaženo přesnosti přes 92 % na datasetu obsahující tři spotřebiče. Přesnost je tedy nižší než u výše představených prací i přes to, že počet kategorií je o mnoho menší.

Rozdíl je však v délce časového okna pro klasifikaci, které je zde mnohem menší, a to pět minut. Použitý dataset je však velmi malý (pouze 273 vzorků) což se na přesnosti nepochybně negativně projevilo. Obecně lze říct, že výzkumů jako výše popsaný [10] mnoho není, více autorů se ještě stále zabývá přístupem NILM a složitými disagregačními algoritmy pro mnoho kategorií s dlouhými hodinovými okny.

Chytrá zásuvka představená v této práci pracuje jako ILM. Dle tabulky 2.1 se jedná o ILM II. Hlavní předností je velmi vysoká rychlost klasifikace (jednotky sekund), což umožňuje téměř okamžitou reakci na událost připojení spotřebiče, nikoliv pouze zpětné zobrazení statistik o spuštění. Takovou reakcí může být například zasílání notifikace na mobilní telefon při dokončení pracího cyklu, upozornění při zapnutí počítače mimo povolený interval, tedy použití pro rodičovskou kontrolu, nebo třeba zhasnutí světla několik sekund po vypnutí televize, pokud je venku tma.

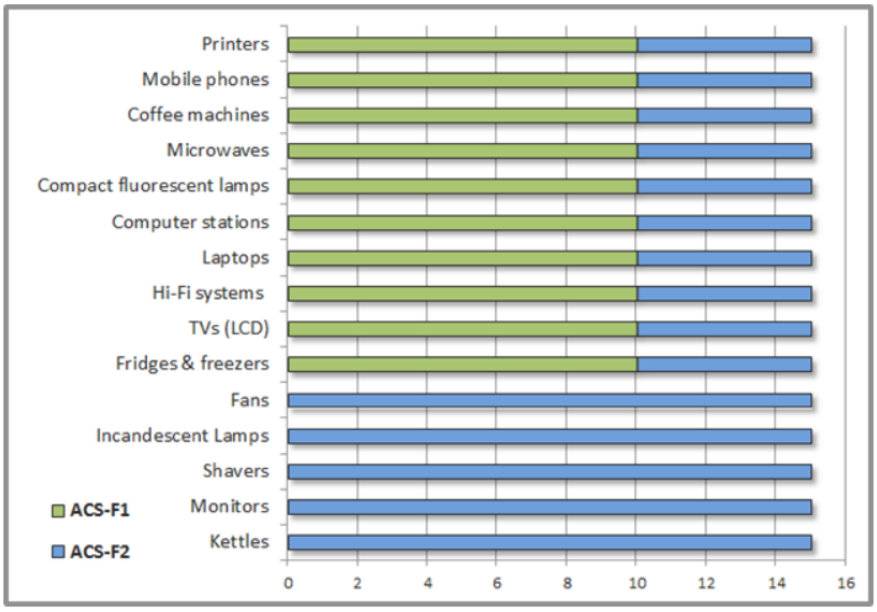
Existující datasety

Jak bylo popsáno výše, datasety zabývající se podobnou problematikou jsou volně k dispozici. Jedná se však o datasety obsahující přes 10 kategorií spotřebičů. Jednotlivé kategorie jsou uvedeny na obrázku 2.3. Pro klasifikaci takového datasetu je potřeba velké neuronové sítě pracující se značným zpožděním. Dalším problémem je, že pokud není dataset aktualizován, nemusí být výsledky příliš dobré. Spotřeby stolních lampiček se díky LED technologii snižují, pro některé přístroje jako například počítač mohou být řádově jiné v závislosti na modelu. V této práci je tedy použitý odlišný přístup, kdy uživatel sice musí pár desítek minut měřit spotřebu, aby se zásuvka mohla spotřebu naučit, na druhou stranu je v principu možné naučit zásuvku rozpoznávat jakýkoliv spotřebič a uživatel není omezen na kategorie, které jsou v datasetu zastoupeny.

2.4 Další možnosti automatizace

Existují i další možnosti, jak umožnit různým zařízením propojení s chytrou domácností. Jedná se zejména o specifická hobby řešení třeba pomocí jednodeskového počítače Arduino s WiFi modulem [9]. Výhodou takových řešení je především nízká cena (pár set korun), velká škála různých modulů (teploměry, vlhkoměry, mikrofony, magnetické senzory atd.) a také velké množství různých návodů a tutoriálů na internetu.

Mezi nevýhody pak patří nutnost alespoň základní znalosti programování, protože uživatel si musí řešení (byť často podle návodu na internetu), zhotovit sám. Pomocí Arduina



Obrázek 2.3: Přehled kategorií v datasetu ACS. Převzato z [5].

lze pak například posílat požadavky do IFTTT nebo Node-RED. Díky tomu lze dosáhnout skutečně plnohodnotného řešení, které může obsahovat i složitá pravidla a logiku. Tato zařízení jsou však obvykle velmi specificky navržena pro svůj účel a pro každý spotřebič je nutné jej značně přizpůsobit. Toto řešení je tedy vhodné především pro hobby nadšence kteří chtějí experimentovat, často měnit funkčnost zařízení a podobně. Na obrázku 2.4 je například chytrý květináč s použitím Arduina, vlhkoměru a LED indikátoru.

2.5 Open source nástroje pro správu domácnosti

V této sekci budou představeny dva open source nástroje pro správu chytré domácnosti. Ty narozdíl od výše zmíněného IFTTT dávají uživateli velkou volnost v tom, co vše lze ovládat. Nevýhodou je však složitější zacházení s těmito systémy.

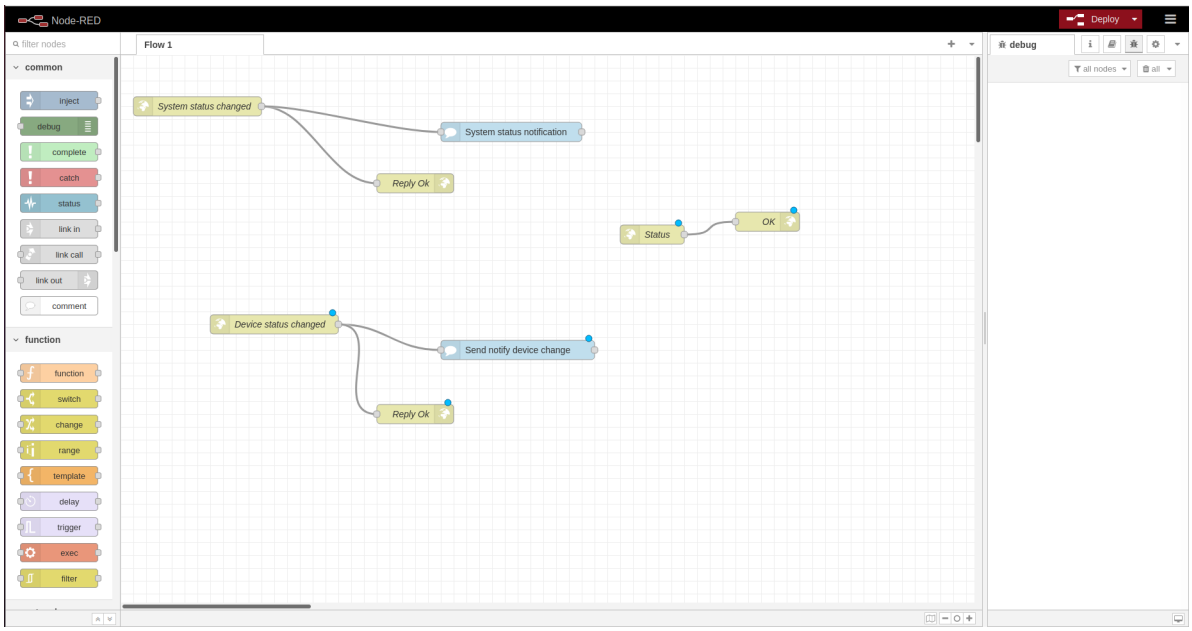
Node-RED

Node-RED je programovací nástroj pro správu pravidel pro chytrá zařízení. Programování zde probíhá vizuálně, pomocí uzlů, které lze propojovat a sestavovat z nich pravidla. Vzhled rozhraní a několik uzlů tvořících pravidlo pro zaslání notifikace po obdržení HTTP požadavku je ilustrováno na obrázku 2.5. Do samotného nástroje lze pak na základě požadované funkčnosti uživatelem doinstalovat zásuvné moduly. Ty jsou tvořeny především komunitou kolem Node-RED a mohou obsahovat jak jednotlivé uzly (nodes) tak celá hotová pravidla (flows). Mezi nejoblíbenější[4] patří třeba uzly pro:

- zveřejňování tweetů,
- zobrazení různých přehledů (dashboardy),
- posílání a příjem jednoduchých emailů,

[4] dle počtu stažení

Obrázek 2.4: Chytrý květináč se senzorem vlhkosti napojeným na Arduino. Převzato z https://www.instructables.com/Smart-Garden-1/



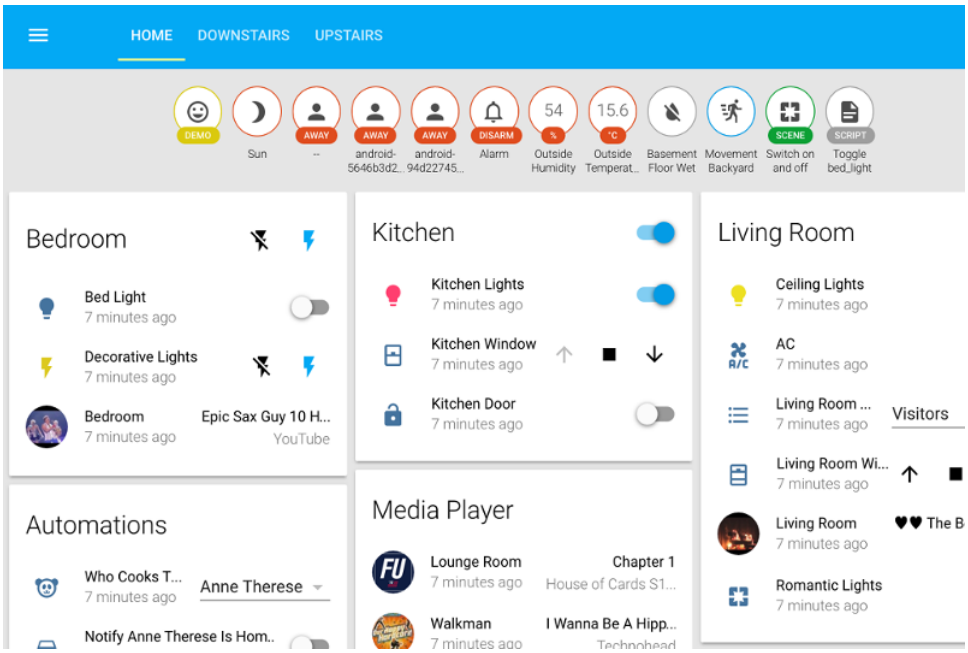
Obrázek 2.5: Rozhraní nástroje Node RED.

- integraci se službami Google,
- posílání a naslouchání HTTP požadavkům,
- komunikaci se sériovými porty,
- zápis a čtení databáze,
- posílání notifikací na mobilní telefon.

Node-RED je open source a může být spuštěn buď lokálně, nebo v cloudu. Také jej lze propojit s výše zmíněnými službami IFTT, Google Home, Amazon Alexa a podobně.

Home Assistant

Home assistant je open source software pro řízení chytré domácnosti vyvíjený od roku 2013. Ovládat jej lze pomocí webového rozhraní, nebo aplikace na iOS a Android. Také umožňuje integraci s hlasovými klienty představenými v sekci 2.3. Home assistant je možné nainstalovat na Windows, macOS i Linux. Také je možné jej hostovat například na Raspberry Pi. Programování zde narozdíl od Node-RED neprobíhá vizuálně, což je také největší rozdíl mezi Node-RED a tímto nástrojem. Jak vypadá dashboard udělaný v Home assistant je ilustrováno na obrázku 2.6. Funkčnost je srovnatelná s Node-RED. Při rozhodování který



Obrázek 2.6: Příklad vzhledu dashboardu pro chytrý dům v Home assistant. Převzato z https://www.berlinger.cz/blog/raspberry-pi-server-pro-senzory.

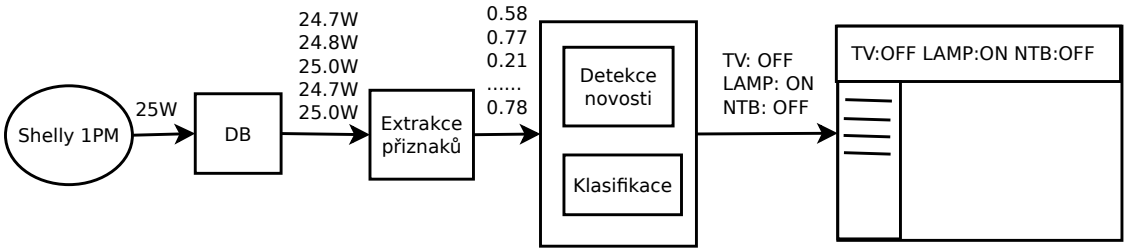
nástroj použít tak bude rozhodovat hlavně osobní preference a názor na vizuální programování. Pro ulehčení začátku práce s Home Assistant jsou na webu[5] k dispozici ukázky některých často používaných procedur (tzv. blueprinty). V této práci byl však především díky možnosti vizuálního programování zvolen nástroj Node-RED.

[5] https://www.home-assistant.io/examples/

Kapitola 3

Návrh řešení

V této kapitole bude popsán obecný návrh systému. Na začátku bude pro lepší orientaci uvedeno blokové schéma. Poté budou uvedeny požadavky na výběr chytrého relé a také to, jak byla sbírána data pro další použití. Bude ilustrováno, jak je možné z naměřených dat udělat dataset vhodný pro další použití. Poté budou obecně popsány metody pro detekci novostí a klasifikaci. V závěru bude představen návrh webového rozhraní pro celý systém a naznačeno, jak je možné komunikovat s dalšími prvky chytré domácnosti. Pro přehlednost je vysokoúrovňové schéma celého systému naznačeno na obrázku 3.1.



Obrázek 3.1: Vysokoúrovňové blokové schéma systému.

3.1 Požadavky na chytré relé

Požadavky při výběru chytrého relé byly především nízká cena realizace, rozumná frekvence odečtu spotřeby, možnost komunikace přes HTTP požadavky. Na základě těchto požadavků byl zvolen modul Shelly 1PM. Ten má následující vlastnosti:

- odečet spotřeby s frekvencí 1 Hz,
- ovládání přes Wi-Fi a možnost HTTP požadavků,
- přehledná dokumentace,
- příznivá ceně (cca 600kč).

Modul lze namontovat přímo do zásuvky, nebo ukrýt do externí krabičky. Rozměry modulu jsou 41 mm x 36 mm x 17 mm. Shelly 1PM také umožňuje odečet aktuální teploty, což při práci nebylo využito. Obrázek 3.2 ukazuje, jak vypadá chytré relé před montáží do zásuvky.

Obrázek 3.2: Shelly 1PM před zapojením.

Pro zařízení Shelly je k dispozici aplikace pro mobilní telefon na Google Play i App Store. V aplikaci lze odečítat aktuální spotřebu a teplotu, díky ukládání do cloudu také zobrazit graf spotřeby. Protože je možné měřáky sdružovat do místností, lze takto agregovat více podobných modulů a mít přehled i o místnostech a celém domě. Existuje i možnost přidání rozvrhu pro zapínání a vypínání v průběhu týdne. Jak aplikace vypadá na telefonu s operačním systémem Android je ukázáno na obrázku 3.3.

3.2 Sběr dat

Sběr dat probíhá pravidelným dotazováním modulu Shelly na aktuální spotřebu pomocí HTTP požadavku. Dotazování probíhá každou sekundu, tedy maximální rychlostí jakou umí Shelly 1PM odečítat spotřebu.

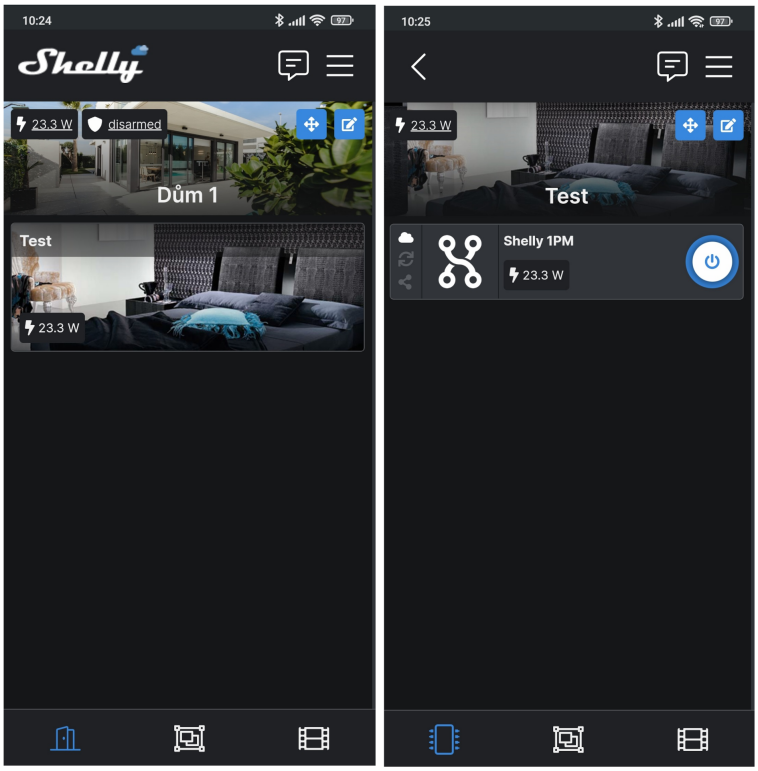
Databáze

Protože systém pracuje výhradně s časovými řadami, je vhodné místo standardní relační databáze použít jinou, vhodnější pro tento typ dat. Takovou databází je například InfluxDB. Pro práci se hodí zejména funkce jako jsou klouzavé průměry, doplňování chybějících hodnot v řadě, interpolace a podobně. Databáze je podrobněji popsána v sekci 4.2.

3.3 Příprava datasetu

Jak bylo uvedeno výše, zásuvka bude rozpoznávat různé kombinace spotřebičů. Aby však mohla být natrénována, je třeba nejprve připravit dataset. V této práci bylo chytré relé připojeno na zásuvku se třemi zástrčkami. Zde představené algoritmy by po drobné modifikaci bylo možné použít pro různé počty zástrček. Jak je však patrné ze vztahu 3.1, se vzrůstajícím počtem zástrček roste počet možných kombinací velmi rychle a lze předpokládat, že přesnost klasifikace při užití vyššího počtu zástrček by byla horší.

$$Kombinace = \sum_{k=1}^{N} \binom{N}{k} = \sum_{k=1}^{N} \frac{N!}{k!(N-k)!} \tag{3.1}$$



Obrázek 3.3: Aplikace pro Shelly zařízení na OS Android.

Bylo by značně nepraktické, pokud by uživatel musel při trénování zkoušet všechny možné kombinace. Proto je požadováno pouze měření jednotlivých spotřebičů a zbylé kombinace jsou dopočítány.

Pro vytvoření kvalitního datasetu a správného natrénování zásuvky je nutné, aby bylo zaznamenáno co nejvíce možných průběhů spotřeby. Například zařízení s konstantní spotřebou 20W není třeba měřit příliš dlouho. U notebooku, jehož spotřeba se velmi liší v závislosti na aktuální činnosti, a navíc možnosti dobíjení, kdy je spotřeba nejvyšší a pouze omezena zdrojem je žádoucí měřit déle. U datasetu představeného v kapitole 5 byl celkový čas měření tří zařízení 2 hodiny a 36 minut. Když jsou k dispozici dílčí spotřeby zařízení, postup vytvoření datasetu je následující:
Nejdříve se vygenerují všechny možné kombinace zapojení spotřebičů. Poté se pro každou kombinaci o počtu spotřebičů N a délce L provede toto:

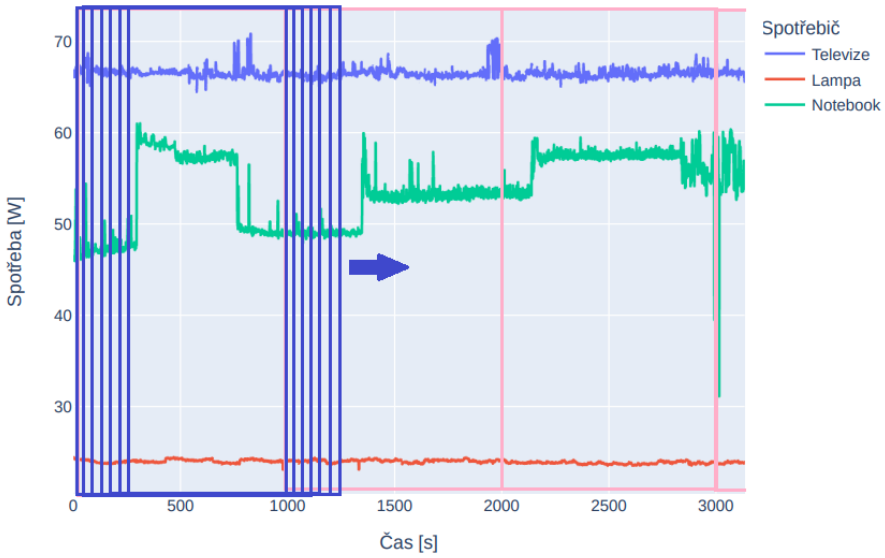
1. pokud je N = 1, pouze se připojí do datasetu (není kombinací),
2. jinak se vygeneruje N náhodných čísel v rozsahu 1 až L,
3. pro každou z N tříd je vybráno náhodné okno s indexem L,
4. odpovídající prvky všech N oken se sečtou a tím se vytvoří pomocné okno opět stejné délky (ilustrováno v tabulce 3.1),
5. pomocné okno se přidá do datasetu.

TV[W]	66.98	67.16	67.08	67.25	67.08
Lampa[W]	24.18	24.20	24.09	24.10	24.07
Součet	↓	↓	↓	↓	↓
TV+Lampa[W]	91.16	91.36	91.17	91.35	91.15

Tabulka 3.1: Uměle vytvořené okno simulující simultánní měření sčítáním jednotlivých tříd.

Po dopočítání je tedy k dispozici M časových řad o délce L_w. Pro klasifikaci signálu je používán výřez okna délky L_o z jeho průběhu. Aby bylo k dispozici více vzorků, nepoužívá se rozdělení signálu na L_w/L_o, ale používá se princip klouzavého okna.

Místo L_w/L_o vzorků je tak k dispozici $L_w - L_o$ vzorků pro každou časovou řadu. Jak je patrné i z obrázku 3.4, vzorků je pak mnohem více.



Obrázek 3.4: Rozdíl mezi pouhým rozřezáním řady a posouváním okna pro výřez.

3.4 Detekce anomálií a klasifikace

V této sekci bude představeno, jak je možné detekovat neznámé spotřebiče. Dále pak jak lze provést klasifikaci připojených zařízení.

Normalizace dat a extrakce příznaků

Jak již bylo uvedeno, systém rozlišuje jednotlivé kombinace spotřebičů dle krátkého okna s průběhem spotřeby. K dispozici je tedy krátká časová řada například pěti hodnot, dle kterých je třeba rozpoznat o kterou kombinaci spotřebičů se jedná. Z té je nutné extrahovat příznaky. Ty, které jsou využité v této práci budou nyní blíže popsány. Mezi jednoduché příznaky patří následující:

- minimum,
- maximum,

- medián.

Dále potom: kvadratický průměr (3.2), energie signálu (3.3), suma rozdílů v řadě (3.4).

$$RMS = \sqrt{\bar{x^2}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} = \sqrt{\frac{X_1^2 + x_2^2 + \cdots + x_n^2}{n}} \tag{3.2}$$

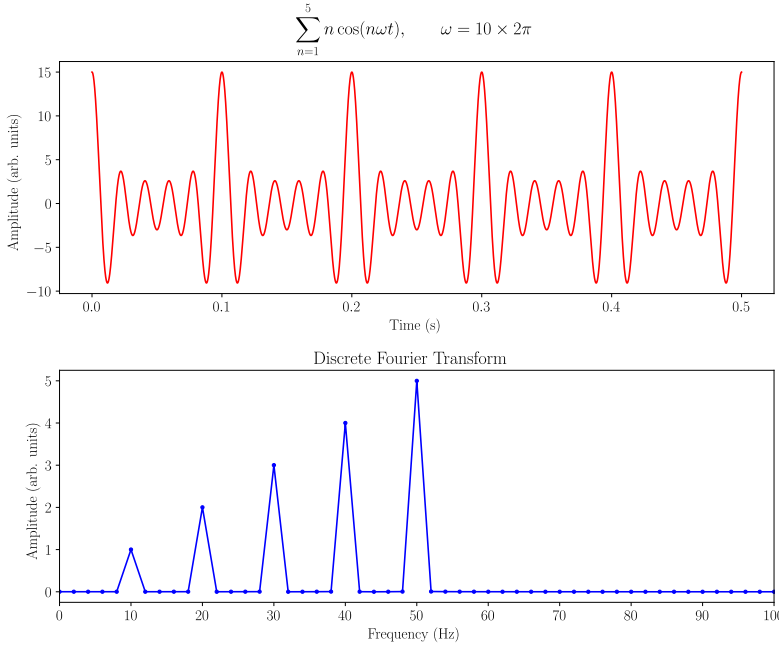
$$E = \sum_{n=0}^{N-1} |x[n]|^2 \tag{3.3}$$

$$S = \sum_{i=1,\ldots,n-1} | \, x_{i+1} - x_i \, | \tag{3.4}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (X_i - \mu)^2} \tag{3.5}$$

Kromě těchto příznaků jsou pak použity i složitější charakteristiky. První z nich jsou koeficienty výsledku rychlé Fourierovy Transformace.

Fourierova transformace (resp. inverzní Fourierova transformace), převádí signál z časového spektra do frekvenčního (resp. z frekvenčního do časového). Tedy dává informaci o tom, z jakých frekvencí se skládá signál. Ilustrace je na obrázku 3.5. Po spočítání rychlé



Obrázek 3.5: Ilustrace efektu diskrétní Fourierovy transformace. Převzato z https://en.wikipedia.org/wiki/Fast_Fourier_transform.

fourierovy transformace lze výsledek zapsat opět jako číselnou řadu. Hodnoty této řady budou odpovídat hodnotám ve výsledku transformace, dle vztahu 3.6.

$$A_k = \sum_{m=0}^{n-1} a_m^{-2\pi i \frac{mk}{n}} \tag{3.6}$$

, kde:
A_k je k-tá hodnota výsledné transformace,
m je hodnota časové řady,
n je počet hodnot v časové řadě.
Podobně jako hodnoty výsledku Fourierovy transformace jsou jako příznaky použity i hodnoty vlnkové transformace. Vztah je vyjádřen rovnicí 3.7.

$$S = \frac{2}{\sqrt{3a}\pi^{\frac{1}{4}}}(1 - \frac{x^2}{a^2})^{(-\frac{x^2}{2a^2})} \tag{3.7}$$

, kde:
a je šířka vlnky,
x je hodnota časové řady.
Poslední příznak je pak vypočítán dle vztahu 3.8. Je počítán dvakrát pro dva různé offsety.

$$S = \frac{1}{n - 2lag} \sum_{i=1}^{n-2lag} x_{i+2\cdot lag} \cdot x_{i+lag} \cdot x_i \tag{3.8}$$

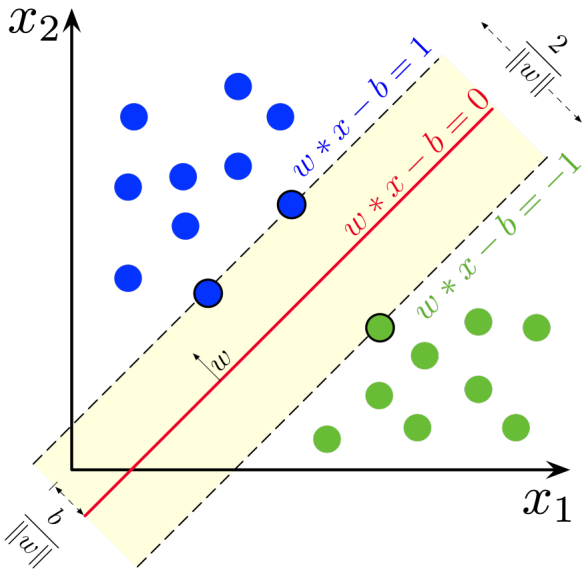
, kde:
$lag \in 1, 2$ je použitý offset,
x je hodnota časové řady.
Z výše uvedených vztahů je patrné, že z nich mohou vycházet (a typicky vychází) řádově velmi odlišné hodnoty. Všechny hodnoty jsou tedy ještě normalizovány do intervalu <0,1>.

Detekce novosti

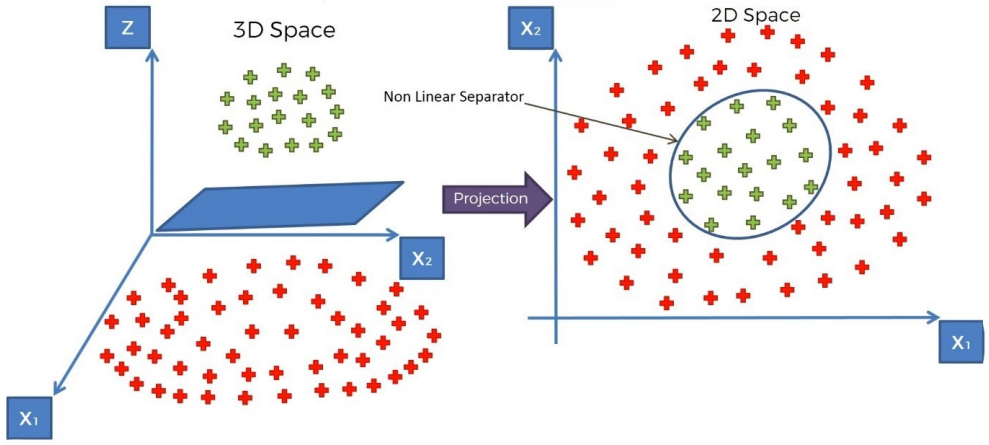
Ještě před samotnou klasifikací je vhodné provést detekci novosti (novelty detection). Ta je podobná detekci anomálií. Rozdíl je následující. Při detekci anomálií se předpokládá, že daný dataset obsahuje nežádoucí anomálie. Ty se pak algoritmus snaží na základě ostatních (většinových) hodnot v datasetu detekovat. Při detekci novosti se předpokládá, že dataset obsahuje pouze validní hodnoty a při predikci algoritmus určuje, zda je dotazovaná hodnota dostatečně podobná hodnotám ze kterých se skládá dataset na kterém byl detektor trénován. Protože lze předpokládat, že hodnoty naměřené uživatelem při trénování jsou validní, zde se jedná o detekci novosti.

K detekci byla použita metoda podpůrných vektorů. Ta byla poprvé představena v roce 1992 [2]. Jedná se o klasifikátor provádějící binární klasifikaci. Ten rozděluje data reprezentované vektory nadrovinou do dvou tříd. Název podpůrných vektorů odkazuje na vektory reprezentující hraniční data. Ty mezi sebou tvoří pás, jehož šířku se klasifikátor snaží maximalizovat. Ilustrováno na obrázku 3.6. Důležitou součástí SVM jsou tzv. jádrové transformace. Ty umožňují této metodě transformovat data do prostoru jiné (typicky vyšší) dimenze. Úloha, která by byla lineárně neseparovatelná se pak stane lineárně separovatelnou, jak je naznačeno na obrázku 3.7. Lze použít různé jádrové funkce, jednodušší funkce nemusí být schopné data dobře oddělit, ale bývají rychlejší pro trénování. Vliv výběru funkce na klasifikaci je ilustrován na obrázku 3.8.

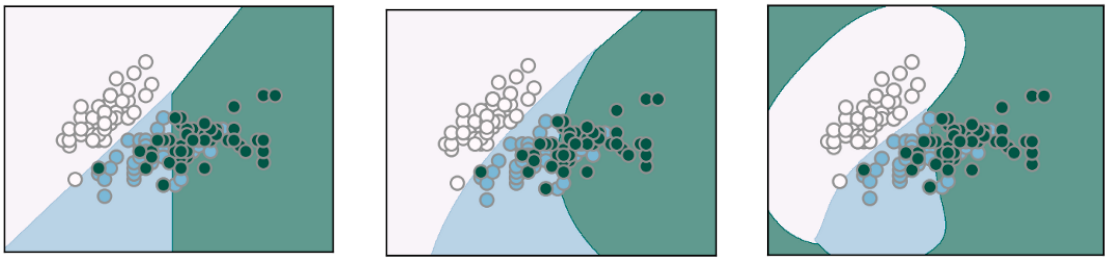
SVM je metoda která v principu pracuje pouze se dvěma třídami. Pokud je použita pro klasifikaci do více tříd, je nutné natrénovat klasifikátorů více.



Obrázek 3.6: Hraniční pásmo (žlutě) s podpůrnými vektory(zakroužkované). Převzato z https://en.wikipedia.org/wiki/Support-vector_machine.



Obrázek 3.7: Oddělení lineárně neseparovatelných dat transformací do vyšší dimenze. Převzato z https://medium.com/@suvigya2001/the-gaussian-rbf-kernel-in-non-linear-svm-2fb1c822aae0.



(a) Linearání jádro (b) Polynomiální jádro (c) RBF jádro

Obrázek 3.8: Různé typy jádrových funkcí pro použití s SVM. Převzato z https://towardsdatascience.com/multiclass-classification-with-support-vector-machines-svm-kernel-trick-kernel-functions-f9d5377d6f02.

Klasifikace připojených spotřebičů

Klasifikace je úloha, při které se vstupnímu vektoru s příznaky přiřadí výstupní vektor s pravděpodobnostmi příslušností k jednotlivým třídám. V této práci je pro klasifikaci použita jednoduchá neuronová síť napsaná ve frameworku Pytorch.

Neuronová síť je výpočetní model inspirovaný lidským mozkem. Skládá se z několika neuronů, které jsou sdružovány do jednotlivých vrstev sítě. Neuronových sítí existuje mnoho typů, přičemž každý typ je vhodný na řešení jiného problému. V dnešní době jsou populární hlavně 2D konvoluční neuronové sítě pro zpracování obrazu. Konvoluční proto, že obsahují vrstvy, které počítají nad vstupními daty operaci konvoluce. Učením sítě se pak rozumí nastavování hodnot v konvolučním filtru.

Dalším typem jsou plně propojené sítě. Plně propojená síť je použitá v této práci. Učením se zde nastavují váhy propojení neuronů. Jeden neuron je pak reprezentován operací sumarizace vstupů. Těmi jsou hodnoty z předchozí vrstvy vynásobené váhou propojení a někdy také tzv. bias, který reprezentuje posun výsledné hodnoty. Výstup jednoho neuronu lze pak vyjádřit vztahem 3.9.

$$y = b + \sum_{i=0}^{N} w_i * x_i \tag{3.9}$$

, kde:
b je bias,
N je počet neuronů v předchozí vrstvě,
i je index neuronu v předchozí vrstvě,
x je hodnota výstupu daného neuronu,
w je naučená váha spojení.

Jeden neuron je však schopen kvůli linearitě vztahu 3.9 řešit pouze lineární problémy. A přesto, že lze jedním neuronem například řešit lineární rovnice [15], k řešení složitějších problémů nestačí.

Aby to bylo možné, je nutné mít k dispozici nelineární funkci. Na výstup samotného neuronu tedy ještě aplikována tzv. aktivační funkce. Celkový výstup neuronu je tedy ještě upraven na 3.10.

$$y = F(b + \sum_{i=0}^{N} w_i * x_i) \tag{3.10}$$

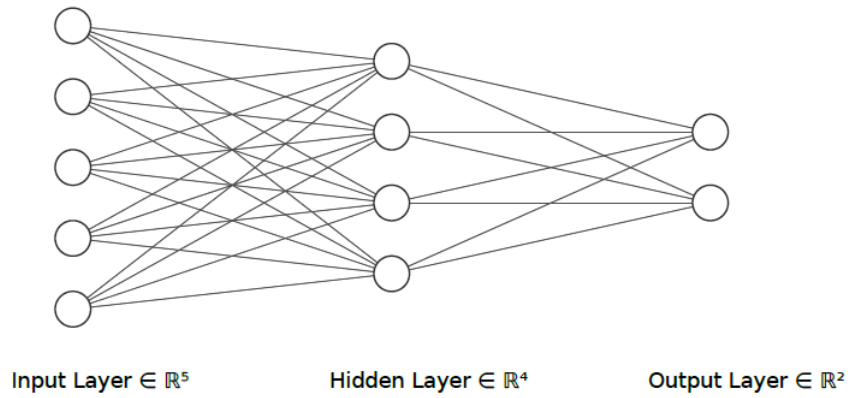
, kde:
b je bias,
N je počet neuronů v předchozí vrstvě,
i je index neuronu v předchozí vrstvě,
x je hodnota výstupu daného neuronu,
w je naučená váha spojení,
F je aktivační funkce.

Používaných aktivačních funkcí je mnoho. Mezi nejpoužívanější patří ReLU (vztah 3.11) pro skryté vrstvy, u výstupní vrstvy pak záleží na typu úlohy. Pro klasifikaci se pak často používá Softmax, nebo Sigmoida (vztah 3.12). Funkce použité v této práci jsou vykresleny na obrázku 3.10.

$$ReLU(x) = max(0, x) \tag{3.11}$$

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.12}$$

Jak vypadá schéma jednoduché plně propojené neuronové sítě je ukázáno na obrázku 3.9. Ve vstupní vrstvě je 5 neuronů. To značí, že vstupem do sítě bude vektor obsahující pět



Obrázek 3.9: Schéma architektury velmi jednoduché plně propojené neuronové sítě.

hodnot. V další (tzv. skryté) vrstvě jsou 4 neurony a v poslední 2. Taková síť by mohla sloužit třeba k (velmi) jednoduché binární klasifikaci. Neuronové sítě pro řešení komplexních problémů však mohou být daleko složitější. Počet vrstev může být i několik desítek a každá může obsahovat stovky neuronů. Pro klasifikaci 12 hodnot to však není potřeba, navíc by se síť hůře učila, kvůli vyššímu počtu parametrů takového modelu.

Jak bylo uvedeno výše, učení neuronové sítě znamená změnu vah u propojení mezi neurony. Z matematického hlediska jde o hledání minima funkce. Tato funkce se nazývá ztrátová. Stejně jako aktivačních funkcí je ztrátových funkcí mnoho a typ použité ztrátové funkce se odvíjí od řešeného problému. Typickými ztrátovými funkcemi mohou být například křížová entropie (cross entropy) pro klasifikaci, střední kvadratická chyba (mean squared error) pro regresi a další. V této práci je použita binární křížová entropie (binary cross entropy), jejíž rovnice je 3.13. Pro minimalizaci této funkce, a tedy učení neuronových sítí se používá algoritmus zpětné propagace chyby (backpropagation), představený poprvé už v roce 1986 [14].

Algoritmus backpropagation, neboli zpětné šíření chyby se tedy používá pro změnu vah v neuronových sítích při procesu učení. Funguje následovně:

1. provede se dopředný průchod sítí s trénovacími daty,
2. dle ztrátové funkce se ohodnotí správnost výstupu sítě,
3. vypočítá se gradient a provede se aktualizace vah.

V posledním kroku algoritmu se postupuje od výstupní vrstvy směrem ke vstupní. Odtud tedy název zpětné šíření chyby.

Síť použitá v této práci je podobná té z obrázku 3.9. Ve vstupní vrstvě je 12 neuronů, každý pro jeden příznak. Další (skryté) vrstvy obsahují 50 a 30 neuronů. Kromě poslední vrstvy využívají neurony aktivační funkci ReLU. V poslední vrstvě se počet neuronů mění v závislosti na počtu zařízení pro které je aktuálně síť natrénována od 1 do 3. V poslední vrstvě je použita aktivační funkce sigmoida. Sigmoida je aktivační funkce, kterou lze vyjádřit vztahem 3.12. Sigmoida je společně s aktivační funkcí softmax často používaná při klasifikaci ve výstupní vrstvě. Při úloze klasifikace se pak často v poslední vrstvě užívá právě softmax. Suma výstupů je pak rovná 1 a hodnoty výstupů určují pravděpodobnosti příslušnosti do jednotlivých kategorií. Zde je toto zbytečné. Pokud se počet výstupních neuronů rovná počtu spotřebičů a každý výstup může nabývat hodnoty 0 až 1, je klasifikace jednodušší. Například kombinaci prvních dvou spotřebičů může značit vektor [1,1,0] místo vektoru o sedmi složkách, kde by byla jedna složka rovna 1, která určuje pravděpodobnosti příslušností do jednotlivých kategorií.

Jako ztrátová funkce byla použita Binary cross entropy, resp. její varianta pro klasifikaci do více tříd. Vztah je uveden v rovnici 3.13.

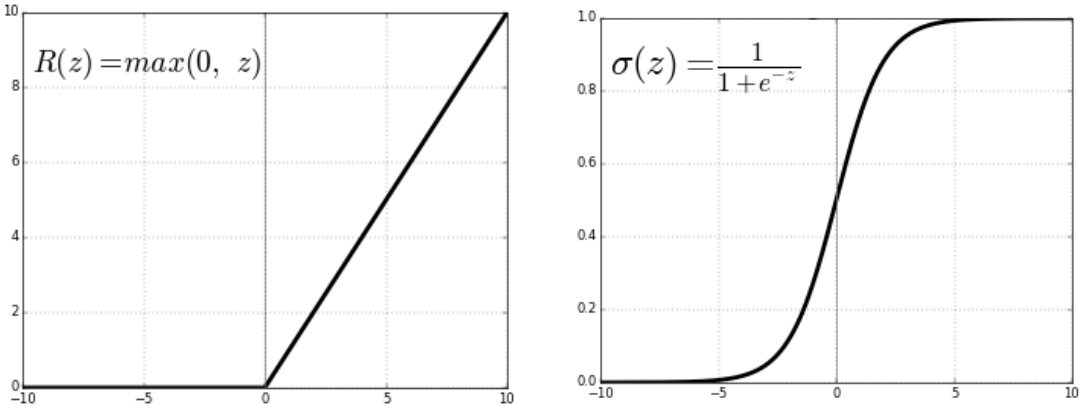
$$BCE = -\sum_{c=1}^{M} y_{o,c} \ln(p_{o,c}) \tag{3.13}$$

kde:
M je počet tříd,
y indikátor, zda byl vzorek o správně klasifikován do třídy c,
p pravděpodobnost, že vzorek o patří do třídy c.
Pro ilustraci jsou uvedené aktivační funkce vykresleny na obrázku 3.10.

3.5 Hystereze klasifikace

Jak bylo popsáno výše, systém klasifikuje připojené spotřebiče jednou za sekundu pomocí okna o délce 5 sekund. Tato hodnota byla zvolena empiricky, aby klasifikace probíhala s rozumnou přesností (přes 99 %) a časové okno bylo co nejkratší. Klasifikátor však nemusí vždy vrátit správný výsledek, nebo může spotřebu vyhodnotit jako neznámé zařízení. To se může stát typicky při připojování nebo odpojování spotřebičů, kdy je při vyčítání hodnot z měřáku vrácena hodnota někde mezi dvěma správnými úrovněmi. K podobnému jevu může také dojít i přirozeně například pokud uživatel pracuje se spotřebičem, kterému nějakou dobu trvá, než se zapne. Jako efektivní prevenci před výše uvedenému lze použít hysterezi klasifikace. Pro skutečnou změnu stavu systému by bylo možné vyžadovat, aby se shodovalo posledních N vektorů klasifikací.

Na základě experimentů byla hodnota N nastavena na 5. Ilustrace změny je uvedena v tabulce 3.2.



(a) ReLU (b) Sigmoida

Obrázek 3.10: Aktivační funkce použité ve vrstvách neuronové sítě. Převzato z https://www.researchgate.net/figure/Sigmoid-and-ReLU-functions-In-order-to-optimize-the-model-forward-propagation-and-back_fig3_352419028.

Čas	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Výsledek klasifikace	1	1	1	1	1	2	3	3	3	3	3	3	3	3	3	3
Skutečný stav systému	1	1	1	1	1	1	1	1	1	1	3	3	3	3	3	3

Tabulka 3.2: Změna stavu systému dle výsledku klasifikace

Kromě snížení chybovosti při připojování a odpojování spotřebičů je také snížena chybovost při klasifikaci obecně, setrvačností se vyfiltrují občasné chybné klasifikace.

Cenou za toto řešení je možné zpomalení klasifikace až o 5 sekund. Celkové zpoždění systému tak v nejhorším případě může být až 10 sekund, což je však stále řádově rychlejší než všechna řešení představena v kapitole 2.3.

3.6 Webové rozhraní

Dle požadavků z kapitoly 1 by mělo uživatelské rozhraní umožňovat přehledně zobrazit následující:

- stav systému (zda je možné komunikovat s databází, zásuvkou),
- aktuální spotřeba,
- aktuálně zapnuté spotřebiče,
- zpětné statistiky o spotřebě,
- zpětné statistiky o zapnutých spotřebičích.

Kromě toho je také nutné umožnit uživateli různá nastavení systému a správu pravidel v Node-RED.

Systém by měl obsahovat menu (například postranní panel), ze kterého by mělo být umožněno se dostat na jednotlivé stránky. Dle účelu by tedy mohlo být navrženo několik stránek.

Aktuální stav systému

Přehled o aktuálním stavu systému je žádoucí zobrazovat na všech obrazovkách. Není důvod tyto informace skrývat například při prohlížení statistik. Informace o dostupnosti databáze a zásuvky by tedy bylo vhodné zobrazovat například ve stavové liště, která by byla dostupná na všech stránkách. Stejně tak by v liště mohly být informace o aktuální spotřebě a zapnutých spotřebičích.

Zpětné statistiky

Zobrazení statistik může probíhat například vynesením jednotlivých hodnot do dvou grafů. První graf může obsahovat průběh spotřeby a druhý informaci o tom, kdy byl který spotřebič zapnutý. Kromě těchto grafů by na stránce mělo být možné měnit časové období za které se statistiky budou zobrazovat.

Konfigurace systému

Systém musí umožnit uživateli alespoň základní konfiguraci. Musí být možné nastavit IP adresy a porty pro zásuvku, databázi, Node-RED a podobně. Kromě toho nelze spoléhat na to, že při klasifikaci bude vždy ideální jedna hodnota prahu ať už při detekci novosti, nebo poté klasifikaci samotné. Kromě výše uvedených adres by tedy mělo být také možné jednoduše nastavit tyto prahy.

Trénování modelu

Trénování modelu sice souvisí s konfigurací systému, přehlednější však bude, když bude probíhat na samostatné stránce. Ta musí umožnit uživateli vybrat soubor s trénovacími daty, nebo specifikovat, jak velký časový úsek se má načíst z databáze. Poté musí být vykreslena spotřeba, aby mohl uživatel na základě vykreslené spotřeby specifikovat, kdy bylo které zařízení zapnuto. Když bude toto hotovo, je možné začít trénovat model.

Node-RED

Protože je Node-RED samostatným nástrojem, bylo by možné pouze odkazovat na stránku, kde běží jeho uživatelské rozhraní. Bylo by nicméně vhodné, aby se otevíralo v novém okně.

3.7 Začlenění do chytré domácnosti

Další komunikace by mohla být realizována pomocí odeslání požadavků do systému Node-RED, kde by se následně zpracovávaly. Pokud by bylo Node-RED k dispozici, podle toho, jak si uživatel nastaví systém by bylo možné posílat buď všechny změny stavu, tj. jak změna připojených spotřebičů tak změna stavu systému (nedostupná databáze, vypnutá zásuvka), nebo pouze některé.

Kapitola 4

Realizace řešení

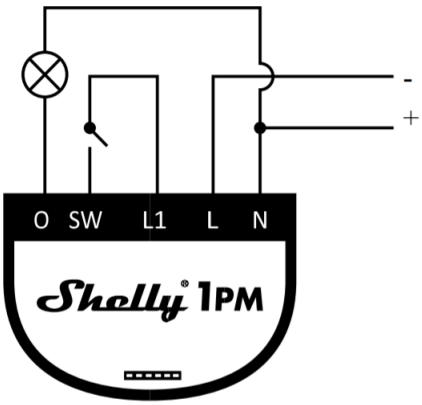
V této kapitole budou představeny konkrétní nástroje a metody pro realizaci návrhu z kapitoly 3. Nejdříve bude popsáno, jak byl modul Shelly zapojen a jak vlastně řešení vypadá. Dále bude popsáno, jak byla realizována komunikace s databází za použití InfluxDB. Bude představen nástroj pro extrakci příznaků z časové řady bez nutnosti programování daných rovnic. Dále budou popsány knihovny Scikit-learn a Pytorch a také to, jak v nich implementovat výše uvedený návrh. Nakonec bude popsán webový framework Flask a realizace webového rozhraní pomocí něj v jazyce Python.

4.1 HW realizace

V této sekci bude popsáno, jak byl modul zapojen a ukázáno výsledné HW řešení.

Zapojení chytrého relé

Samotné zapojení modulu bylo provedeno dle schématu na obrázku 4.1. Význam zkratek je zde následující:

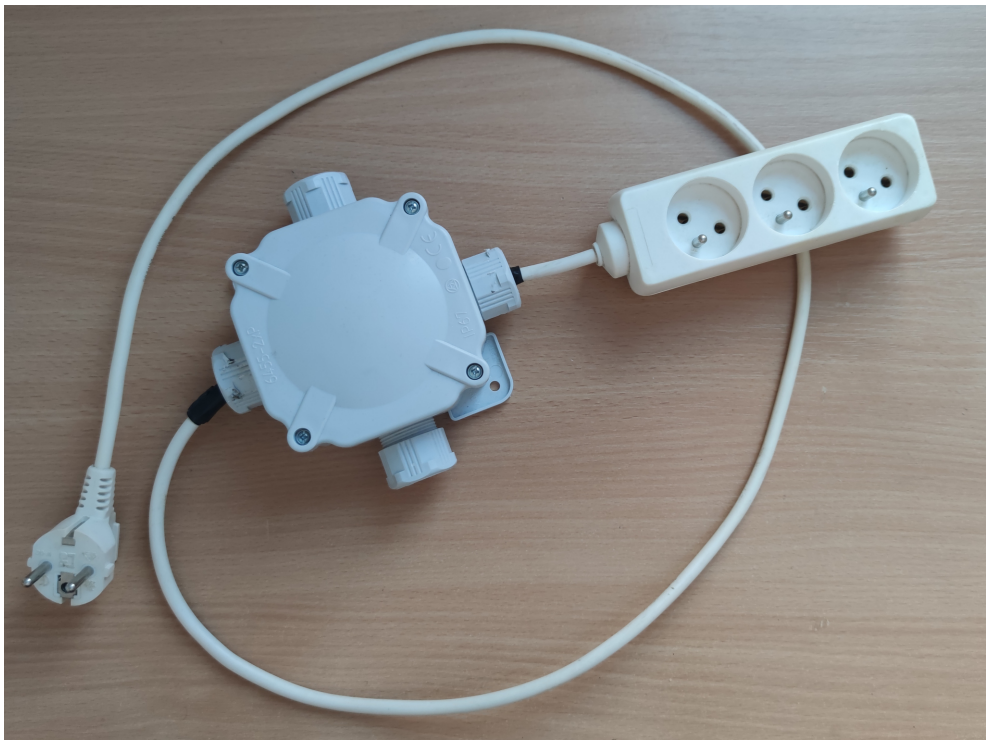


Obrázek 4.1: Schéma zapojení Shelly 1PM.

- O = spínaný výstup,
- N = nulový vodič,

- L = fáze napájení,
- SW, L1 = kontakty pro manuální vypínač (nepoužito).

Aby se se zásuvkou dalo lépe manipulovat pro potřeby sbírání dat, v rámci této práce byl modul Shelly použit v kombinaci s prodlužovací šňůrou se třemi zástrčkami. Výsledek je ukázán na obrázku 4.2.



Obrázek 4.2: HW realizace chytré zásuvky.

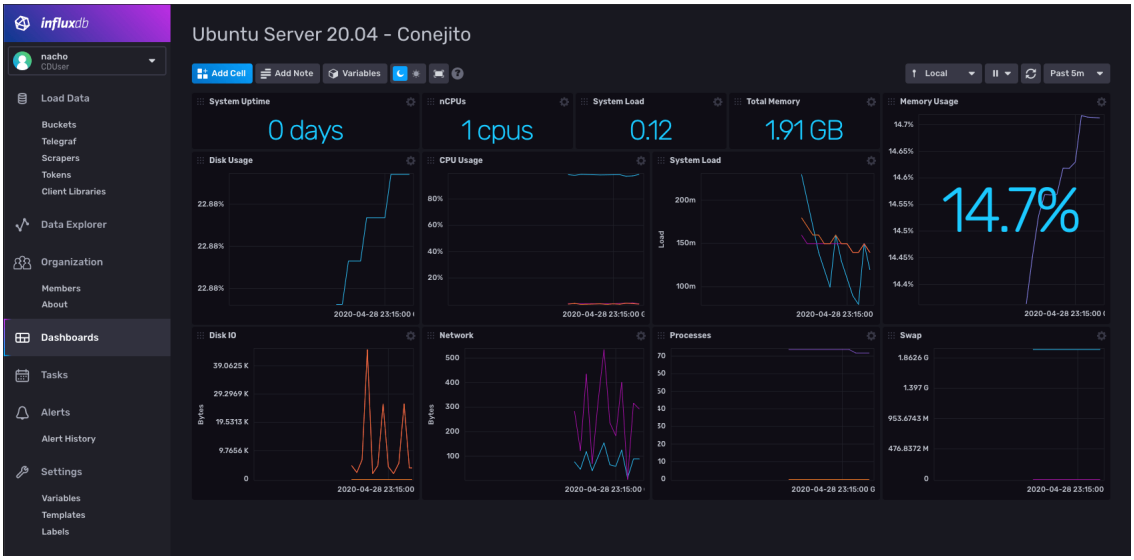
4.2 Komunikace s databází

Jak bylo naznačeno v sekci 3.2, jako databáze byla zvolena taková, která usnadňuje práci s časovými řadami. Zde bude popsáno, jak probíhá práce s takovou databází.

InfluxDB

InfluxDB je open-source databáze. Kromě zápisu a čtení dat také podporuje integraci s nástrojem Graphite[1], díky kterému je možné data přehledně vizualizovat do grafů. K přístupu do databáze byl použit InfluxDb client pro Python.
Protože má databáze i webové rozhraní, ovládání je velmi jednoduché. Rozhraní je ukázáno na obrázku 4.3. Databáze standardně běží na portu 8086. Architektura databáze se od běžných relačních databázi velmi liší. V InfluxDB se nepracuje s tabulkami, primárními a cizími klíči a podobně. Časová řada se ukládá do tzv. Bucketu. Každý bucket má tzv. retention period, což je doba, po kterou jsou data v databázi uchována. Data starší, než

[1] https://graphite-api.readthedocs.io/en/latest



Obrázek 4.3: Webové rozhraní InfluxDB.

nadefinovaná hodnota se automaticky mažou. Každý bucket patří nějaké organizaci. Ta sdružuje uživatele, kteří pracují s jednotlivými buckety. Vzniká zde tedy hierarchie, kdy je možné smazat třeba celou organizaci a všechna s ní související data. Odlišně od běžných databází probíhá i čtení a zápis dat.

Čtení z databáze

Čtení z databáze probíhá pomocí metody query objektu query_api. Jednoduchý dotaz pro zobrazení hodnot spotřeby za poslední hodinu vypadá například takto:
`from(bucket:"shelly") |> range(start: '-1h')`.
Obzvlášť pro delší časové úseky a některé spotřebiče je však žádoucí použít spíše dotaz za použití klouzavého průměru. Dotaz na zobrazení spotřeby zpětně za 1h bez a s použitím klouzavého průměru jsou ilustrovány na obrázku 4.4.

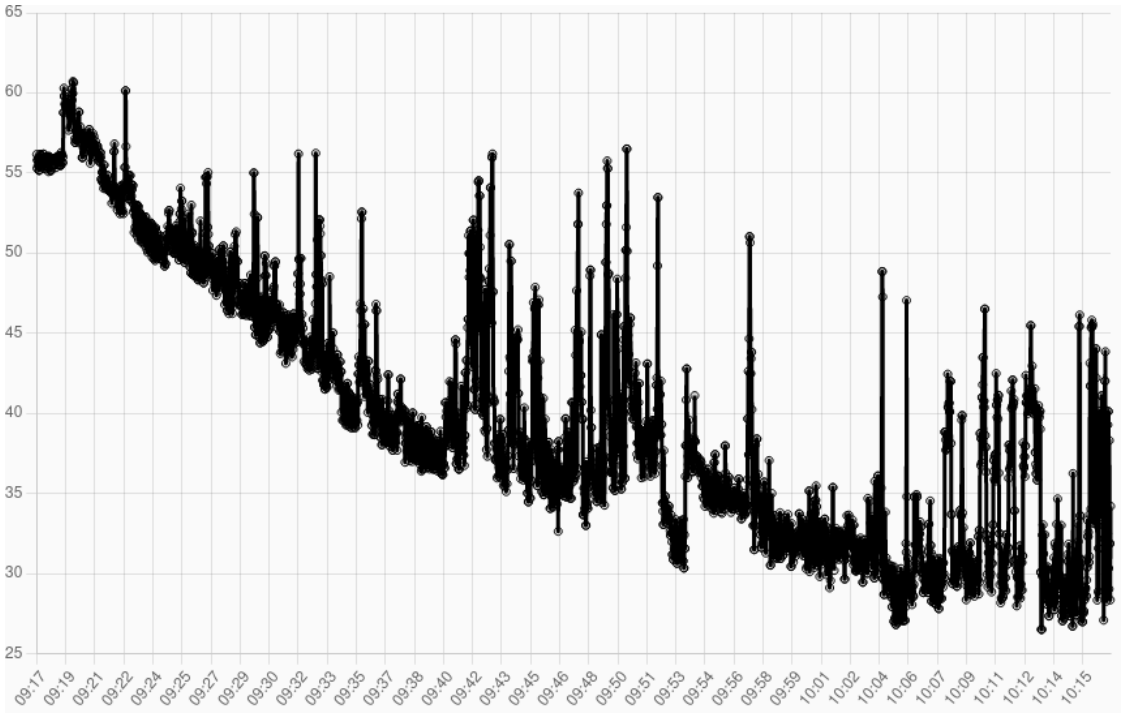
Na obrázku 4.4 nahoře je vidět, že takový dotaz vrací přesné hodnoty. Ty však po vykreslení nejsou příliš dobře patrné, a proto je vhodnější použít složitější dotaz za použití klouzavého průměru a agregace po jedné minutě. Ten je vypadá následovně:

```
from(bucket:"shelly") |> range(start: -'1d')\\|>
filter(fn:(r) => r.\_measurement == "measurement")\\|>
filter(fn: (r) => r["\_field"] == "power")\\|>
aggregateWindow(every: 1m, fn: mean,createEmpty: true)')
```

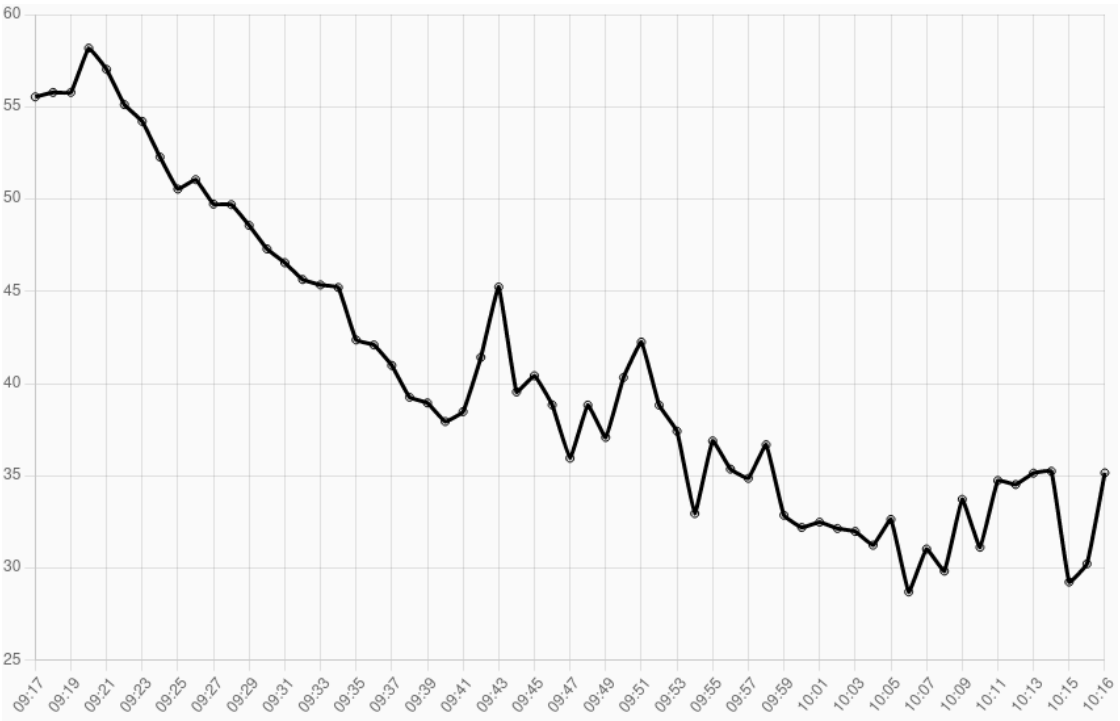
Zde je kromě klouzavého průměru s agregací po jedné minutě také využito funkce createEmpty, která doplní do výsledku chybějící hodnoty. Pokud hodnota v daném bodě chybí, je místo ní vložena hodnota 0.

Zápis do databáze

Zápis databáze probíhá pomocí metody write objektu write_api. Do databáze se zapisují tzv. body. Bod je nejprve třeba vytvořit jako objekt. Poté je možné jej zapsat:



(a) Spotřeba notebooku bez použití klouzavého průměru



(b) Spotřeba notebooku s použitím klouzavého průměru

Obrázek 4.4: Rozdíl při použití klouzavého průměru pro vykreslování spotřeby.

`p = Point("classif").field("dev0", status['devices']['dev0']['status'])`
Pak je pomocí výše zmíněné metody možné jej zapsat do databáze:
`write_api.write(bucket="Shelly", record=p)` .
To je nutné provést pro každé zařízení a spotřebu.

4.3 Extrakce příznaků a normalizace

V této sekci budou nejdříve popsány použité knihovny pro extrakci příznaků a poté, jak pomocí nich byl návrh z kapitoly 3 implementován.

Scikit-learn

Scikit-learn je balíček, který obsahuje kompletní sadu nástrojů pro extrakci příznaků, klasifikaci, regresi, detekci anomálií a podobně.
V této práci je použit na mnoha místech, například pro normalizaci příznaků, detekci novosti (SVM) a v průběhu vývoje byl také používán pro výběr nejvíce relevantních příznaků pomocí metody chí-kvadrát. Kromě toho je v něm možné namodelovat i jednoduchou neuronovou síť. V této práci je k tomu však použita pokročilejší knihovna PyTorch, která bude představena dále.

TSFRESH

TSFRESH[2] je balíček pro programovací jazyk Python, který automaticky počítá velké množství příznaků z časové řady. Jde řádově o stovky příznaků. Některé z nich se počítají pouze pro delší časové řady (například delší než 60 vzorků). TSFRESH počítá velmi různorodé příznaky od jednoduchých veličin jako jsou průměr, maximální hodnota, standardní odchylka až po energie signálu, různé transformace a korelace mezi rozloženími. Podrobnosti o extrahovaných příznacích lze najít na webu[3]. Po vypočítání příznaků umožňuje také automatické zahození nevalidních hodnot.

Primárním účelem balíčku TSFRESH není počítání příznaků z mnoha časových řad o velmi krátké délce, ale spíše výpočet velkého počtu charakteristik pro dlouhé časové řady. Z tohoto důvodu je velká část charakteristik, které TSFRESH počítá pro tuto práci nepoužitelná. TSFRESH nicméně umožňuje automaticky vypočítat příznaky popsané v sekci 3.4, bez nutnosti programování rovnic. Vzhledem k množství příznaků, které jsou počítány pro jednu časovou řadu je nutné také pamatovat na zpoždění výpočtu. Kromě toho, že příznaky uvedené v 3.4 byly určeny jako nejlépe charakterizující, výpočet je asi 4x rychlejší než při výpočtu všech příznaků. U některých také mohou nastávat problémy s hodnotami nekonečna. To u vybraných příznaků nastat nemůže.

Extrakce příznaků

Před samotnou extrakcí je nutné převést naměřená data do formátu, který požaduje na vstupu knihovna TSFRESH. Ten je naznačen v tabulce 4.1 pro dvě časové řady o délce 5 vzorků. Extrakce příznaků probíhá zavoláním metody `extract_features()`. Ta kromě datového rámce jako argument přijímá ještě slovník se seznamem příznaků, které se mají počítat. Slovník může vypadat například takto:

[2] https://tsfresh.readthedocs.io/en/latest/
[3] https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html

```
default_fc_parameters = {
maximum: None,
mediam: None,
fft_coefficient: [{"coeff": 0, "attr": "abs"}]
}
```

None zde značí, že příznak nemá parametry.

ID řady	čas vzorkování	hodnota
0	0	24.21
0	1	24.11
0	2	24.19
0	3	24.19
0	4	24.36
0	5	24.39
1	0	24.27
1	1	24.30
1	2	24.22
1	3	24.19
1	4	24.05
1	5	23.98

Tabulka 4.1: Formát datového rámce pro extrakci příznaků.

Normalizace příznaků

Pro další práci s vypočítanými příznaky je vhodné je normalizovat do rozsahu <0,1>, což jsou hodnoty vhodné pro vstup do SVM a neuronové sítě.
K normalizaci je použita metoda `fit_transform()` objektu MinMaxScaler z knihovny Scikit-learn. Po zavolání metody se objekt naučí, jak příznaky normalizovat a normalizaci také hned provede. Pro pozdější normalizaci je také pomocí knihovny pickle uložen na disk. Zavoláním metody `transform()` na příznakový vektor se pak při klasifikaci transformují vektory každou sekundu pro klasifikaci.

4.4 Detekce neznámých zařízení

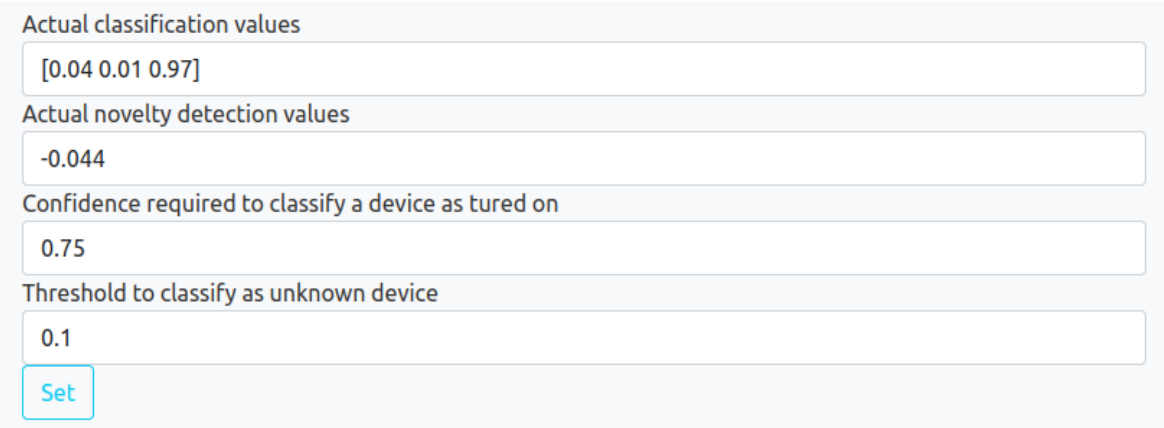
V předchozí sekci bylo popsáno, jak byla implementována extrakce příznaků. Nyní bude popsáno, jak pomocí nich probíhá detekce neznámých zařízení.

Klasifikace pomocí SVM

Jak probíhá detekce novosti pomocí SVM bylo popsáno v sekci 3.4. Implementace SVM je k dispozici v knihovně Scikit-learn. Trénování zde probíhá pouhým zavoláním metody `fit()` na vstupní data. Stejně jako objekt pro škálování je i objekt SVM po natrénování uložen na disk. Každou sekundu se pak provádí klasifikace vypočítaných příznaků. SVM pro pouze jednu třídu nelze dotazovat na pravděpodobnost příslušnosti do třídy. Metoda `predict()` zde nevrací pravděpodobnost příslušnosti do třídy, ale vzdálenost vzorku od dělící roviny. Ta je kladná pro vzorek, který do třídy náleží a záporná pro novost. Tedy dle vztahu 4.1.

$$V(x) = \begin{cases} O & \text{pro } D(x) < 0 \\ I & \text{pro } D(x) \geq 0 \end{cases} \tag{4.1}$$

, kde:
D je vzdálenost od dělící nadroviny,
x je vektor příznaků z časové řady,
O značí novost (outlier),
I značí vzorek blízký datasetu (inlier).
To může být problematické pro uživatele při volení správného prahu. Jak je však patrné z obrázku 4.5, na obrazovce s nastavením jsou vypisovány aktuální hodnoty, které model vrací. Uživatel tak může rychle vidět, která hodnota bude vracet nejlepší výsledky. Pokud



Obrázek 4.5: Formulář pro nastavení prahu klasifikace a detekci novosti.

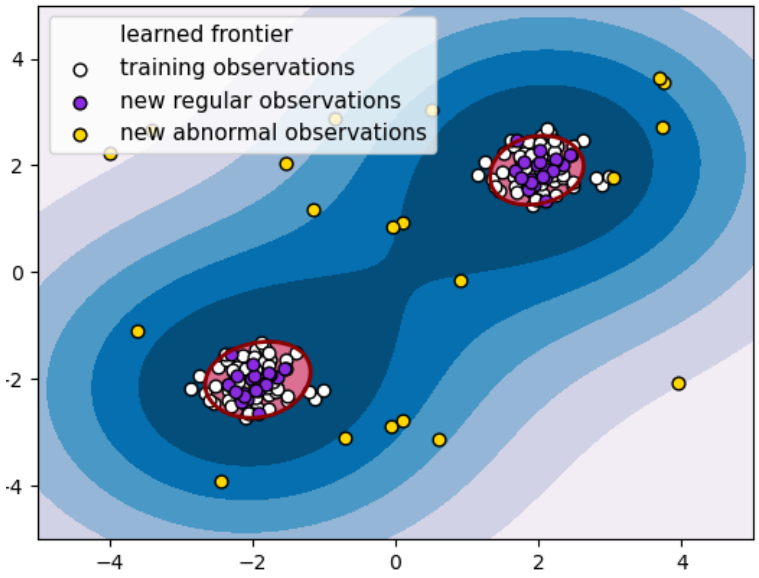
systém detekuje novosti chybně, uživatel může na základě zobrazovaných hodnot jednoduše nastavit správný práh. Jak je patrné z obrázku 4.6, detekce novostí je schopná správně klasifikovat příznaky které jsou v prostoru daleko od těch na kterých se klasifikátor trénoval. Tento klasifikátor však není schopen dobře zachytit příznaky které jsou „někde mezi“. SVM je zde tedy použito především pro detekci extrémních hodnot. Nejistota mezi jednotlivými třídami je pak řešena prahem při klasifikaci zařízení, jak je popsáno v 4.5.

4.5 Klasifikace připojených zařízení

V této sekci bude popsána již samotná klasifikace připojených zařízení do zásuvky za použití neuronové sítě.

Pytorch

Pro samotnou klasifikaci příznaků z časové řady je použita neuronová síť. Jednoduchou neuronovou síť je možné udělat i ve výše zmíněném balíčku Scikit-learn. Pro větší kontrolu byl v této práci však použit pokročilejší framework Pytorch vyvíjený společností Meta. Ten je dnes více méně standardem při tvorbě neuronových sítí. Ve frameworku Pytorch je možné implementovat i velmi složité neuronové sítě. Narozdíl od dále představených polí v Numpy a listu v jazyce Python je zde základní strukturou tenzor. Ten má velmi podobné vlastnosti jako dále představené Numpy pole.



Obrázek 4.6: Detekce novostí s příznaky ve 2D prostoru. Převzato z https://scikit-learn.org/stable/modules/outlier_detection.html.

Numpy

Numpy je knihovna pro práci s vícerozměrnými poli. Jde o knihovnu často používanou v oblasti strojového učení. Základem je pole, což je datová struktura podobná listu v jazyce Python. Numpy pole jsou však typicky rychlejší a zabírají méně paměti. Numpy také nabízí širokou škálu metod pro pole, jako jsou například metody pro změnu dimenze (například 4x1 na 2x2), vrácení indexu maximální hodnoty, přibližné porovnání dvou polí dle tolerance a podobně. Listy na druhou stranu umožňují různé typy objektů v jednom listu a také jsou vhodnější tam, kde se často mění velikost pole.[3]

Pandas

Pandas je knihovna v některých směrech podobná Numpy. Narozdíl od Numpy je ale primárně určena pro práci s časovými řadami. Základem je tzv. datový rámec (dataframe), který narozdíl od Numpy obsahuje hlavičku, kde jsou sloupce pojmenovány. Indexace pak probíhá pomocí těchto názvů, a nikoliv pouze podle čísla (to je nicméně také možné). Datové rámce také umožňují jednoduchou práci s klouzavými průměry, maximy a podobně. Knihovna se také často používá pro práci s CSV soubory díky podobnosti datových rámců s tímto formátem. V této práci se používá při počítání příznaků v TSFRESH.

Implementace neuronové sítě

Již bylo uvedeno, že síť je implementována ve frameworku Pytorch. První předpokladem pro možnost klasifikace je mít k dispozici model, který bude možné trénovat. Model použitý pro klasifikaci zapsaný v jazyce Python za použití Pytorche vypadá takto:

```
class MLP(Module):
    def __init__(self, n_inputs,n_devices):
        super(MLP, self).__init__()
        self.hidden1 = Linear(n_inputs, 50)
        self.act1 = ReLU()
        self.hidden2 = Linear(50, 30)
        self.act2 = ReLU()
        self.output = Linear(30, n_devices)
        self.act3 = Sigmoid()
```

Tím jsou nadefinovány vrstvy modelu jejich aktivační funkce. Je třeba nadefinovat ještě dopředný průchod sítí, který vypadá takto:

```
def forward(self, X):
    X = self.hidden1(X)
    X = self.act1(X)
    X = self.hidden2(X)
    X = self.act2(X)
    X = self.output(X)
    X = self.act3(X)
    return X
```

Aby bylo možné s modelem pracovat, je nutné jej podle definované architektury instanciovat, dále zvolit optimalizátor a ztrátovou funkci. To lze díky frameworku Pytorch provést třemi řádky:

```
model = MLP(n_features, device_count)
criterion = torch.nn.BCELoss(reduction='mean')
optimizer = Adam(model.parameters(),lr=0.001)
```

Samotné trénování pak probíhá voláním dalších několika řádků v cyklu, dokud se nevyčerpá počet epoch, nebo není ztrátová funkce dostatečně nízká.

```
optimizer.zero_grad()
yhat = model(X_train)
loss = criterion(yhat, y_train)
loss.backward()
optimizer.step()
```

Když je model natrénován, inference probíhá zavoláním metody `predict()` na vstupní vektor s naškálovánými příznaky. Metoda `predict()` pak vrací vektor s N hodnotami pro N spotřebičů. Dle uživatelem nadefinovaného prahu jsou pak pravděpodobnosti příslušnosti ke každé třídě c interpretovány pro každou třídu c jako 0 nebo 1 dle vztahu 4.2.

$$X(c) = \begin{cases} 0 & \text{pro } P(c) < T \\ 1 & \text{pro } P(c) \geq T \end{cases} \tag{4.2}$$

, kde:
X(c) je výsledná příslušnost ke třídě c,
P(c) je pravděpodobnost příslušnosti ke třídě c dle výstupu neuronové sítě.

4.6 Implementace webového rozhraní

Tato sekce se zabývá implementací webového rozhraní dle návrhu ze sekce 3.6 pomocí frameworku Flask.

Flask

Flask je webový microframework pro jazyk Python. Narozdíl od větších frameworků jako jsou například Django a podobné je velmi kompaktní. Využívá šablonovací systém Jinja2. Díky tomu je možné využívat například dědičnost při vytváření jednotlivých stránek. To se hodí třeba u stavové lišty (obrázek 4.8), která je k dispozici na všech stránkách v systému, bez nutnosti zásahu do všech šablon při změně v liště. Také je možné doinstalovat různá rozšíření z repozitáře jako například WTForms pro validaci formulářů, knihovny pro práci s databázemi a podobně. Celá aplikace je pak rozdělena do několika souborů. Ty jsou uspořádány v následující struktuře:

```
smart_socket
├── app.py
├── classification.py
├── communication.py
├── config.json
├── db_access.py
├── forms.py
├── static/
└── templates/
```

Soubor **app.py** obsahuje routy pro jednotlivé stránky aplikace.

V souboru **classification.py** jsou metody pro klasifikaci a práci s modelem. Jsou zde definovány metody pro vytvoření datasetu, výběr a extrakci příznaků, architektura neuronové sítě a metody pro trénování a inferenci. Výsledky klasifikace se aktualizují každou sekundu do stavové lišty, která je na obrázku 4.8.

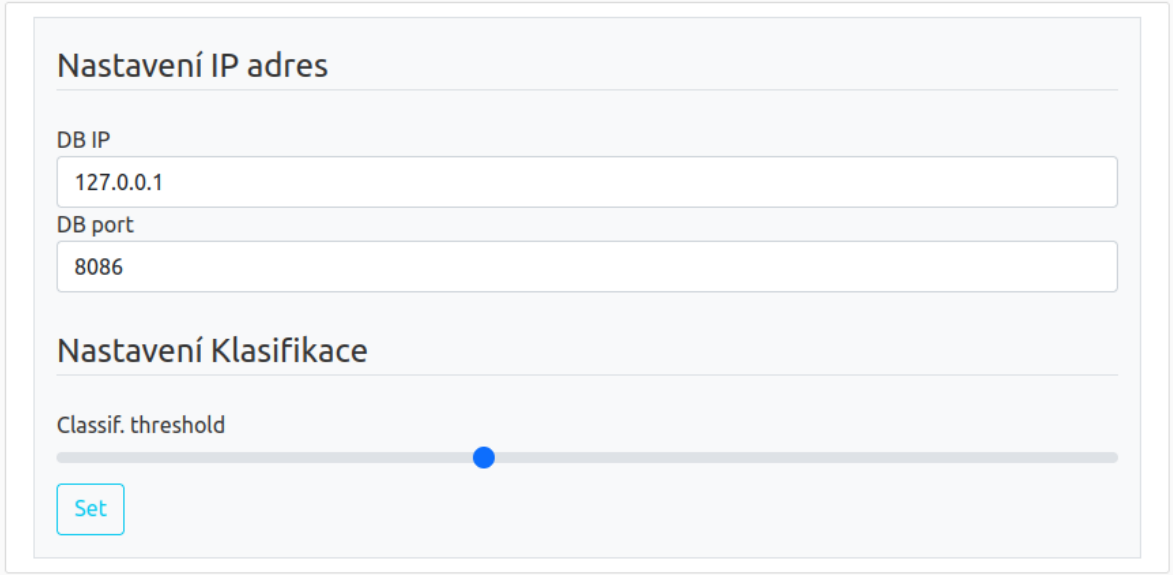
Metody pro komunikaci s databází jsou v souboru **db_access.py**. Jedná se o metody pro vyčtení informací z databáze a jejich formátování pro klasifikaci a vykreslování do grafu. Také jsou zde metody pro kontrolu připojení. V souboru **communication** je implementace komunikace s Node-RED.

Ve **forms.py** jsou definovány formuláře implementované pomocí výše zmíněného modulu WTForms. Díky jeho použití je možná snadná validace a definice formulářů. Formuláře také obsahují skryté políčko pro použití csrf tokenu pro zamezení útoku Cross-site Request Forgery. Jednoduchý formulář může vypadat například takto:

```
db_ip = StringField('DB IP', validators=[DataRequired(), IPAddress()])
db_port = StringField('DB port', validators=[DataRequired()])
thr = IntegerRangeField('Classif. threshold', validators=[DataRequired()])
submit = SubmitField('Set')
```

Výsledek je pak na obrázku 4.7. Zajímavé je především velmi jednoduché použití validace. Validátorů je mnoho, kromě výše uvedených je k dispozici například kontrola rozsahu zadaných hodnot, kontrola MAC adresy, emailové adresy, nebo pokročilá validace pomocí uživatelem zadaného regulárního výrazu.

Ve složce **static/** jsou obrázky. Ve složce **templates/** pak šablony pro jednotlivé stránky



Obrázek 4.7: Jednoduchý formulář vytvořený za pomocí WTForms, Jinja2 a Bootstrapu.

ve formátu pro Jinja2.

AJAX

AJAX (Asynchronous JavaScript and XML) je technologie umožňující asynchronní komunikaci. To znamená, že pro změnu obsahu na stránce není nutné ji znovu načítat. AJAX je v aplikaci použitý na mnoha místech. Hlavním použitím je provádění klasifikace každou sekundu. Komunikace probíhá následovně:

1. zjistí se, zda je možná komunikace s databází a zásuvkou,
2. pokud komunikace není možná, do stavové lišty se propíšou tyto informace,
3. pokud komunikace možná je, pokračuje se vyčtením posledních 5 hodnot z databáze,
4. je provedena extrakce příznaků, normalizace, detekce novosti, klasifikace,
5. výsledky se porovnají s nastavenými prahy a dle výsledku je do databáze zapsaná aktuální spotřeba a zároveň výsledek klasifikace,
6. na základě výsledků je aktualizována stavová lišta.

Tento algoritmus je prováděn pomocí funkce `setInterval()` každou sekundu.

Implementace stavové lišty

Ve stavové liště je zobrazen aktuální stav systému. Jak lišta vypadá je vidět na obrázku 4.8. Vlevo se nacházejí indikátory systému. Tedy zda běží Node-RED, zásuvka je zapnutá a připojená na internet a zda je možný přístup do databáze. Připojení k Node-RED není vyžadováno. Pokud je však zjištěno, že není možná komunikace se zásuvkou nebo databází,

(a) Lišta v běžném stavu

(b) Lišta při detekci neznámého připojeného zařízení

(c) Lišta při nemožnosti komunikaci se zásuvkou

Obrázek 4.8: Různé stavy přehledové lišty.

klasifikace je pozastavena (nebylo by co klasifikovat). V momentě, kdy se zjistí že je možné se k zásuvce i databázi připojit, je klasifikace opět spuštěna.

V pravé části lišty jsou zobrazeny informace o zapnutých spotřebičích. Ty se aktualizují automaticky v reálném čase jednou za sekundu pomocí technologie AJAX představené výše. Pokud je zjištěno připojení neznámého spotřebiče, jsou ikonky spotřebičů nahrazeny nápisem „UNKNOWN DEVICE CONNECTED“. Pokud nastane situace, kdy není možná komunikace se zásuvkou nebo databází, je uživateli zobrazen text „DISCONNECTED“, jak je také vidět na obrázku.

Implementace statistik

Jak bylo zmíněno v kapitole 3.6, kromě zobrazení aktuálních informací je možné zobrazovat informace i zpětně. Uživatel může zvolit období, které se má vyčíst z databáze a pro toto období se do dvou grafů vykreslí naměřená spotřeba a zapnutá zařízení v danou dobu. Grafy průběhu spotřeby a detekovaných zařízení pro tuto spotřebu jsou uvedeny na obrázku 4.9. Pro lepší přehlednost zobrazení jsou pro různé časové úseky použité různé hodnoty klouzavých průměrů, aby měla křivka spotřeby lepší vypovídající hodnotu.

Implementace konfigurace systému

Dle návrhu ze sekce 3.6 byla implementována stránka s formulářem, který umožňuje nastavení všech potřebných hodnot. Těmi jsou následující:

- ip adresa zásuvky,
- adresa databáze,
- port databáze,
- ip adresa Node-RED,
- port Node-RED,
- zasílání požadavků do Node-RED,
- citlivost klasifikace,

- citlivost detekce novostí.

Po uložení jsou data zapsána na disk do souboru JSON v následujícím formátu do souboru config.json.:

```
config = {
"db_ip":"127.0.0.1",
"db_port":"8086",
"socket_ip":"192.168.0.113",
"node_ip":"127.0.0.1",
"node_port":"1880",
"notifications_system":false,
"notifications_devices":false,
"dev0":"TV",
"dev1":"LAMP",
"dev2":"NOTEBOOK",
"classification_threshold":0.7
"novelty_detection_threshold":0.1
}
```

Všechny hodnoty v soboru jsou povinné. U hodnot dev1 a dev2 může být uveden prázdný řetězec. To značí, že zařízení nebylo využito. Pokud je tedy například dev2 prázdný řetězec, značí to, že byla zásuvka natrénována pro dvě zařízení. V takovém případě je pak ve stavové liště příslušné políčko skryto a zobrazují se pouze dvě hodnoty.

Implementace rozhraní trénování modelu

Rozhraní pro trénování modelu se skládá ze dvou stránek. Na první je možné zvolit zdroj trénovacích dat (DB/soubor). Při zvolení databáze je nutné zadat kolik minut zpět se má vykreslit. Na další stránce jsou 3 x 2 políčka. Pro každé zařízení je nutné zadat počáteční a konečný index ohraničující spotřebu daného zařízení. Zařízení je také nutné pojmenovat. Trénovat lze pro 1 až 3 zařízení. Vyplnění hodnot pro device0 je povinné. Další dvě jsou volitelné. Trénování modelu se pak spustí kliknutím na tlačítko „Start training“. Po dokončení trénování je uživatel přesměrován na stránku s nastavením pro případnou úpravu prahu klasifikace.

4.7 Připojení k chytré domácnosti a komunikace

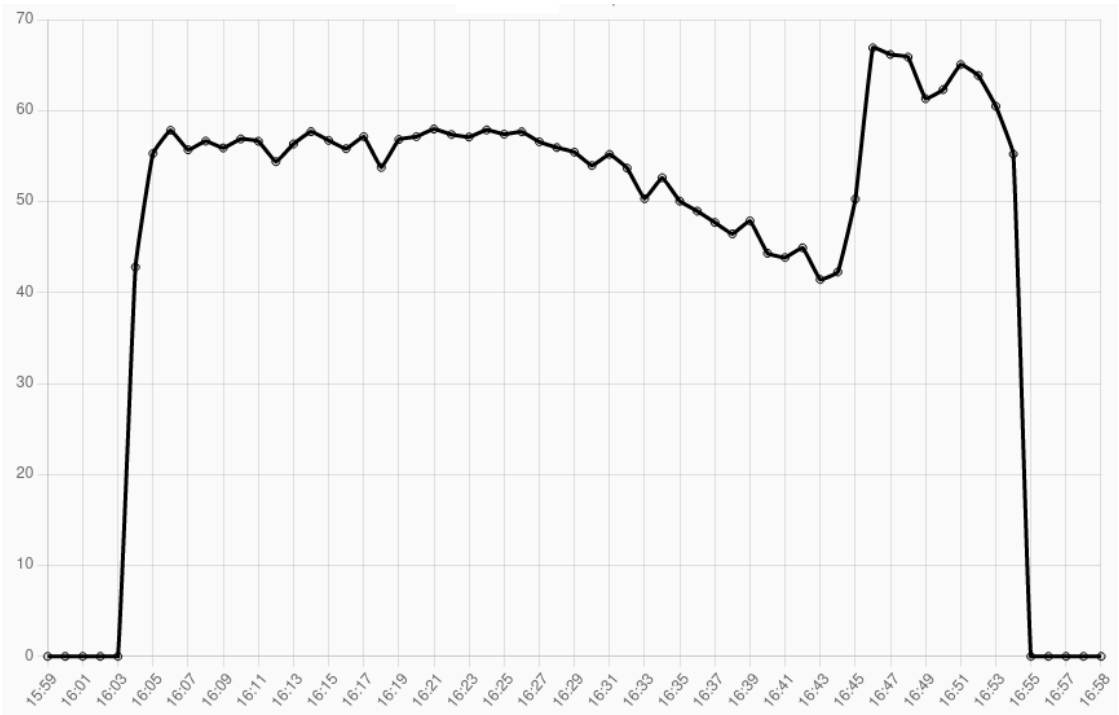
V této sekci je popsáno, jak je naimplementována komunikace s Node-RED.

Implementace komunikace

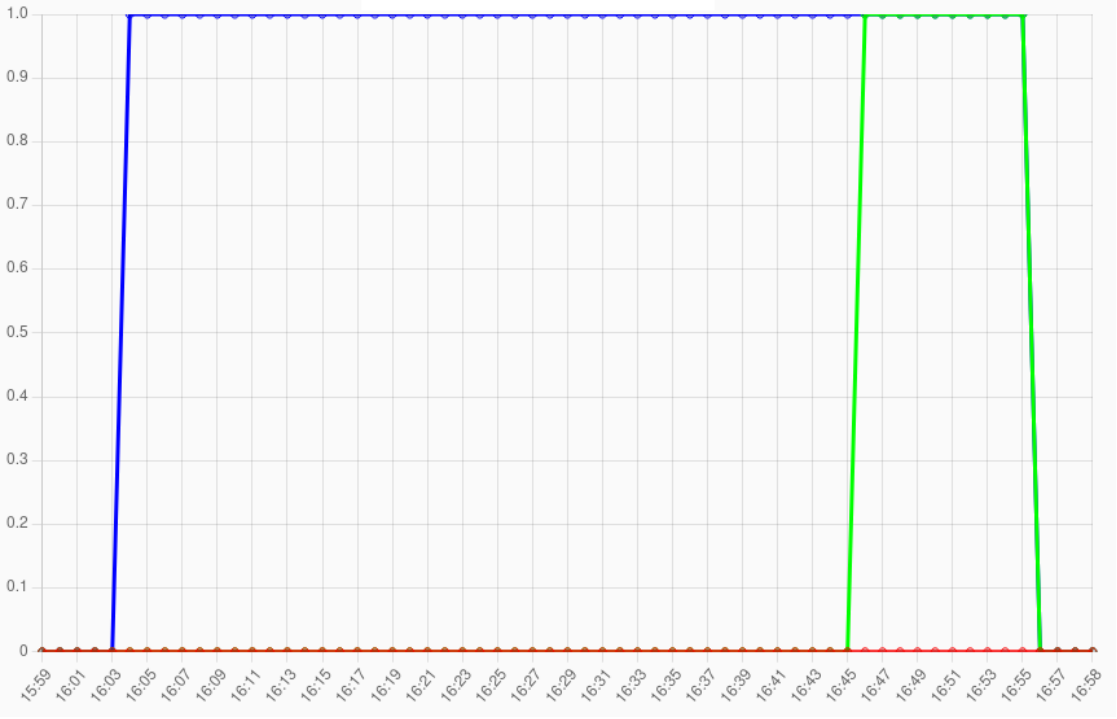
Komunikace s Node-RED je implementována pomocí knihovny requests. Ta umožňuje zasílaní GET nebo POST požadavků na uživatelem zadanou adresu. Z konfiguračního souboru uvedeného v sekci 4.6 je vyčtena adresa a port na které se má požadavek poslat. Odeslání požadavku na endpoint „status_devices“ pak může vypadat například takto:

```
change = {'change':'LAMP: OFF -> ON'}
requests.post("127.0.0.1:1880+"/status_devices", data=change)
```

Aby bylo možné přijímat požadavky na straně Node-RED je potřeba jej definovat. Ukázka

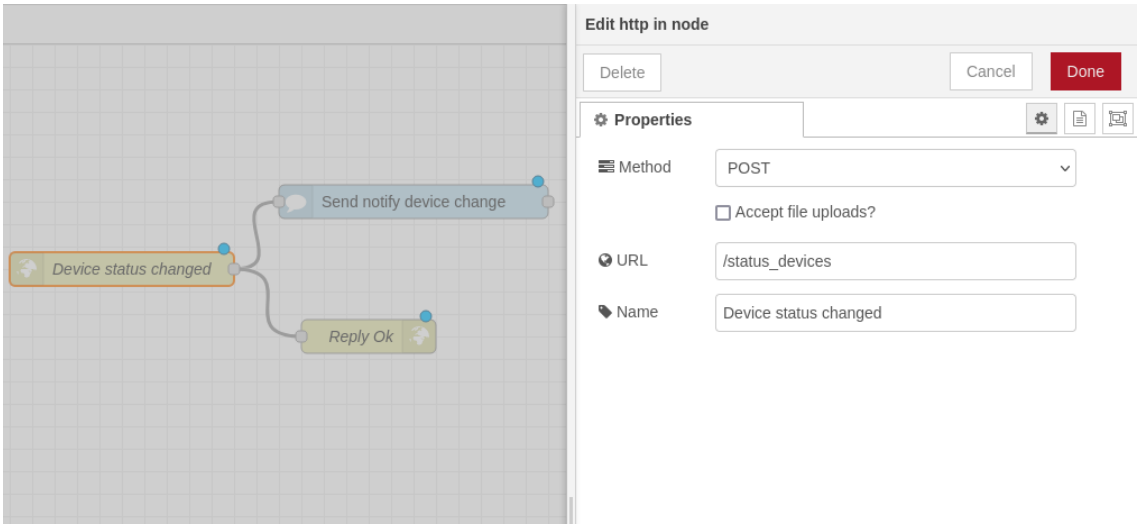


(a) Zpětně zobrazená spotřeba za poslední hodinu



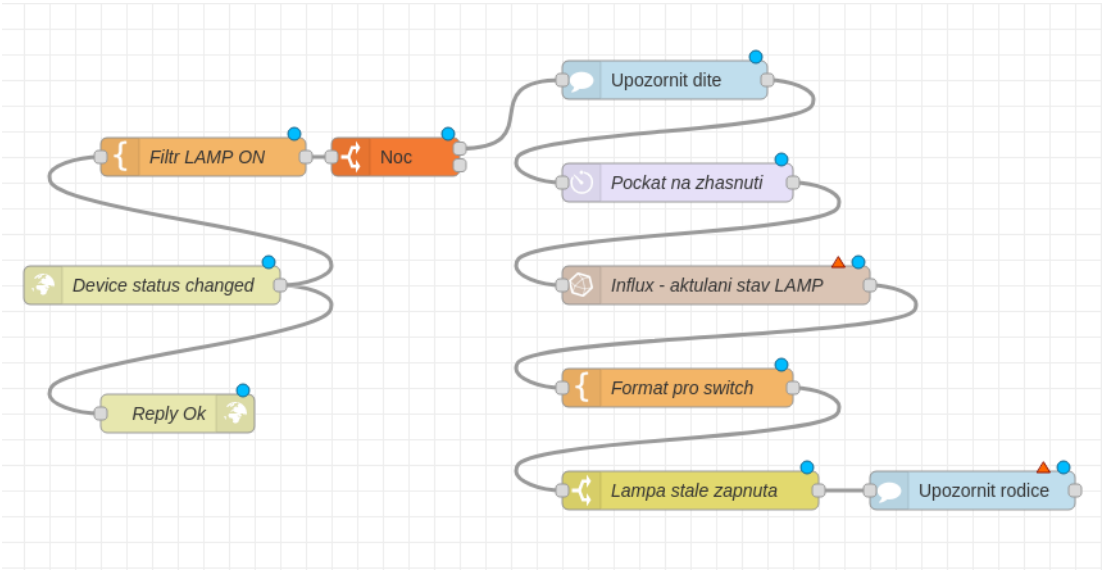
(b) Klasifikace spotřebičů dle spotřeby (červeně TV, zeleně lampa, modře notebook)

Obrázek 4.9: Zpětné zobrazení spotřeby a klasifikace.



Obrázek 4.10: Kontextová nabídka při editaci endpointu v Node-RED.

kontextového menu při editaci endpointu je ukázána na obrázku 4.10. Když přijde požadavek na danou adresu, kromě odpovědi OK je také vidět uzel pro zaslání notifikace na mobilní telefon. Na obrázku 4.11 je ukázáno složitější pravidlo pro rodičovskou kontrolu



Obrázek 4.11: Příklad složitějšího pravidla pro Node-RED.

(zapnutí lampičky v noci, kdy by dítě mělo spát). Po obdržení HTTP požadavku o změně stavu zařízení je vyfiltrováno zařízení „LAMP“ šablonovacím uzlem. Poté je pomocí uzlu pro rozsah časů rozhodnuto, zda je zrovna noc (22:00 až 07:00). Pokud ano, je nejprve upozorněno notifikací dítě, že by mělo spát a ať lampu zhasne. Node-RED poté počká 5 minut a pomocí dotazu do InfluxDB je vyhodnoceno, zda je lampa stále rozsvícená. Poté se odpověď zformátuje a pomocí přepínače se vyhodnotí, zda je stále zapnutá. Pokud ano, je upozorněn rodič.

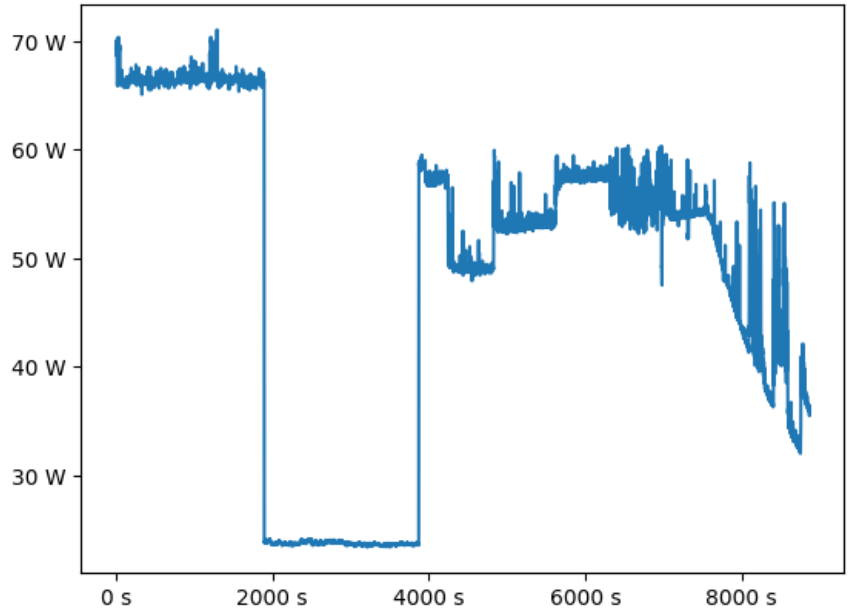
Kapitola 5

Dataset a vyhodnocení

V této kapitole bude detailněji popsán použitý dataset, poté bude popsáno, jak probíhalo vyhodnocení a jeho výsledky. Na konci kapitoly je také uvedeno několik nápadů, jak by bylo možné představenou aplikaci dále vylepšit.

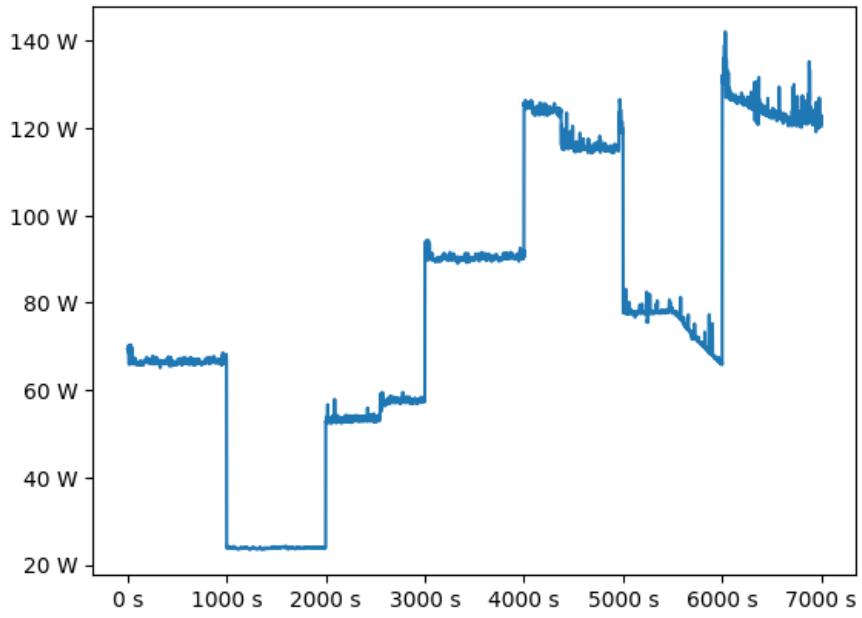
5.1 Použitý dataset

Dataset použitý pro vyhodnocení se skládá ze tří běžných spotřebičů, tedy 7 možných kategorií pro klasifikaci. Prvním spotřebičem byla televize, druhým stolní lampička a třetím notebook. Jejich průběhy jsou vykresleny na obrázku 5.1.



Obrázek 5.1: Průběh spotřeby jednotlivých zařízení. V čase 0-2000s TV, 2000-4000s lampa a dále notebook.

Z těchto dat byl postupem popsaným v 3.3 vytvořen dataset pro trénování. Graf skutečně naměřené spotřeby pro kombinace je na obrázku 5.2.



Obrázek 5.2: Naměřené hodnoty pro jednotlivé kombinace spotřebičů (ořezáno na 1000s pro kategorii).

5.2 Metodika vyhodnocení

Při trénování a vyhodnocování neuronových sítí se obvykle používá přístup rozdělení datasetu na 2 části, například v poměru 80 a 20 %, přičemž na první části je model trénován a na druhé testován.
Takový přístup však v této práci není možné použít. Jak je popsáno v kapitole 3.3, je žádoucí, aby měl model při trénování k dispozici co nejrůznorodější průběhy spotřeby.
Zde je tedy použitý přístup, kdy jsou změřeny průběhy spotřebičů jednotlivě a z těch je sestaven trénovací dataset. Poté se provede druhé měření, kdy jsou již spotřebiče skutečně zapínány a vypínány a spotřeba není sčítána uměle jako je uvedeno v tabulce 3.1, ale je měřena dohromady.

5.3 Výsledky

V této sekci jsou popsány výsledky detekce neznámého zařízení a klasifikace.

Detekce neznámého zařízení

Testování detekce neznámého zařízení probíhalo pro 2 spotřebiče. Prvním z nich byl vysavač se spotřebou cca 1400W. Zde bylo zařízení označeno jako novost pomocí SVM a to s úspěšností 100 % pro všechny vyzkoušené kombinace. To lze díky řádově odlišné spotřebě oproti ostatním zařízením očekávat. Druhým spotřebičem byl aroma difuzér se spotřebou v rozsahu 8 až 18W. Při samostatném zapojení tohoto spotřebiče byla úspěšnost detekce neznámého zařízení opět 100 %.

TP	FP	TN	FN
450	21	264	166

Tabulka 5.1: Senzitivita a specificita detekce neznámého spotřebiče pro různé kombinace.

	TV	LAMP	NTB	TV+LAMP	TV+NTB	LAMP+NTB	ALL
TV	2050	0	0	0	0	7	0
LAMP	0	2031	0	0	0	0	0
NTB	0	0	2081	0	0	0	0
TV+LAMP	0	0	0	1981	0	0	0
TV+NTB	0	0	0	0	2004	0	7
LAMP+NTB	9	0	0	0	0	2080	0
ALL	0	0	0	3	2	0	1985

Tabulka 5.2: Matice záměn pro měřené hodnoty.

Při kombinování s jinými (naučenými) spotřebiči však byla detekce horší. Celkově bylo testováno 901 vzorků různých kombinací. Výsledky jsou uvedeny v tabulce 5.1. Z té je patrné, že pokud začnou být spotřeby kombinovány, začne správnost detekce klesat.

Klasifikace

Z celkového množství bylo správně predikováno přes 99 % časových řad. Matice záměn jednotlivých kategorií je uvedena v tabulce 5.2. Z tabulky je patrné, že klasifikace je velmi přesná. Při využití hystereze, kdy klasifikace neprobíhá pouze dotazováním modelu jsou výsledky téměř bezchybné. Záleží zde však na použitém datasetu. Pokud by měly spotřebiče velmi podobný průběh, klasifikace by mohla dopadnout hůře. I přesto, že má klasifikátor k dispozici různé příznaky, je zřejmé že nejjednodušší je rozlišit ty s velmi různými hodnotami. V takovém případě podobných průběhů by se model mohl naučit rozlišovat na základě změn v okně, nebo složitějších příznaků. V případě velmi podobných průběhů by však bylo nejspíše nutné nastavit delší časové okno pro měření, jak je uvedeno v sekci 5.4.

5.4 Možná vylepšení

V této sekci budou popsána možná vylepšení, která by mohla zlepšit funkčnost systému.

Lepší měřák

Pravděpodobně nejjednodušším a zároveň nejlepším vylepšením (při zanedbání ceny) by bylo použití zařízení, které by umožňovalo měření spotřeby s vyšší frekvencí. V případě, že by umělo snímat aktuální spotřebu třeba 10x za sekundu, klasifikátor by měl k dispozici 10x delší řadu při zachování stejné délky časového okna. Další možnosti by bylo třeba měření účiníku. To by se pravděpodobně výrazně projevilo na přesnosti klasifikace. Extrakce příznaků by sice trvala déle, ale klasifikace by pravděpodobně byla přesnější.

Takové zařízení by bylo možné také využít naopak, tedy ke zrychlení detekce. Časové okno by tak mohlo být kratší a hodnot by bylo stejně nebo více. Vzhledem k tomu, že cílem této práce bylo představit cenově výhodné řešení, byl použit měřák za cca 600kč s frekvencí 1Hz. Algoritmy by nicméně byly s drobnou změnou využitelné i pro měřák s vyšší frekvencí.

Lepší model

Dalším vylepšením by mohlo být vylepšení/zvětšení modelu pro klasifikaci. Neuronová síť použitá v této práci byla navržena tak, aby běžela i na zařízeních typu Rasberry Pi a podobných.

Celá síť se nyní trénuje na CPU bez využití GPU nebo TPU akcelerátorů. To klade značně omezující podmínky na velikost a složitost sítě. V případě, že by byla přidána možnost trénování sítě například v cloudu, bylo by možné použít větší model, který by se po natrénování stáhl do pomalejšího zařízení, kde by již probíhala pouze inference. Pokud by byl použit větší model a více příznaků, přesnost, resp. rychlost klasifikace by se pravděpodobně zvýšila.

Větší volnost nastavení klasifikace

Velikost okna i hystereze byly nastaveny empiricky dle použitého datasetu uvedeného v sekci 5, aby byla klasifikace téměř bezchybná a co nejrychlejší. V současné podobě je uživateli umožněno nastavovat pouze prahy pro klasifikaci a detekci novostí. Pokud by chtěl uživatel do klasifikace více zasahovat, bylo by možné některé parametry nastavovat třeba před spuštěním trénování.

To by bylo výhodné zejména při klasifikaci zařízeních s velmi odlišnými průběhy. Pokud by uživatel chtěl zásuvku například použít pro dvě zařízení, přičemž jedno by mělo konstantní spotřebu 60W a druhé 20W, klasifikace by byla triviální. V takovém případě by například bylo možné úplně vypnout hysterezi a okno zkrátit například na 3 sekundy. Reakce systému by tak byla mnohem rychlejší.

V opačném případě by bylo možné naopak časové okno a délku hystereze zvýšit za cenu přesnější klasifikace. Při detekci konce pracího cyklu by uživateli nemuselo vadit zpoždění příchodu notifikace třeba půl minuty. Pokud by však byla klasifikace falešným pozitivem, nejspíš by nadšený nebyl, kdyby po příchodu do prádelny zjistil, že pračka bude prát ještě půl hodiny. Pro oba tyto případy by tedy mohlo být vhodné nastavení těchto parametrů na vlastní hodnoty.

Další funkce

Kromě klasifikace by bylo možné zásuvku používat například k detekci poruch, nebo anomálií v síti, a ne pouze detekci připojení neznámého spotřebiče. Stejně jako je do Node-RED posílána informace o připojení spotřebiče by tak bylo možné například poslat třeba informaci o přepětí v síti.

Kapitola 6

Závěr

Cílem této práce bylo představit opravdu chytrou chytrou zásuvku, která by překonala nedostatky dnes dostupných řešení. Taková zásuvka by měla umožňovat odečet aktuální spotřeby, zobrazovat co je do ní kdy připojeno a uchovávat statistiky pro zpětné zobrazení. Informace o změně stavu by měla být schopna posílat do nástroje pro správu chytré domácnosti, kde lze s informací dále pracovat. To vše s co nejmenší investicí času a peněz pro koncového uživatele.

Při práci bylo nejdříve potřeba naměřit data a ta vypisovat z databáze. K tomu byla použita databáze InfluxDB, která je určená k práci s časovými řadami. Z časové řady bylo za pomocí knihovny TSFRESH vypočítáno několik příznaků. Ty byly po normalizaci nástroji Scikit-learn a Pytorch klasifikovány dle krátkého časového okna s průběhem spotřeby jako jednotlivé spotřebiče. K zásuvce bylo také vytvořeno webové rozhraní pomocí frameworku Flask. Dále byla implementována možnost změnu stavu posílat dále do nástroje Node-RED.

Výsledkem práce je plně funkční chytrá zásuvka, která se umí naučit kombinace až tří současně připojených spotřebičů, díky čemuž je řešení velmi levné. Spotřebiče v představeném datasetu je schopna rozpoznat s přesností přes 99 % v řádu několika sekund, a to včetně upozornění při detekci připojení neznámého spotřebiče.

Zásuvku lze díky webovému rozhraní využít k zobrazení informací o aktuální spotřebě a momentálně připojených zařízeních. Díky pravidelnému zápisu do databáze spotřeby a výsledků klasifikace do databáze je také možné zpětné zobrazení statistik o spotřebě, nebo přehledu, kdy bylo jaké zařízení zapnuto v zásuvce.

Vzhledem k rychlosti detekce a možnosti komunikace se systémem Node-RED, lze zásuvku jednoduše začlenit do celé chytré domácnosti. Tak ji lze využít například pro rodičovskou kontrolu, nebo informování o konci pracího cyklu pračky zasíláním notifikace na mobilní telefon. Kromě těchto jednoduchých úkonů lze také v Node-RED vytvořit složitější pravidla a na základě změny stavu tak třeba ovládat ostatní prvky v chytré domácnosti.

Literatura

[1] *Smart Home* [https://www.statista.com/outlook/dmo/smart-home/worldwide]. Accessed: 2022-4-19.

[2] Boser, B. E., Guyon, I. M. a Vapnik, V. N. A Training Algorithm for Optimal Margin Classifiers. In: *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*. ACM Press, 1992, s. 144–152.

[3] Chanduthedev. *Python list vs Numpy Array*. DEV Community, Jan 2021. Dostupné z: https://dev.to/chanduthedev/python-list-vs-numpy-array-3pjp.

[4] Gubbi, J., Buyya, R., Marusic, S. a Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*. 2013, sv. 29, č. 7, s. 1645–1660. DOI: https://doi.org/10.1016/j.future.2013.01.010. ISSN 0167-739X. Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0167739X13000241.

[5] Hamid, O., Barbarosou, M., Papageorgas, P., Prekas, K. a Salame, C.-T. Automatic recognition of electric loads analyzing the characteristic parameters of the consumed electric power through a Non-Intrusive Monitoring methodology. *Energy Procedia*. 2017, sv. 119, s. 742–751. DOI: https://doi.org/10.1016/j.egypro.2017.07.137. ISSN 1876-6102. International Conference on Technologies and Materials for Renewable Energy, Environment and Sustainability, TMREES17, 21-24 April 2017, Beirut Lebanon. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1876610217326942.

[6] Jahn, M., Jentsch, M., Prause, C. R., Pramudianto, F., Al Akkad, A. et al. The Energy Aware Smart Home. In: *2010 5th International Conference on Future Information Technology*. 2010, s. 1–8. DOI: 10.1109/FUTURETECH.2010.5482712.

[7] Jiang, L., Liu, D.-Y. a Yang, B. Smart home research. In: *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*. 2004, sv. 2, s. 659–663 vol.2. DOI: 10.1109/ICMLC.2004.1382266.

[8] King, N. *Smart Home Paper - Housing lin*. 2003. Dostupné z: https://www.housinglin.org.uk/_assets/Resources/Housing/Housing_advice/Smart_Home_-_A_definition_September_2003.pdf.

[9] MertArduino. *Turn any decorative thing into a smart device.* Instructables, Mar 2021. Dostupné z: https://www.instructables.com/Turn-Any-Decorative-Thing-Into-a-Smart-Device/.

[10] Nguyen, V. K., Phan, M.-H., Zhang, W. E., Sheng, Q. Z. a Vo, T. D. A Hybrid Approach for Intrusive Appliance Load Monitoring in Smart Home. In: *2020 IEEE International Conference on Smart Internet of Things (SmartIoT).* 2020, s. 154–160. DOI: 10.1109/SmartIoT49966.2020.00031.

[11] Pedrasa, M. A. A., Spooner, T. D. a MacGill, I. F. Coordinated Scheduling of Residential Distributed Energy Resources to Optimize Smart Home Energy Services. *IEEE Transactions on Smart Grid.* 2010, sv. 1, č. 2, s. 134–143. DOI: 10.1109/TSG.2010.2053053.

[12] Revuelta Herrero, J., Lozano Murciego, l., Barriuso, A., Iglesia, D. Hernández de la, Villarrubia, G. et al. Non Intrusive Load Monitoring (NILM): A State of the Art. In:. červen 2018, s. 125–138. DOI: 10.1007/978-3-319-61578-3.12. ISBN 978-3-319-61577-6.

[13] Ridi, A., Gisler, C. a Hennebert, J. Processing smart plug signals using machine learning. In: *2015 IEEE Wireless Communications and Networking Conference Workshops (WCNCW).* 2015, s. 75–80. DOI: 10.1109/WCNCW.2015.7122532.

[14] Rumelhart, D. E., Hinton, G. E. a Williams, R. J. Learning representations by back-propagating errors. *Nature.* 1986, sv. 323, s. 533–536.

[15] Sahu, V. *Power of a single neuron.* Towards Data Science, Jun 2021. Dostupné z: https://towardsdatascience.com/power-of-a-single-neuron-perceptron-c418ba445095.

[16] Simons, H. *We asked, you told us: Here's how often you upgrade to a new TV.* Dec 2021. Dostupné z: https://www.androidauthority.com/how-often-upgrade-tv-poll-results-3070117/.

[17] Wise, J. *Smart Home Statistics 2022: How many smart homes are there?* Mar 2022. Dostupné z: https://earthweb.com/smart-home-statistics/.