Artificial Intelligence Integration with Context-Awareness in Smart Ecosystem

Bruno Izaias Bonotto <izaias.bonotto@gmail.com>
Fabíola Maria Kretzer <fabiolakretzer@hotmail.com>
João Vicente Souto <joaovicentesouto@gmail.com>

Introdução

Com o avanço tecnológico nas áreas de comunicação, sensoriamento e dispositivos de atuação, ambientes inteligentes vem se tornando uma realidade cada vez mais alcançável. Utilizando-se da grande base de conhecimento tecnológico produzido ao longo do tempo, a Internet da Coisas - Internet of Things (IoT) surge a partir de um esforço conjunto entre a área de pesquisa e o setor industrial para integrar a grande heterogeneidade de dispositivos conectados a internet.

Embora cloud, edge e fog serem conceitos ainda com bastante desenvolvimento, apenas quando existirem sistemas de fato conscientes de contexto é que será possível construir ambientes realmente inteligentes, capazes de apreender, compreender e se adaptar às adversidades do mundo da melhor forma possível.

Desta forma, utilizando o conceito de computação consciente de contexto - Context-Aware Computing buscaremos implementar uma hierarquia de sistemas, baseando-se em técnicas de cloud computing, machine learning e big data, com o intuito de prover uma abstração de como seria um ecossistema inteligente. Por fim, tal ecossistema será capaz de aprender e determinar o contexto do ambiente com base nas preferências de determinado usuário.

Conceitos Importantes

Para melhor entendimento deste documento é importante que o leitor esteja familiarizado com certos conceitos, os quais são amplamente utilizados para viabilizar este trabalho. Assim, este capítulo busca, de forma mais superficial, informar o que se faz necessário entender antes de se aprofundar na leitura deste documento.

i. Cloud computing

Consiste no acesso global a uma arquitetura, chamada de cloud, que permite o gerenciamento das informações de dispositivos físicos {1}. Utilizando a cloud o armazenamento dos dados da IoT e as capacidades de computação podem ser aumentadas de maneira escalável, ou seja, alteradas de acordo com a demanda. Assim, os sensores e dados por eles coletados podem ser utilizados de qualquer lugar {2}.

ii. Internet of things (IoT)

A Internet das Coisas - Internet Of Things é uma rede na qual se conecta diversos objetos físicos heterogêneos, os quais são chamados de “coisas”. Tais dispositivos podem ser sensores, computadores, câmeras de vigilância, eletrodomésticos etc. Cada objeto é exclusivamente endereçado e conectado à internet utilizando protocolos de comunicação, os quais permite a comunicação entre eles {2}.

iii. Smart ecosystem

Um ecossistema inteligente - smart ecosystem é uma abstração da integração de diversos ambientes inteligentes em um único sistema inteligente e adaptativo. O conceito determina que todos os objetos conectados a IoT coletem e compartilhem dados e informações. Dessa maneira, este ecossistema (sistemas, pessoas etc.) realiza ações para facilitar a vida do ser humano {2}.

iv. Context-aware computing

Computação consciente de contexto - Context-aware computing é um termo utilizado para representar o aprendizado e adaptação de dispositivos inteligentes ao perceberem as mudanças no ambiente. Para isso um sistema sensível ao contexto precisa adquirir, compreender e reconhecer o contexto do ambiente e realizar determinadas ações {2}. Dessa maneira, faz-se necessário o conhecimento e aplicação de conceitos de machine learning e big data {1}.

v. Machine learning

Aprendizagem de máquina - Machine learning é uma área de estudo que busca oferecer aos computadores a capacidade de aprendizagem, sem que o conhecimento seja explicitamente fornecido. Seu principal uso em IoT consistem em fornecer algoritmos de previsão e decisão, a partir do uso da grande quantidade de dados disponíveis e coletados de fontes diferentes {2}.

vi. Data mining

Exploração de dados - Data mining consiste no processo de utilizar diferentes perspectivas para realizar uma análise de dados. Durante este processo, os dados são resumidos e transformados em informações úteis, as quais podem ser usadas para diversos fins. Em geral, este processo é aplicado para encontrar correlações ou padrões entre dezenas de campos em grandes bases de dados. É possível, também, reduzir o tamanho do espaço analítico e melhorar a precisão dos algoritmos usados, uma vez definida as relevâncias de determinadas variáveis em relação às outras {3}.

vii. Feature selection

Seleção de características - Feature selection é uma estratégia de pré-processamento de dados a qual provou-se eficaz e eficiente na preparação de dados para os algoritmos de data mining e machine learning {5}. Devido à presença de dados redundantes e irrelevantes os algoritmos de aprendizado podem ser lentos, diminuindo o desempenho de tarefas de aprendizagem {6}. Para minimizar os efeitos desses problemas, o algoritmo de feature selection constrói modelos mais simples e mais compreensíveis, apresentando os dados mais relevantes e melhorando o desempenho dos demais algoritmos {5}.

viii. Deep learning (redes neurais)

Na Internet das Coisas (IoT), diversos sensores coletam e (ou) geram uma grande quantidade de dados ao longo do tempo, os quais são utilizados por diversas aplicações {7}. Nesse sentido, aprendizagem profunda - deep learning é uma estratégia para extrair informações precisas desses dados, realizar análises sobre eles e descobrir novas informações com base nas já conhecidas, prover insights, e tomar decisões de controle {7}.

Objetivos

Objetivos gerais

Este trabalho tem como principal objetivo a implementação de uma arquitetura de context-aware computing com base na arquitetura proposta em {4}. No entanto, este trabalho está mais focado na implementação de um controlador (daemon) remoto responsável por lidar com a grande quantidade de dados recebidos do gateway. Assim, evita-se comunicação excessiva da aplicação de aprendizado com o banco de dados e, também, é possível manter a qualidade de serviços oferecida em {4} lidando com as restrições impostas por sistemas remotos e de tempo real.

Objetivos especifícos

1. Coletar dados do ambiente utilizando sensores (smart data);
2. Implementar um gateway capaz de receber dados enviados pelos sensores e enviá-los ao servidor e também reconhecer um usuário na rede;
3. Implementar, no servidor do Lisha, um controlador com as seguintes características:
    1. Controlar a comunicação da aplicação de aprendizagem com o banco de dados, evitando excesso de solicitações de dados;
    2. Controlar volume de dados enviados pelo gateway através de uma cache de dados;
    3. Controlar o volume de comandos enviados pelo usuário através de uma cache de comando do usuário;
    4. Utilizar um modelo de rede neural para prever as condições ideias do ambiente para um dado usuário;
    5. Determinar quando a rede neural precisa ser re-treinada com base no volume de dados e comandos do usuário recebidos;
    6. Comunicar-se com o dispositivo do usuário permitindo-o receber as configurações para o ambiente no qual ele se encontra;
4. Criar um servidor de aprendizado remoto utilizando a biblioteca WEKA para filtragem dos dados e aprendizagem do contexto desses dados;
5. Implementar uma aplicação para dispositivos móveis que permita visualizar as configurações ideais para o ambiente em que o usuário se encontra;
6. Implementar contextualização multiusuário (opcional).

Metodologia

Para a implementação da arquitetura de computação consciente de contexto, primeiramente é necessário a obtenção de informações do ambiente em que o usuário está inserido. Para isso, são utilizados sensores EPOS Mote III+ capazes de monitorar certas características desse ambiente, tais como a temperatura e umidade. Além disso, são realizadas coletas de dados fora do ambiente do usuário, ou seja, ao “ar livre”. Tais dados são necessários para estabelecer relação entre a configuração do tempo local (no quesito clima) e a do ambiente configurado pelo usuário.

Assim como ilustra a Figura 1, os sensores instalados no ambiente realizam medições periódicas de determinadas grandezas do ambiente e enviam os dados coletados ao gateway (EPOS Mote III). Este gateway é responsável por receber esses dados e enviá-los de forma organizada ao servidor. Além disso, o

gateway também é responsável por reconhecer o usuário na rede quando o mesmo se conectar. A Figura 1 ilustra também os principais componentes do servidor, sendo eles: a REST API, o controlador, a rede neural e o banco de dados.

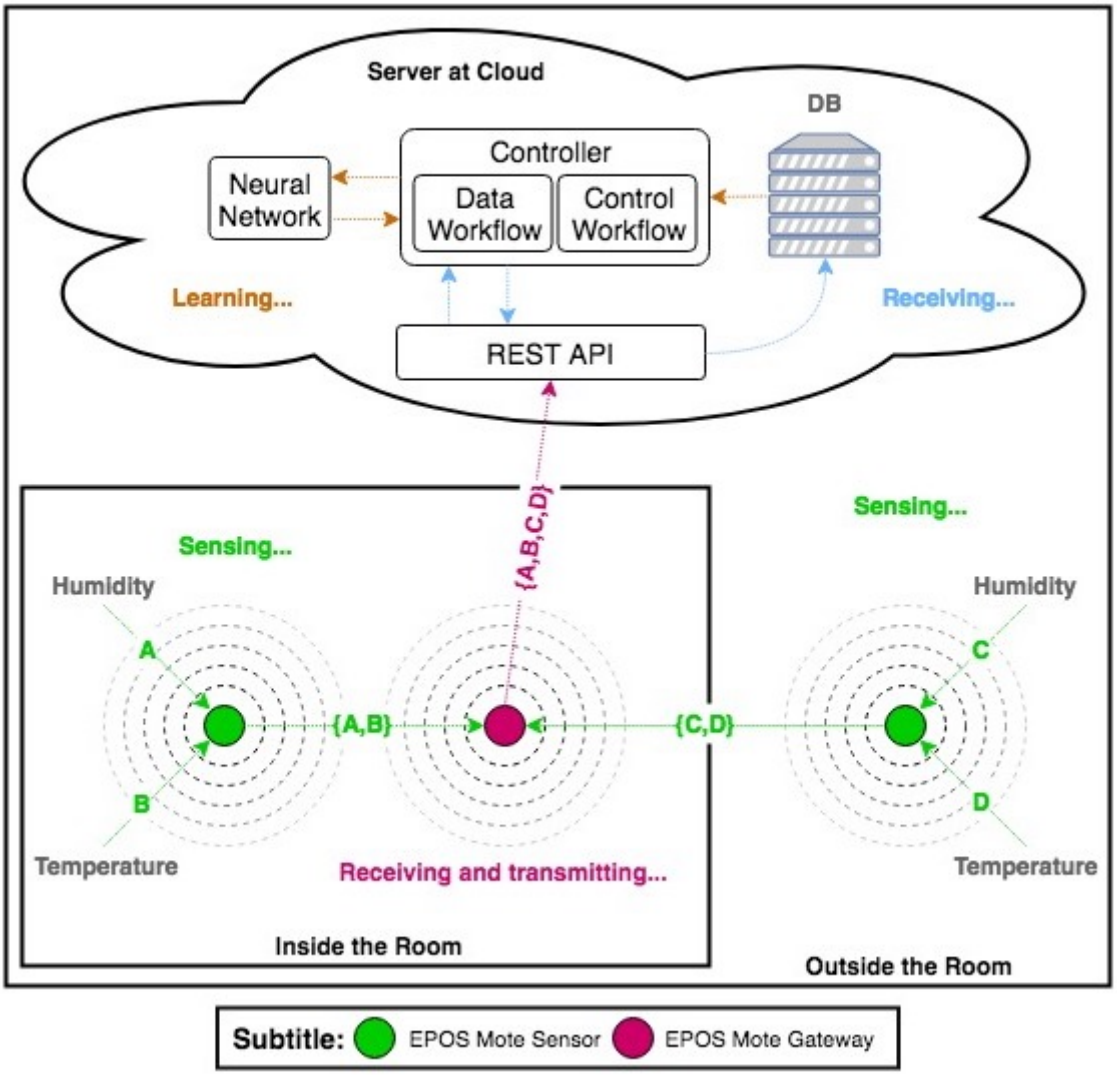


Figura 1: ilustra os principais componentes do servidor, o processo de coleta de dados do ambiente (tanto dentro quanto fora) e a transmissão desses dados para o servidor.

O controlador do servidor, foco deste trabalho, é responsável por organizar os dados recebidos pelo gateway, os quais são inicialmente armazenados em uma cache de dados. Além disso, quando o usuário está no ambiente, o mesmo pode realizar ajustes no ambiente e, dessa maneira, serão enviados comandos do usuário ao servidor. Neste caso, o controlador também é responsável por receber e organizar os comandos, armazenando-os em uma cache de comandos.

Sempre que a quantidade de dados recebidos for relevante ou o usuário realizou algum ajuste no ambiente (alterou as configurações preditas através do modelo da rede neural), o controlador solicitará à rede neural que atualize o modelo de predição, passando à ela os dados das caches. A Figura 2 ilustra esse cenário.

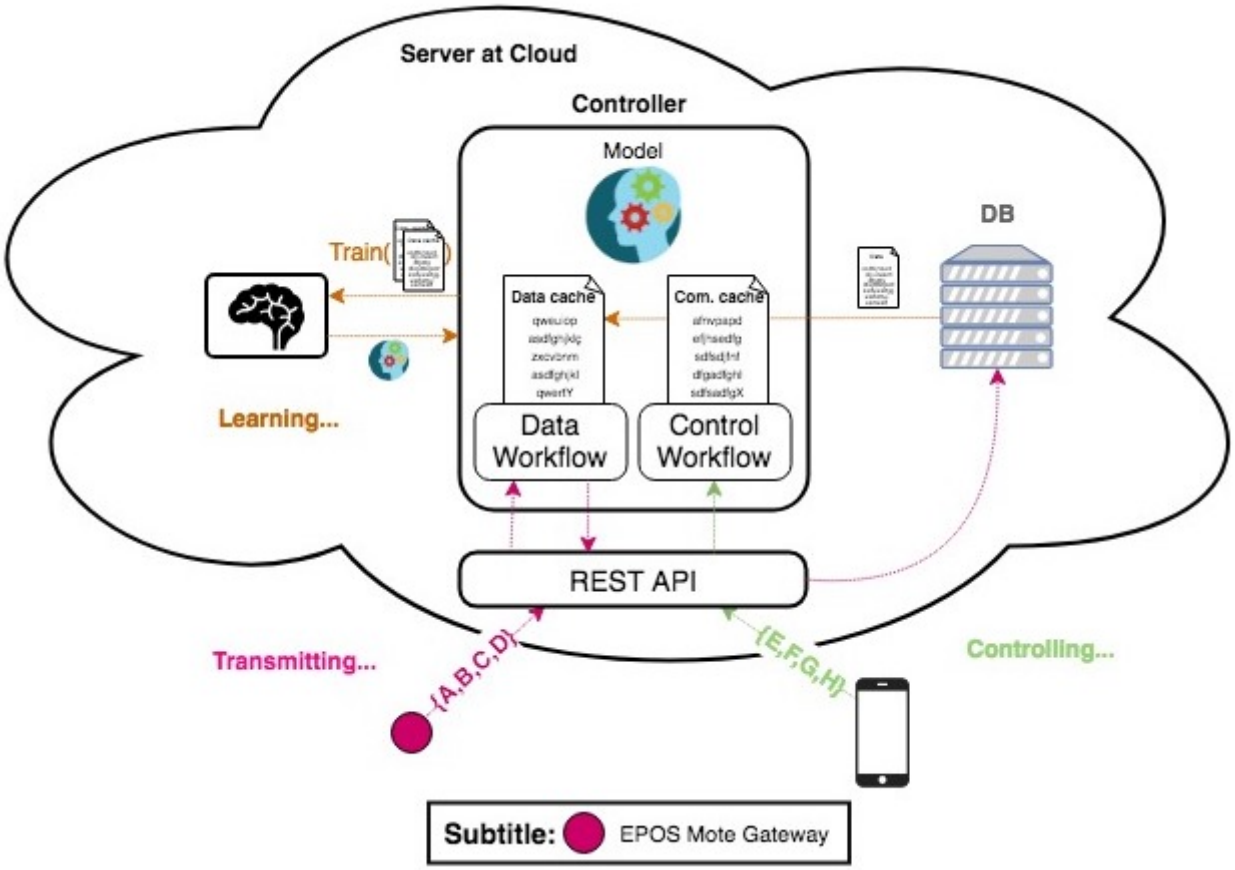


Figura 2: ilustra o processo de armazenamento de dados e comandos de usuários nas caches de dados e comandos, respectivamente, assim como a solicitação de retreinamento da rede neural para atualização do modelo de predição.

A rede neural, implementada através da biblioteca WEKA, além de realizar uma pré-filtragem dos dados através de um algoritmo de feature selection, tem a principal função de aprender com os dados e comandos. Este processo, realizado através de redes neurais, consiste em avaliar os dados coletados para encontrar relações entre eles. Estas relações, por sua vez, permitirá determinar as condições do ambiente que mais agradam o usuário quando o tempo local apresentar determinadas características (por exemplo, sempre que a temperatura do tempo local atinge 30º celsius o usuário regula a temperatura do ar condicionado em 24º celsius).

Sempre que solicitado pelo controlador (quando houver uma quantidade razoável de dados novos ou quando o usuário realizar algum ajuste), a rede neural receberá as informações necessárias para que ela possa determinar as características que agradam o usuário, levando em consideração a dinamicidade do tempo local e, como resposta a rede neural devolverá um modelo de predição ao controlador. Este padrão serve unicamente para informar como o ambiente deve ser configurado para melhorar o conforto do usuário. Neste sentido, o controlador utilizará as condições atuais do ambiente e fora dele para gerar as configurações do ambiente, as quais serão enviadas à aplicação no smartphone do usuário. Este processo é ilustrado na Figura 3.

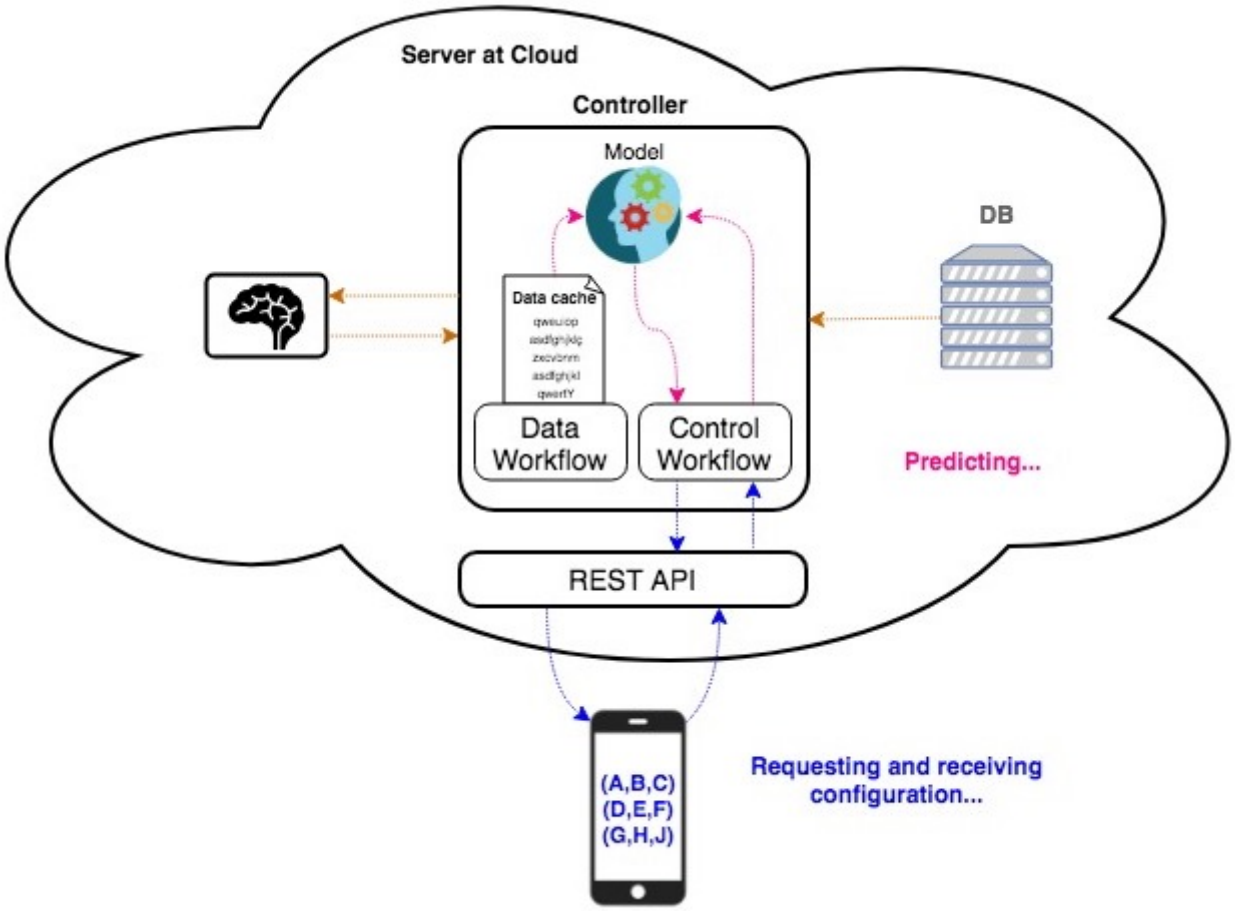


Figura 3: ilustra o processo de predição das configurações ideais ao usuário através do uso dos dados atuais do ambiente em que ele se encontra. A aplicação no smartphone mostra as configurações do ambiente.

Para visualização e controle do usuário, será desenvolvido uma aplicação que permitirá visualizar um histórico dos dados coletados ao longo do tempo. Ainda, ao entrar no ambiente, o usuário receberá em seu smartphone as configurações ideais do local. Por não termos nenhum dispositivo atuador, a configuração será realizada manualmente pelo usuário e caso ele queira realizar algum ajuste, a aplicação será responsável por informar o servidor sobre esses ajustes.

a. Contextualização dos dados

A forma de contextualização do ambiente será feita de forma semelhante que {3}, ou seja, com base nas grandezas físicas passíveis de serem monitoradas, por exemplo, temperatura, umidade, velocidade do vento, grau de luminosidade etc. Dessa forma, ao treinar redes neurais distintas para cada um dos dispositivos que monitoram tal grandeza, serão capturadas as condições ideais para um determinado usuário. Porém, diferente de {3} este trabalho usará plataformas, bibliotecas e algoritmos diferentes, buscando uma implementação mais simplificada e eficaz.

Os dados contextuais que serão monitorados neste trabalho são:

- **Usuário:** qual usuário está no ambiente.
- **Temperatura interna:** temperatura atual dentro da sala.
- **Umidade interna:** umidade atual dentro da sala.
- **Temperatura externa:** temperatura atual fora da sala.
- **Umidade externa:** umidade atual fora da sala.

- **Minuto:** minuto em que o contexto foi capturado.
- **Hora:** hora em que o contexto foi capturado.
- **Dia:** dia em que o contexto foi capturado.
- **Dia da semana:** dia da semana em que o contexto foi capturado.

b. Plataforma IoT

Como base para comunicação, armazenamento e execução necessários para o nosso projeto, será utilizado a plataforma IoT do Lisha, onde será implantado um processo daemon que realizará a manipulação dos diferentes fluxos de dados recebidos, o controle das caches e o aprendizado do contexto como comentado nas seções anteriores.

c. REST API

Todas as comunicações serão realizadas por intermédio da interface de comunicação REST API presente na plataforma IoT do Lisha (http://epos.lisha.ufsc.br/IoT+with+EPOS#iot.lisha.ufsc.br_REST_API). Tal interface disponibiliza regras bem definidas de inserção (create, attach e put) e recuperação (get) dos dados coletados no ambiente de estudo.

Padrão de representação dos dados

A representação dos dados utiliza o padrão JSON, é mostrada a seguir:

```
{
        "smartdata" : Array
        [
            {
                "version" : string
                "unit" : unsigned int
                "value" : double
                "error" : unsigned int
                "confidence" : unsigned int
                "x" : int
                "y" : int
                "z" : int
                "t" : unsigned int
                "dev" : unsigned int
                "signature" : unsigned int
            }
        ]
        "credentials" : Object
        {
            "domain" : string
            "username" : string
            "password" : string
        }
        "workflow" : unsigned
}
```

Autenticação da comunicação

Um certificado digital foi emitido pela equipe do Lisha, junto com uma chave criptográfica, para permitir a comunicação segura e confiável do gateway e outros dispositivos com a plataforma IoT. Assim, esse par, chave e certificado, estão atrelados ao domínio associados ao nosso grupo de trabalho, permitindo a comunicação, a criação e manipulação do banco de dados, dispensando o usuário e senha presentes nas credenciais.

d. Fluxos de dados, controle e configuração

O processo de comunicação do sistema como um todo pode ser dividido em dois fluxos de informações, basicamente. O primeiro, referente ao fluxo de dados coletados pelos sensores no ambiente, que são e enviados através do gateway para a REST API. O segundo, por sua vez, é referente ao fluxo de comandos enviados pela aplicação mobile do usuário. Ambos os fluxos de informações terá associado um script, campo workflow no JSON, que passará ao nosso controlador (daemon) o dado recebido para que sejam devidamente tratados.

O controlador possui 3 estados diferentes de execução, stopped, idle e activated. Ele iniciará em stopped até que se auto-configure e comece a receber solicitações. Após a configuração, o controlador passará ao estado idle, onde aguardará o reconhecimento do usuário podendo voltar ao estado stopped caso for solicitado o seu desligamento. Após a identificação do usuário no ambiente, o controlador irá para o estado activated, neste estado ele estará apto a receber e processar as mensagens. Caso o usuário saia do ambiente, uma mensagem levará o controlador novamente ao estado idle.

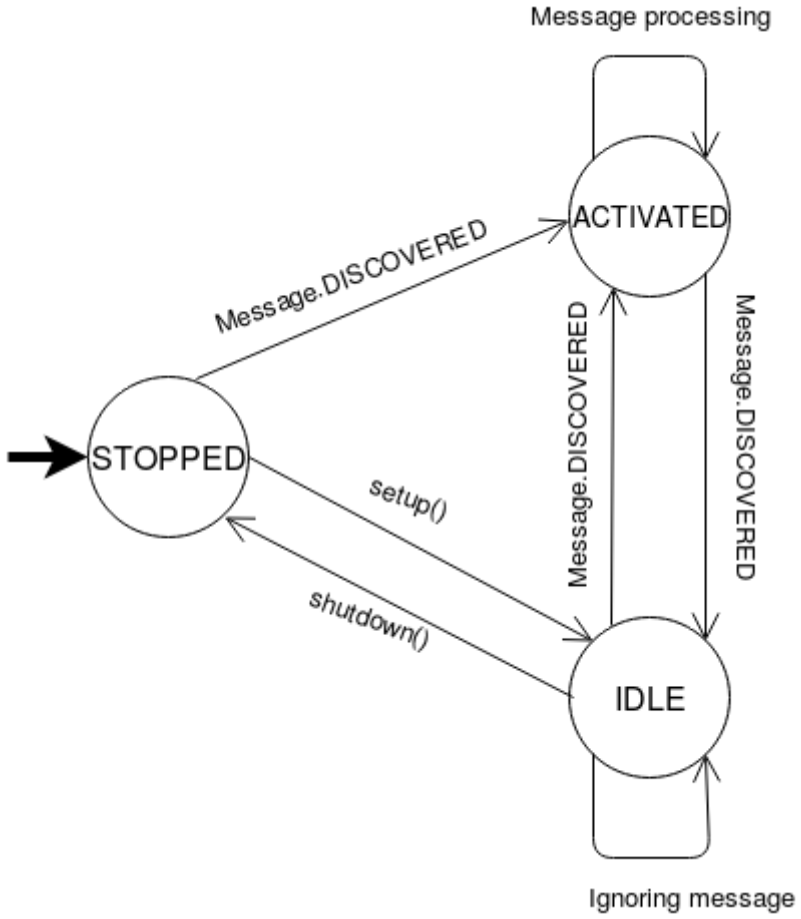


Diagrama 0: Representação dos estados do controlador.

Sempre que um dado coletado chega através de uma solicitação put da API, o campo workflow do JSON informará o fluxo de dados coletados (data workflow) para ser executado. Assim como mostra o Diagrama 1, o fluxo de comunicação de dados inicia com o gateway e finaliza com a confirmação da operação pela plataforma IoT. Vale ressaltar que caso o controlador não esteja inicializado quando necessário, será realizado um procedimento para tal.

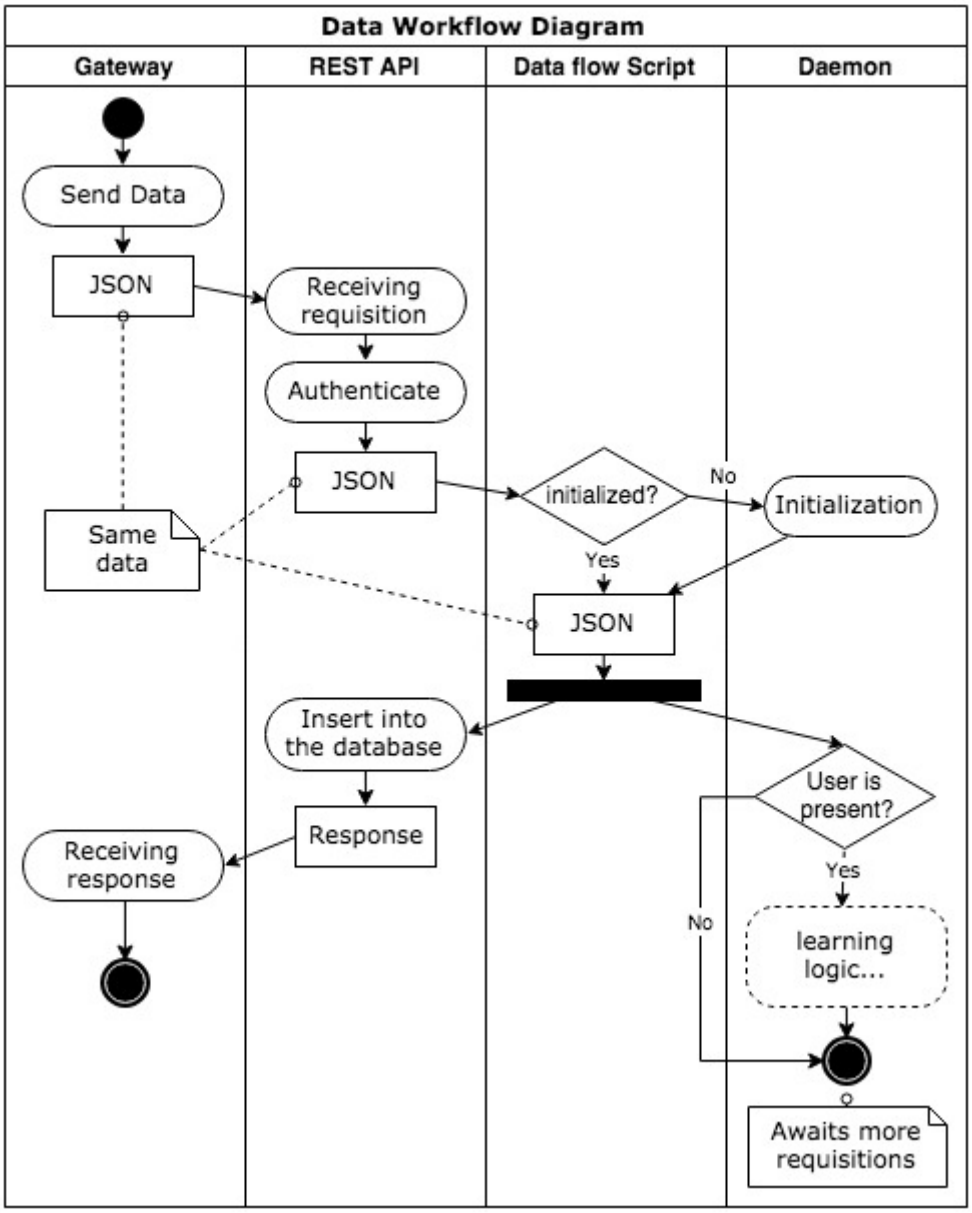


Diagrama 1: simplificação do fluxo de dados coletados desde o gateway até o controlador e banco de dados IoT do Lisha.

O Diagrama 2 exemplifica a passagem da solicitação à thread auxiliar para realização da política de cache de dados implementada. De forma resumida, a lógica de aprendizado buscará criar instâncias que representam um contexto de período de tempo do usuário e realizando o aprendizado a partir de uma quantidade mínima de instâncias.

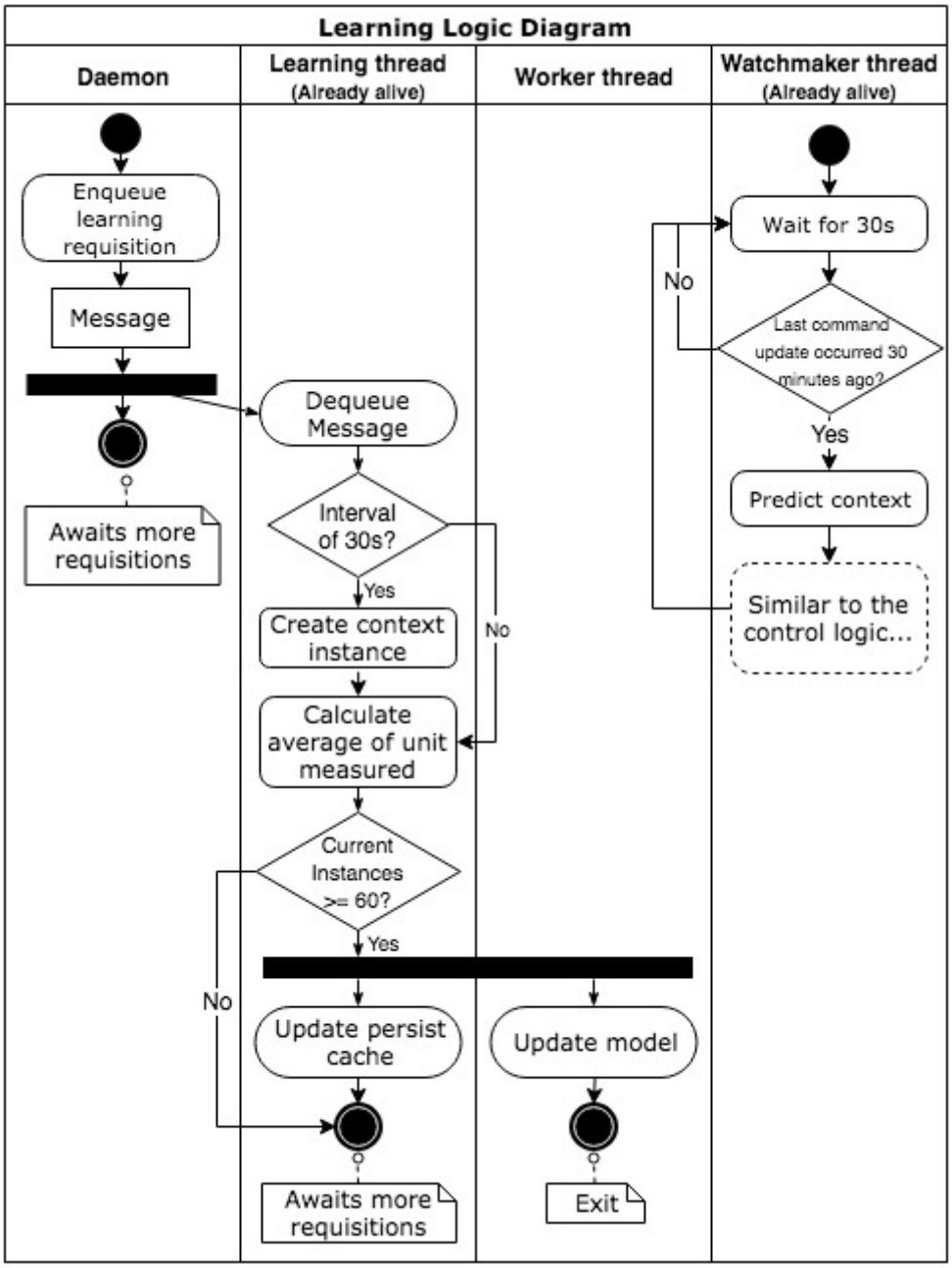


Diagrama 2: Fluxo de dados dentro do contexto da thread Learning..

O fluxo de controle acontece de forma semelhante ao de dados, diferenciando-se após a passagem do dado ao controlador, o Diagrama 3 exemplifica esse processo. Ao receber um comando, ou seja, uma configuração desejada pelo usuário, o controlador dá início a atualização da cache de comando. Após a realização das ações necessárias, o dado de controle não é passado adiante, afinal, tais dados ocorrem em menor quantidade e não necessitam ser armazenados no banco de dados pois não são coletados do ambiente.

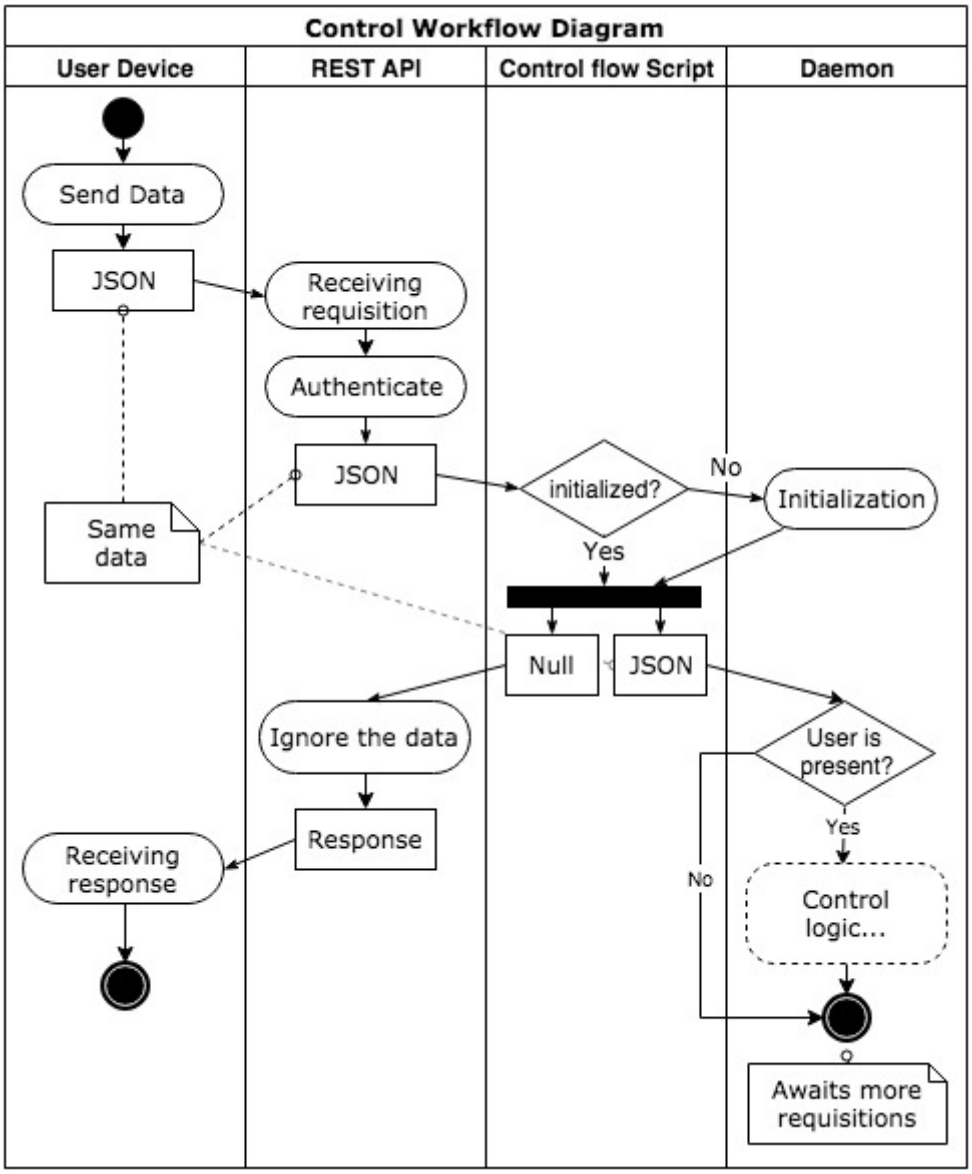


Diagrama 3: simplificação do fluxo de comandos do usuário coletados desde a aplicação mobile até o controlador e banco de dados IoT do Lisha.

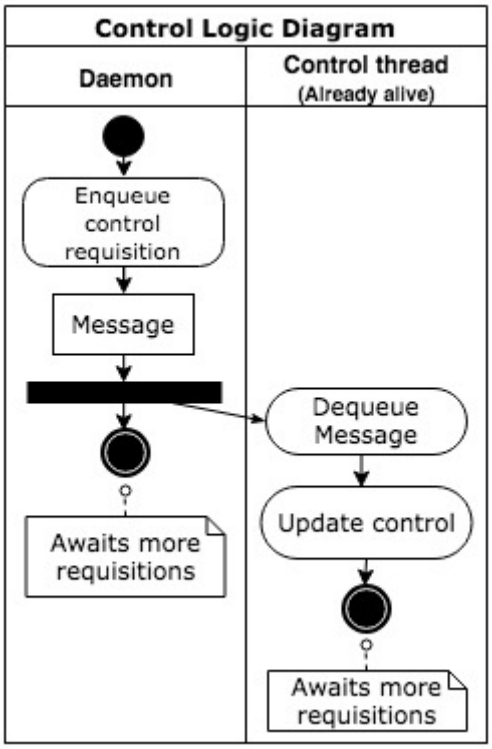


Diagrama 4: Atualização da cache de comandos..

A requisição da configuração ideal para o ambiente também será realizada através da REST API utilizando uma operação get personalizada e, de forma similar ao fluxo de comandos, executar um script que construa a configuração ideal em JSON para que a API a encaminhe ao solicitante.

e. Seleção de dados

O processo de aprendizagem de contexto ocorrerá em duas etapas distintas. A primeira, responsável por filtrar as variáveis do conjunto observável (feature selection), será feita através de um algoritmo de seleção de atributos relevantes, resultando num subconjunto de variáveis que têm alta significância na predição de uma determinada condição do ambiente. O algoritmo será aplicado sobre uma parcela dos dados, os quais pertencem às caches de dados e últimos comandos do usuário. Mais detalhes estão na seção de desenvolvimento.

f. Aprendizagem

Após a etapa de feature selection é realizada a preparação dos parâmetros de testes e o treino da rede neural, assim como a execução do algoritmo de deep learning para de fato a treinar. De modo similar a filtragem, o aprendizado utilizará uma parcela dos dados e configurações realizadas pelo usuário, os quais estão armazenados nas caches. Caso o usuário não modifique uma predição feita então ela será vista como uma configuração do usuário. Mais detalhes estão na seção de desenvolvimento.

Tecnologias

- **Hardware**
  - **Epos Mote III Project - V2.0**
    - **Sensores:** para realizar a captura da temperatura e umidade do ambiente serão utilizados dois EPOS Mote III (um dentro do ambiente e outro fora), uma vez que ele possui um sensor Silicon Labs Si7020 com alta precisão e exatidão.
    - **Gateway:** assim como para realizar a captura de dados, será utilizado um EPOS Mote III como gateway na arquitetura. Este por sua vez estará ligado à um computador desktop com comunicação via rede com o servidor de dados.
  - **Computador desktop i7**
    - Será utilizado um computador com processador i7 com um EPOS Mote III conectado via USB que será responsável por capturar a temperatura e umidade dentro do ambiente e enviar as coletas ao gateway e um outro EPOS Mote III que será o responsável por receber os dados dos sensores e enviá-los ao servidor.
  - **Computador desktop i7**
    - Será utilizado um computador com processador i7 com um EPOS Mote III conectado via USB que será responsável por capturar a temperatura e umidade dentro do ambiente e enviar as coletas ao gateway e um outro EPOS Mote III que será o responsável por receber os dados dos sensores e enviá-los ao servidor.
  - **Servidor do Lisha**
    - Para executar o controlador (daemon), aplicação de aprendizagem e o banco de dados será utilizado o servidor do Lisha com toda a infraestrutura IoT já implementada e em funcionamento.

- **Smartphone Android**
    - Será utilizado um smartphone LG para executar a aplicação de monitoramento da coleta de dados e recebimento das configurações ideais do ambiente.

- **Software**
    - **Servidor**
        - Será utilizado a REST API disponibilizada pelo servidor de dados do Lisha para armazenamento das variáveis observadas. Para configuração de acesso, será necessário a criação de um único usuário dentro de um único domínio.
    - **Algoritmo de identificação do usuário**
        - O gateway realizará uma varredura na rede local verificando a existência de um usuário conhecido pelo endereço MAC do smartphone..
    - **Algoritmo para coleta de dados**
        - Coleta dos dados será realizada utilizando o padrão smart data visto em aula.
    - **Modelagem dos dados para armazenamento**
        - Será utilizada a estrutura JSON especificada pela REST API e o padrão SI do smart data, de modo a integrar com a modelagem de séries temporais implementado pelo banco de dados Cassandra.
    - **Comunicação**
        - Serão utilizadas bibliotecas Java baseado em REST, JSON, HTTP e TCP/UDP para estabelecimento de comunicações entre o gateway, usuário e servidor.
    - **Algoritmo de feature selection**
        - Será utilizado o algoritmo de correlação através da biblioteca WEKA para filtragem das variáveis relevantes para o aprendizado. O algoritmo correlation based feature selection do WEKA utiliza coeficientes de Pearson para calcular a correlação entre cada variável e a saída esperada. São selecionadas apenas as variáveis que possuem correlação positiva ou negativa de níveis moderado a alto.
    - **Algoritmo de aprendizagem de máquina**
        - Será utilizado extensões da biblioteca WEKA (SGD e MultilayerPercepton, WekaDeeplearning4J etc) para realizar o aprendizado utilizando uma técnica de deep learning e redes neurais.
    - **Aplicação para android**
        - Será utilizado a IDE Android Studio para o desenvolvimento de um aplicativo para dispositivos android que permitirá o usuário visualizar as configurações ideais do ambiente assim como monitorar a coleta de dados.

Análise de Viabilidade

Objetivo 1

Os exercícios de coleta de dados utilizando smart data realizados em aula já são suficientes para cumprir com esse objetivo.

Objetivo 2

Em {3} o essencial é apresentado de forma superficial, mesmo assim será suficiente para guiar a

implementação neste trabalho. Baseando-se no trabalho desenvolvido em {3}, foi desenvolvido e testado um script bash (cujo código é mostrado abaixo) que identifica um usuário na rede local através do endereço MAC do seu dispositivo móvel, a Figura 4 mostra o resultado da execução do script.

```
#!/bin/bash
# References: https://epos.lisha.ufsc.br/Prescient
# MacOS version
MAC_ADDR="$1"

if [ "$#" -ne 1 ]; then
     exit 1
else
     echo "$(nmap -sn 192.168.25.2/24
          && arp -a
          | grep -B2 -i ${MAC_ADDR} --line-buffered
          | awk -v var=${MAC_ADDR} '{if($4==var)print substr($2,2,length($2)-2)}')"
fi
```

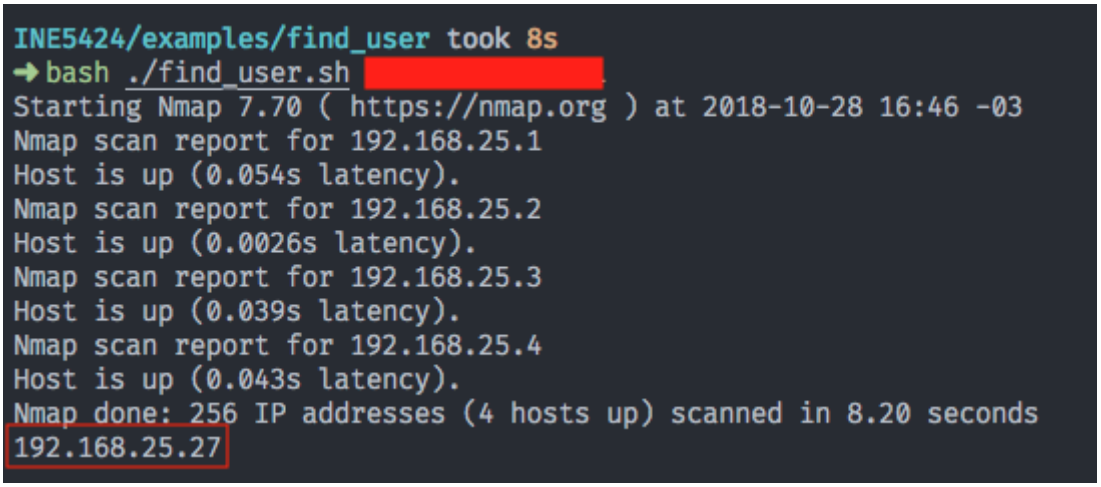


Figura 4: resultado da execução do script acima mencionado.

Objetivo 3

A plataforma IoT do Lisha já está completa e operando, basta apenas a criação do usuário e ambiente para que o controlador seja implementado. O código abaixo mostra um exemplo da criação de uma série temporal através da REST API e a inserção de um dado na plataforma IoT do Lisha utilizando o certificado gerado para nosso projeto. A Figura 5 mostra o resultado da execução do algoritmo abaixo.

```
#!/usr/bin/env python3
import time, requests, json

put_url ='https://iot.lisha.ufsc.br/api/put.php'
attach_url ='https://iot.lisha.ufsc.br/api/attach.php'
CLIENT_CERTIFICATE = ['client-xx-xxxxx.pem', 'client-xx-xxxxx.key']

attach_query = {
   'series': {
      'version': 1.1,
      't0': 0,
      't1':1600483896157363,
```

```
      'unit': 2224179556,
      'dev': 0,
      'r':2000,
      'y': 302,
      'x': 302,
      'z': 0
   },
   'credentials': {
      'domain': 'grupo2'
   }
}
query = {
   'smartdata': [ {
      'version': 1.1,
      'confidence': 0,
      'time': 0,
      'unit': 2224179556,
      'error': 0,
      'dev': 0,
      'y': 1000,
      'x': 1000,
      'z': 0,
      'value': 26,
      'mac': 0
   } ],
   'credentials': {
      'domain': 'grupo2'
   }
}

session = requests.Session()
session.cert = CLIENT_CERTIFICATE
session.headers = {'Content-type' : 'application/json'}
response = session.post(attach_url, json.dumps(attach_query))
print("Attach [", str(response.status_code), "]", sep='')

if response.status_code == 204:
   print('Attach: OK!')
else:
    print("Attach: Failed!")

response = session.post(put_url, json.dumps(query))
print("Put [", str(response.status_code), "]", sep='')

if response.status_code == 204:
   print('Put: OK!\n')
   print(json.dumps(query, indent=4, sort_keys=False))
else:
    print("Put: Failed!")
```

```
~/code/Credentials
➜ python3 put_example.py
Attach [204]
Attach: OK!
Put [204]
Put: OK!

{
    "smartdata": [
        {
            "version": 1.1,
            "confidence": 0,
            "time": 0,
            "unit": 2224179556,
            "error": 0,
            "dev": 0,
            "y": 1000,
            "x": 1000,
            "z": 0,
            "value": 26,
            "mac": 0
        }
    ],
    "credentials": {
        "domain": "grupo2"
    }
}
```

Figura 5: resultado da execução do algoritmo acima.

O código abaixo mostra como é realizada a recuperação (através da REST API) dos dados inseridos no exemplo anterior. A Figura 6 mostra o resultado da execução desse código.

```
#!/usr/bin/env python3
import time, requests, json

get_url = 'https://iot.ufsc.br/api/get.php'
CLIENT_CERTIFICATE = ['client-xx-xxxxx.pem', 'client-xx-xxxxx.key']

query = {
       'series' : {
              'version' : 1.1,
              'unit' : 2224179556,
              'x' : 1000,
              'y' : 1000, 'z' : 0,
              'r' : 0,
              't0' : 0,
              't1' : 0,
              'dev' : 0
       },
       'credentials' : {
              'domain' : 'grupo2'
       }
}

session = requests.Session()
session.cert = CLIENT_CERTIFICATE
```

```
session.headers = {'Content-type' : 'application/json'}
response = session.post(get_url, json.dumps(query))

print("Get [", str(response.status_code), "]", sep='')
if response.status_code == 200:
   print("Get: OK!\n")
   print(json.dumps(response.json(), indent=4, sort_keys=False))
else:
    print("Get: Failed!")
```

```
~/code/Credentials
➜ python3 get_example.py
Get [200]
Get: OK!

{
    "series": [
        {
            "version": 17,
            "unit": 2224179556,
            "timestamp": 0,
            "value": 26,
            "error": 0,
            "confidence": 0,
            "x": 1000,
            "y": 1000,
            "z": 0,
            "dev": 0
        }
    ]
}
```

Figura 6: resultado da execução do algoritmo acima.

Ainda no escopo do objetivo 3, para a simulação da plataforma IoT, base para implementação e testes do controlador, foram criados quatro programas, cada um para simular os componentes da arquitetura de comunicação mostrada no Diagrama 1.

O gateway, neste caso, é uma simples aplicação em python que se conecta ao servidor através de um socket vinculado ao endereço localhost e porta 8000, onde são enviados dados no formato JSON e esperada uma mensagem de confirmação. O código do gateway é mostrada abaixo e o resultado de sua execução na Figura 7.

```
import json, socket

host = '0.0.0.0'
port = 8000

// Cria os dados para envio

querys_list = [
     {'series' : {'version' : 1.1, 'unit': 2224979500, 'x': 0, 'y': 0, 'z': 0, 'r': 30000,
                        't0': 9540629458393590, 't1': 9540629468393594, 'dev': 0},
      'credentials' : { 'domain' : 'grupo2'}
```

```
    },
    {'series' : {'version' : 2.2, 'unit': 2224979500, 'x': 1,'y': 1,'z': 1,'r': 30000,
                 't0': 9540629458393590, 't1': 9540629468393594, 'dev': 0},
    'credentials' : { 'domain' : 'grupo2'}
    },
    {'series' : {'version' : 3.3, 'unit': 2224979500, 'x': 2, 'y': 2, 'z': 2, 'r': 30000,
                 't0': 9540629458393590, 't1': 9540629468393594, 'dev': 0},
    'credentials' : { 'domain' : 'grupo2'}
    }
]

// Envia os dado em JSON codificados como uma string
for query in querys_list:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((host, port))
        sock.sendall(str.encode(json.dumps(query)))
        print(sock.recv(1024))
        print('\n')
        sock.close()
```

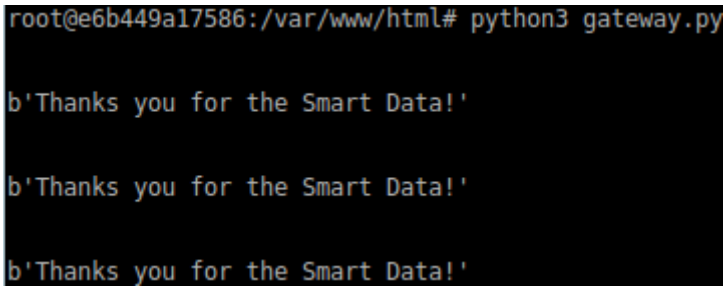


Figura 7: resultado da execução do algoritmo acima. Envio de dados ao servidor.

Para a simulação do comportamento do servidor IoT do Lisha foi criado um simples servidor PHP em localhost:8000. Tal servidor é responsável por aguardar dados enviados pelo gateway, executar um workflow passando o dado recebido do gateway e imprimir uma mensagem como se o dado tivesse sido armazenado em um banco de dados. O código do servidor PHP é mostrada abaixo e o resultado de sua execução na Figura 8.

```
<?php
    // Cria um servidor vinculado ao endereço e porta fornecidos
    $serv = stream_socket_server("tcp://0.0.0.0:8000", $errno, $errstr) or die("Create
Server Failed!");
    // Define o tempo limite de espera para operar o recurso fornecido
    stream_set_timeout($serv, 60)
    echo "\n\nWaiting requisitions...\n";
    while ($serv) {
        // Cria uma conexão com o servidor especificado
        $conn = stream_socket_accept($serv);
        while (feof($conn))
                sleep(10);
        // Lê os dado do socket
        $data = fread($conn, 2048);
        $str = sprintf('python3 workflow.py "%s" > /dev/tty', $data);
        echo "\nServer received: " . $data;
```

```
        echo "\nData sent to Workflow\n\n";
        // Executa o workflow.py
        exec($str, $ret);
        for ($i = 0; $i < count($ret); $i++)
                echo $ret[$i];
        echo "Data inserted on Data Base\n\n";
        // Responde ao gateway e fecha a conexão com o servidor especificado
        fwrite($conn, "Thanks you for the Smart Data!", 2048);
        fclose($conn);
        echo "\nWaiting more requisitions...\n";
    }
?>
```

```
root@e6b449a17586:/var/www/html# php server.php


Waiting requisitions...

Server received: {"credentials": {"domain": "grupo2"}, "series": {"z": 0, "dev": 0, "r": 30000, "version": 1.1, "t0": 9540629458393590, "x": 0, "y": 0, "unit": 2224979500, "t1": 9540629468393594}}
Data sent to Workflow

### Workflow Initialization ###

Data received: {credentials: {domain: grupo2}, series: {z: 0, dev: 0, r: 30000, version: 1.1, t0: 9540629458393590, x: 0, y: 0, unit: 2224979500, t1: 9540629468393594}}
Data sent to Controller

### Workflow Finalization ###

Data inserted on Data Base


Waiting more requisitions...

Server received: {"credentials": {"domain": "grupo2"}, "series": {"z": 1, "dev": 0, "r": 30000, "version": 2.2, "t0": 9540629458393590, "x": 1, "y": 1, "unit": 2224979500, "t1": 9540629468393594}}
Data sent to Workflow

### Workflow Initialization ###

Data received: {credentials: {domain: grupo2}, series: {z: 1, dev: 0, r: 30000, version: 2.2, t0: 9540629458393590, x: 1, y: 1, unit: 2224979500, t1: 9540629468393594}}
Data sent to Controller

### Workflow Finalization ###

Data inserted on Data Base


Waiting more requisitions...

Server received: {"credentials": {"domain": "grupo2"}, "series": {"z": 2, "dev": 0, "r": 30000, "version": 3.3, "t0": 9540629458393590, "x": 2, "y": 2, "unit": 2224979500, "t1": 9540629468393594}}
Data sent to Workflow

### Workflow Initialization ###

Data received: {credentials: {domain: grupo2}, series: {z: 2, dev: 0, r: 30000, version: 3.3, t0: 9540629458393590, x: 2, y: 2, unit: 2224979500, t1: 9540629468393594}}
Data sent to Controller

### Workflow Finalization ###

Data inserted on Data Base


Waiting more requisitions...
```

Figura 8: resultado da execução do algoritmo acima. Recebimento de dados do gateway, execução do workflow e armazenamento no banco de dados (Workflow Initialization e Finalization mostra o resultado da execução do workflow).

Para simular o comportamento do workflow, foi implementado um simples script em python, ficando a cargo dele apenas o redirecionamento do dado recebido para a aplicação do controlador através de um named pipe. O código do workflow é mostrada abaixo e o resultado de sua execução na Figura 9.

```
import os, sys

print("### Workflow Initialization ###\n")

FIFO = 'mypipe'
data = sys.argv[1]

print("Data received: " + data)
    // Criação do named pipe
try:
    os.mkfifo(FIFO)
```

```
except OSError:
    x = "Nothing"

// Abertura do named pipe
pipe = open(FIFO, "w")
// Escrita de dados
pipe.write(data)

print("Data sent to Controller\n")
print("### Workflow Finalization ###\n")

pipe.close()
```

```
### Workflow Initialization ###

Data received: {credentials: {domain: grupo2}, series: {z: 0, dev: 0, r: 30000, version: 1.1, t0: 9540629458393590, x: 0, y: 0, unit: 2224979500, t1: 9540629468393594}}
Data sent to Controller

### Workflow Finalization ###
```

Figura 9: resultado da execução do algoritmo acima. Execução do workflow e envio via named pipe para o controlador.

Por fim, para simular o controlador uma aplicação em python simplesmente permanece aguardando dados através da leitura do named pipe e quando o obtém, imprime o dado, imprime uma mensagem simulando o processo de tratamento dele e por fim voltar a aguardar mais dados. O código do controller é mostrada abaixo e o resultado de sua execução na Figura 10.

```
import os, sys

print("### Controller Initialization ###\n\n")
FIFO = 'mypipe'

try:
    os.mkfifo(FIFO)
except OSError:
    x = "Nothing"

pipe = open(FIFO, "r")
print("Waiting more requisitions...\n")

// Leitura dos dados através do named pipe
while True:
    data = pipe.read()
    if (data):
        print('Received data: "{0}"'.format(data) + "\n")
        print("Treating data...\n\n")
        print("Waiting more requisitions...\n")

pipe.close()
```

```
root@e6b449a17586:/var/www/html# python3 Controller.py
### Controller Initialization ###


Waiting more requisitions...

Received data: "{series: {version: 1.1, z: 0, x: 0, r: 30000, t1: 9540629468393594, y: 0, unit: 2224979500, t0: 9540629458393590, dev: 0}, credentials: {domain: grupo2}}"

Treating data...


Waiting more requisitions...

Received data: "{series: {version: 1.1, x: 0, t0: 9540629458393590, y: 0, r: 30000, unit: 2224979500, z: 0, dev: 0, t1: 9540629468393594}, credentials: {domain: grupo2}}"

Treating data...


Waiting more requisitions...

Received data: "{series: {dev: 0, version: 1.1, r: 30000, y: 0, x: 0, t0: 9540629458393590, t1: 9540629468393594, unit: 2224979500, z: 0}, credentials: {domain: grupo2}}"

Treating data...


Waiting more requisitions...
```

Figura 10: resultado da execução do algoritmo acima. Recebimento de dados através do named pipe e impressão desses dados.

Objetivo 3.a

Esse objetivo será alcançado assim que cumpridos os objetivos 3.b e 3.c.

Objetivo 3.b e 3.c

Implementar caches de dados e comandos são bem parecidas a princípio, de forma que somente a primeira demandará maiores esforços. Ainda assim, existem diversos métodos para armazenamento de dados temporários e algoritmos prontos disponíveis na internet, tais como o MariaDB, um banco de dados robusto e adaptável (basicamente um aprimoramento do MYSQL), ou simplesmente um arquivo binário, que seria a opção mais simples e funcional.

Objetivo 3.d e 4

Com base em um exemplo fornecido pelo próprio Weka e através de sua interface GUI, um conjunto de dados é filtrado e dá origem a um modelo de predição. O exemplo faz uso do algoritmo de construção e treinamento de redes neurais denominado MultilayerPerceptron. Esse algoritmo, disponível na biblioteca Weka, pode ser utilizado direto em um código java ou executado a partir do terminal.

As Figuras 11 e 12 mostram a representação de uma rede neural e o arquivo de entrada para treinamento e geração do modelo, respectivamente. No lado esquerdo do modelo estão os atributos de entrada (retângulo verde) e no direito a saída (retângulo amarelo). Os nós intermediários são os neurônios da rede neural e as arestas são a sinapses (os pesos não são apresentados). Esse modelo tem como objetivo prever a idade de uma pessoa utilizando certas características, tais como o nível de glicose no sangue, pressão arterial, massa corporal, entre outros. Uma rede neural como esta pode ajudar médicos a prever se determinadas doenças, como hipertensão arterial, têm relação com a idade.

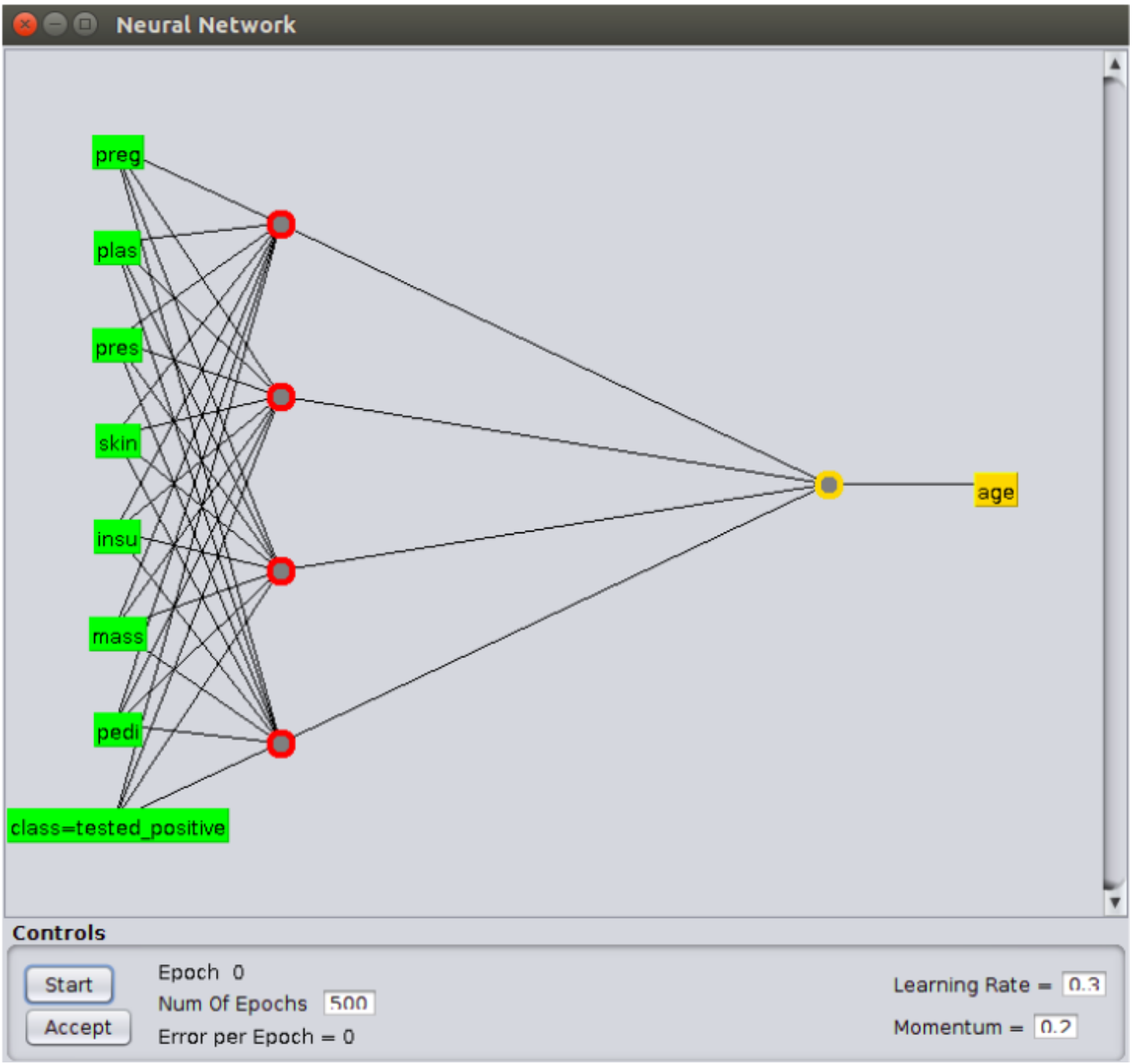


Figura 11: ilustra uma rede neural após a sua construção e treinamento. O aprendizado foi realizado através do algoritmo MultilayerPerceptron.

```
1 @relation pima_diabetes
2 @attribute 'preg' numeric
3 @attribute 'plas' numeric
4 @attribute 'pres' numeric
5 @attribute 'skin' numeric
6 @attribute 'insu' numeric
7 @attribute 'mass' numeric
8 @attribute 'pedi' numeric
9 @attribute 'age' numeric
10 @attribute 'class' { tested_negative, tested_positive}
11 @data
12 6,148,72,35,0,33.6,0.627,50,tested_positive
13 1,85,66,29,0,26.6,0.351,31,tested_negative
14 8,183,64,0,0,23.3,0.672,32,tested_positive
15 1,89,66,23,94,28.1,0.167,21,tested_negative
16 0,137,40,35,168,43.1,2.288,33,tested_positive
17 5,116,74,0,0,25.6,0.201,30,tested_negative
18 3,78,50,32,88,31,0.248,26,tested_positive
19 10,115,0,0,0,35.3,0.134,29,tested_negative
20 2,197,70,45,543,30.5,0.158,53,tested_positive
21 8,125,96,0,0,0,0.232,54,tested_positive
22 4,110,92,0,0,37.6,0.191,30,tested_negative
23 10,168,74,0,0,38,0.537,34,tested_positive
24 10,139,80,0,0,27.1,1.441,57,tested_negative
25 1,189,60,23,846,30.1,0.398,59,tested_positive
26 5,166,72,19,175,25.8,0.587,51,tested_positive
27 7,100,0,0,0,30,0.484,32,tested_positive
28 0,118,84,47,230,45.8,0.551,31,tested_positive
```

Figura 12: arquivo Attribute-Relation File Format (ARFF) (padrão de entrada do WEKA) utilizado para representação dos dados de entrada utilizados na geração do modelo de exemplo. Maiores detalhes podem ser vistos em 9.

Outro ponto importante que viabiliza esta etapa do trabalho é a detalhada documentação da biblioteca WEKA, a ampla variedade de algoritmos disponíveis nessa ferramenta e sua fácil integração com aplicações em Java. A utilização do algoritmo MultilayerPerceptron não é a melhor opção para um projeto que possui restrições impostas por sistemas remotos e de tempo real. Pois, esse algoritmo não é possível realizar o re-treinamento, ou seja, toda vez que chegam novos dados a rede neural anterior é descartada. Por essa razão, será utilizado o algoritmo SGD, no qual pode ser re-treinado.

Objetivo 3.e

O algoritmo é simples, basta apenas implementá-lo. Resume-se em chamar uma função da aplicação de aprendizagem passando os dados das caches de dados e comandos. Será necessário apenas determinar quando a cache de dados possui dados o suficiente para gerar um retreinamento da rede neural. Neste caso, estima-se que o re-treinamento ocorra quando a cache de dados tiver aproximadamente mil dados

(será realizado o treinamento a cada 13 ou 14 minutos aproximadamente) ou quando for realizado um comando do usuário.

Objetivo 3.f e 5

Devido a existência de diversos tutoriais e templates de aplicativos móveis e também à simples aplicação a ser desenvolvida, este objetivo torna-se passível de ser cumprido. Ainda, o processo de comunicação será realizado através dos protocolos TCP/UDP e IP já vistos e utilizados em trabalhos de redes de computadores, podendo ser realizado com base nos algoritmos implementados para validação do controlador.

Objetivo 6 (opcional)

Para realizar este objetivo será necessário desenvolver uma pequena extensão no servidor de dados e de aprendizagem, sendo necessária a criação de um usuário composto. A partir daí, basta aprender o seu contexto com base nos contextos pré-existentes de ambos os usuários.

Desenvolvimento

Esta seção tem como principal objetivo apresentar o desenvolvimento deste trabalho e os módulos que já estão implementados.

a. Sensores

Os dois sensores EPOS Mote III foram instalados no ECL (Laboratório de Computação Embarcada) do departamento INE, e configurados para que os dados coletados permaneçam válidos por 15 segundos. Esse período de tempo estipulado desconsidera a baixa variação da temperatura e umidade, buscando apenas uma maior precisão no monitoramento e distinção nos valores coletados. A implementação a seguir baseia-se nos tutoriais vistos em aula e exemplificam o monitoramento dos sensores no ambiente.

```
#include <smart_data.h>
#include <alarm.h>
#include <i2c.h>

using namespace EPOS;
OStream cout;

class ContextTemperatureSensor {
        …
        static I2C_Temperature_Sensor sensor;
        …
        static const unsigned int UNIT =
Unit::Get_Quantity<Unit::Temperature,Unit::F32>::UNIT;
        …
        static void sense(unsigned int dev, Smart_Data<ContextTemperatureSensor> * data) {
                data->_value = sensor.get();
                cout << "Serie: " << data->db_series() << endl;
                cout << "Record: " << data->db_record() << endl << endl;
        }
};
```

```
class ContextHumiditySensor {
        ...
        static I2C_Humidity_Sensor sensor;
        …
        static const unsigned int UNIT =
        Unit::Get_Quantity<Unit::Amount_of_Substance,Unit::F32>::UNIT;
        ...
};

typedef Smart_Data<ContextTemperatureSensor> TemperatureSensor;
typedef Smart_Data<ContextHumiditySensor> HumiditySensor;

I2C_Temperature_Sensor ContextTemperatureSensor::sensor;
I2C_Humidity_Sensor ContextHumiditySensor::sensor;

int main() {
        GPIO g('C', 3, GPIO::OUT);
        g.set(true);

        TemperatureSensor temperature(0, 15000000, TemperatureSensor::ADVERTISED);
        HumiditySensor humidity(0, 15000000, HumiditySensor::ADVERTISED);

        Thread::self()->suspend();
        return 0;
}
```

b. Gateway

O EPOS Mote III utilizado como gateway na arquitetura deste trabalho, é responsável pelo recebimento de dados coletados pelos sensores e envio destes a plataforma IoT do Lisha. O gateway foi instalado no mesmo ambiente que os sensores e configurado para receber e enviar dados com um período de 5 segundos. O curto período de tempo foi selecionado para que o controlador necessite lidar de forma otimizada com o grande volume de dados coletados. O código abaixo baseia-se nos exemplos e tutoriais vistos em aula e utiliza o protocolo implementado pela função print() para comunicar-se com a máquina host.

```
#include <smart_data.h>
#include <alarm.h>
#include <i2c.h>

using namespace EPOS;
USB io;

template<typename T>
class Printer: public Smart_Data_Common::Observer {
        …
        void update(Smart_Data_Common::Observed * obs) {
                print(_data->db_record());
        }
        template<typename D>
        void print(const D & d) {
                bool was_locked = CPU::int_disabled();
```

```
                 if(!was_locked)
                        CPU::int_disable();

                if(EQUAL<D, Smart_Data_Common::DB_Series>::Result)
                        io.put('S');
                else
                        io.put('R');

                for(unsigned int i = 0; i < sizeof(D); i++)
                        io.put(reinterpret_cast<const char *>(&d)[i]);
                for(unsigned int i = 0; i < 3; i++)
                        io.put('X');
                if(!was_locked)
                        CPU::int_enable();
                …
        }
};

class ContextTemperatureSensor { … }
class ContextHumiditySensor { … }

typedef Smart_Data<ContextTemperatureSensor> TemperatureSensor;
typedef Smart_Data<ContextHumiditySensor> HumiditySensor;

int main() {
        GPIO g('C', 3, GPIO::OUT);
        g.set(true);
        TSTP::Coordinates center(300, 300, 0);
        const TSTP::Time DATA_PERIOD = 5000000;
        const TSTP::Time DATA_EXPIRY = DATA_PERIOD;
        const TSTP::Time INTEREST_EXPIRY = 2ull * 1200000000;
        TSTP::Time start = TSTP::now();
        TSTP::Time end = start + INTEREST_EXPIRY;
        TSTP::Region region(center, 5000, start, end);
        TemperatureSensor temperature(region, DATA_EXPIRY, DATA_PERIOD);
        HumiditySensor humidity(region, DATA_EXPIRY, DATA_PERIOD);
        Printer<TemperatureSensor> temperaturePrinter(&temperature);
        Printer<HumiditySensor> humidityPrinter(&humidity);
        Thread::self()->suspend();
        return 0;
}
```

A comunicação do cliente com a plataforma IoT utiliza o script eposiotgw disponibilizado em aula. A Figura 13 mostra os dados sendo coletados em um dos sensores e os mesmos sendo apresentados no gateway.

```
Put [204] (201) {"smartdata": [{"confidence": 0, "x": 298, "z": 0, "value": 27.510000228881836, "y": 298, "t": 994051355, "version": 17, "error": 0, "unit": 2224179556, "dev": 0}], "credentials": {"domain": "grupo2"}}
Put [204] (201) {"smartdata": [{"confidence": 0, "x": 298, "z": 0, "value": 53.29999923706055, "y": 298, "t": 1001048013, "version": 17, "error": 0, "unit": 2224179500, "dev": 0}], "credentials": {"domain": "grupo2"}}
Put [204] (202) {"smartdata": [{"confidence": 0, "x": 298, "z": 0, "value": 27.520000457763672, "y": 298, "t": 1004084113, "version": 17, "error": 0, "unit": 2224179556, "dev": 0}], "credentials": {"domain": "grupo2"}}
Put [204] (201) {"smartdata": [{"confidence": 0, "x": 298, "z": 0, "value": 53.20000076293945, "y": 298, "t": 1006067178, "version": 17, "error": 0, "unit": 2224179500, "dev": 0}], "credentials": {"domain": "grupo2"}}
Put [204] (202) {"smartdata": [{"confidence": 0, "x": 298, "z": 0, "value": 27.530000686645508, "y": 298, "t": 1009094275, "version": 17, "error": 0, "unit": 2224179556, "dev": 0}], "credentials": {"domain": "grupo2"}}
Put [204] (201) {"smartdata": [{"confidence": 0, "x": 298, "z": 0, "value": 53.20000076293945, "y": 298, "t": 1011070342, "version": 17, "error": 0, "unit": 2224179500, "dev": 0}], "credentials": {"domain": "grupo2"}}
Put [204] (202) {"smartdata": [{"confidence": 0, "x": 298, "z": 0, "value": 27.549999237060547, "y": 298, "t": 1014098441, "version": 17, "error": 0, "unit": 2224179556, "dev": 0}], "credentials": {"domain": "grupo2"}}
Put [204] (201) {"smartdata": [{"confidence": 0, "x": 298, "z": 0, "value": 53.20000076293945, "y": 298, "t": 1016074508, "version": 17, "error": 0, "unit": 2224179500, "dev": 0}], "credentials": {"domain": "grupo2"}}
Put [204] (201) {"smartdata": [{"confidence": 0, "x": 298, "z": 0, "value": 27.56999969482422, "y": 298, "t": 1019101606, "version": 17, "error": 0, "unit": 2224179556, "dev": 0}], "credentials": {"domain": "grupo2"}}
Put [204] (201) {"smartdata": [{"confidence": 0, "x": 298, "z": 0, "value": 53.20000076293945, "y": 298, "t": 1021076673, "version": 17, "error": 0, "unit": 2224179500, "dev": 0}], "credentials": {"domain": "grupo2"}}
Put [204] (201) {"smartdata": [{"confidence": 0, "x": 298, "z": 0, "value": 27.56999969482422, "y": 298, "t": 1024104772, "version": 17, "error": 0, "unit": 2224179556, "dev": 0}], "credentials": {"domain": "grupo2"}}
Put [204] (202) {"smartdata": [{"confidence": 0, "x": 298, "z": 0, "value": 53.099998474121094, "y": 298, "t": 1026079839, "version": 17, "error": 0, "unit": 2224179500, "dev": 0}], "credentials": {"domain": "grupo2"}}
Put [204] (202) {"smartdata": [{"confidence": 0, "x": 298, "z": 0, "value": 27.559999465942383, "y": 298, "t": 1029106937, "version": 17, "error": 0, "unit": 2224179556, "dev": 0}], "credentials": {"domain": "grupo2"}}
```

```
Serie: {ve=17,u=2224179556,dst=(298,298,0)+0,t=[0,18446744073709551615]}
Record: {ve=17,u=2224179556,va=27.549,e=0,src=(298,298,0),t=1009094275,d=0}

Serie: {ve=17,u=2224179500,dst=(298,298,0)+0,t=[0,18446744073709551615]}
Record: {ve=17,u=2224179500,va=53.200,e=0,src=(298,298,0),t=1011070342,d=0}

Serie: {ve=17,u=2224179556,dst=(298,298,0)+0,t=[0,18446744073709551615]}
Record: {ve=17,u=2224179556,va=27.569,e=0,src=(298,298,0),t=1014098441,d=0}

Serie: {ve=17,u=2224179500,dst=(298,298,0)+0,t=[0,18446744073709551615]}
Record: {ve=17,u=2224179500,va=53.200,e=0,src=(298,298,0),t=1016074508,d=0}

Serie: {ve=17,u=2224179556,dst=(298,298,0)+0,t=[0,18446744073709551615]}
Record: {ve=17,u=2224179556,va=27.569,e=0,src=(298,298,0),t=1019101606,d=0}
```

Figura 13: na parte inferior são exibidos os dados de umidade e temperatura coletados pelo sensor dentro do ambiente enquanto que, na parte superior, são mostrados os dados recebidos pelo gateway.

c. Seleção de dados

Para a implementação do algoritmo de seleção de dados relevantes será utilizada as classes da biblioteca WEKA. Abaixo são apresentadas as classes e suas principais características:

- weka.core.converters.ConverterUtils.DataSource: auxilia o processo de carregar dados de arquivos e URLs, deixando mais transparente quais métodos devem ser utilizados para realizar conversão de dados quando os mesmos estão sendo carregados em memória {8}.
- weka.core.Instances: serve como auxílio à manipulação de conjuntos ordenados de instâncias de acordo com o peso delas {8}.
- weka.filters.supervised.attribute.AttributeSelection: possui um filtro para selecionar os atributos relevantes. É bastante flexível, permitindo até a combinação de técnicas de filtragem {8}.
- weka.attributeSelection.CfsSubsetEval: utilizada para avaliar um subconjunto de atributos e selecionar apenas os que sejam relevantes, levando em consideração o grau de redundância entre eles {8}.
- weka.attributeSelection.GreedyStepwise: permite a execução de buscas avançadas através dos subconjuntos de atributos, podendo começar com nenhum ou todos os atributos em um ponto arbitrário no espaço de busca {8}.
- weka.filters.Filter: uma classe abstrata que serve de base para instanciação de diversas técnicas de filtragem e, portanto, assume que os métodos de filtragem serão sobrescritos na implementação {8}.

A implementação do método que realiza a seleção de dados recebe como parâmetro um conjunto instâncias (Instances) presentes no objeto dataset. O arquivo ARFF (Attribute-Relation File Format) é então transformado em Instances pelo escopo que chamou este método. Abaixo está o código implementado.

```
// carrega o conjunto de dados
DataSource source = new DataSource("filename.arff");
Instances dataset = source.getDataSet();

// cria um objeto para filtrar os atributos
AttributeSelection filter = new AttributeSelection();

// cria uma objeto para avaliar os atributos e o algoritmo de busca
CfsSubsetEval cfs = new CfsSubsetEval();
GreedyStepwise search = new GreedyStepwise();
```

```
// definir método de procura
search.setSearchBackwards(true);

// atribui as configurações (algoritmos) a serem utilizadas no filtro
filter.setEvaluator(cfs);
filter.setSearch(search);

// especifica o conjunto de dados
filter.setInputFormat(dataset);

// aplicação do filtro no conjunto de dados gerando um novo conjunto de dados
Instances newData = Filter.useFilter(dataset, filter);
```

Em razão das poucas variáveis monitoradas no sistema, os membros da equipe entenderam que não teria um ganho considerável de desempenho realizando o algoritmo de feature selection em tempo de execução. Então o algoritmo foi realizado previamente com um conjunto de dados limitados, e as variáveis de maior relevância foram utilizadas para os algoritmos posteriores.

d. Aprendizado

Para a implementação do algoritmo de aprendizado, são listadas abaixo as principais classes do WEKA utilizadas para a criação e treinamento da rede neural:

- weka.core.converters.ConverterUtils.DataSource e weka.core.Instances: já foram mencionadas anteriormente.
- weka.core.Instance: interface que guarda uma instância, tendo todos os valores guardados internamente 8.
- weka.classifiers.Evaluation: utilizada, essencialmente, para avaliar modelos de aprendizado de máquina 8.
- weka.classifiers.UpdateableClassifier: interface para modelos de classificação incremental que podem aprender usando uma instância por vez 8.
- weka.classifiers.functions.SGD: implementa o algoritmo de gradiente descendente para a aprendizagem de vários modelos lineares. Substitui globalmente todos os valores ausentes e transforma os atributos nominais em binários. Também normaliza todos os atributos, de modo que os coeficientes na saída são baseados nos dados normalizados 8.
- weka.core.SelectedTag: representa um valor selecionado de um conjunto finito de valores 8.
- weka.classifiers.functions.MultilayerPerceptron: implementa um algoritmo de classificação através de retropropagação para classificação de instâncias, permitindo ter seus parâmetros monitorados e alterados durante o processo de aprendizagem 8. MultilayerPerceptron é a versão multicamada do algoritmo Perceptron 10. O Perceptron é uma rede neural que utiliza classificador binário que mapeia sua entrada x para um valor de saída f(x). O algoritmo atribui pesos às sinapses que ligam os neurônios da rede 11.

A implementação do algoritmo de aprendizado SGD está em SGDModel.java. Essa classe possui três métodos:

- update(Instances data): é responsável por re-treinar a rede rede neural, para o conjunto de instâncias (Instances) recebidos como parâmetro.

- update(Instance data): também é responsável pelo re-treinamento da rede, mas a partir de apenas uma instância (Instance).
- relearning(Instances data): recebe como parâmetro um conjunto instâncias (Instances) a serem utilizados para o treinamento. Em caso de erro na predição este é o método a ser utilizado.

O algoritmo MultilayerPerceptron também foi implementado, MultilayerModel.java, mas não será utilizado pois não possui a opção de re-treinamento.

e. Seleção e Aprendizado sobre os Dados Reais

Inicialmente, foi utilizado arquivos no formato ARFF para representação dos dados de entrada aos algoritmos implementados pela biblioteca Weka. Assim, para a realização do feature selection e aprendizagem foi criado um script capaz de pegar os Smart Data do banco de dados Cassandra, utilizando a REST API, para então convertê-los para um arquivo com a extensão ARFF.

O script, implementado em python, realiza uma query no banco de dados através da classe get.php, disponível na plataforma REST API. Em seguida, cada Smart Data é inserido em uma lista, de acordo com o sensor que o coletou (sensores que monitoram dentro e fora do ambiente, respectivamente) e a unidade de representação (temperatura e umidade).

Para construir o contexto do ambiente em determinado momento no arquivo ARFF, foi utilizado o próprio timestamp do Smart Data (contém informações temporais do momento de coleta do dado) e assim o arquivo ARFF foi preenchido. Além disso, para simular a temperatura que o usuário considera ideal, foi utilizada a ideia de que próximo ao meio-dia, em geral, os usuários gostam de uma temperatura mais baixa, enquanto que nos demais períodos do dia eles gostam de uma temperatura mais alta. As Figuras 14, 15 e 16 mostram, respectivamente, o arquivo ARFF resultante de uma query ao banco de dados, o arquivo ARFF resultante da execução do algoritmo de feature selection e o modelo de rede neural resultante após o uso do algoritmo MultilayerPercepton.

```
@relation weather

@attribute temperature_inside numeric
@attribute temperature_outside numeric
@attribute humidity_inside numeric
@attribute humidity_outside numeric
@attribute second numeric
@attribute minute numeric
@attribute hour numeric
@attribute day numeric
@attribute w_day numeric
@attribute month numeric
@attribute year numeric
@attribute temperature_ideal numeric

@data
28.15999984741211,23.780000686645508,69.80000305175781,83.4000015258789,47,37,6,7,3,8,48821445,25
28.15999984741211,23.780000686645508,69.69999694824219,83.5999984741211,47,37,6,7,3,8,48821445,25
28.18000030517578,23.760000228881836,69.69999694824219,83.5999984741211,58,49,8,1,0,12,48821445,22
28.18000030517578,23.760000228881836,69.80000305175781,83.5999984741211,58,49,8,1,0,12,48821445,22
28.18000030517578,23.760000228881836,69.80000305175781,83.5999984741211,58,49,8,1,0,12,48821445,22
28.15999984741211,23.729999542236328,69.69999694824219,83.69999694824219,35,1,11,27,4,3,48821446,22
28.15999984741211,23.729999542236328,69.69999694824219,83.69999694824219,35,1,11,27,4,3,48821446,22
28.15999984741211,23.729999542236328,69.69999694824219,83.80000305175781,35,1,11,27,4,3,48821446,22
28.149999618530273,23.739999771118164,69.69999694824219,83.80000305175781,44,17,16,21,1,7,48821446,24
28.149999618530273,23.739999771118164,69.80000305175781,83.9000015258789,44,17,16,21,1,7,48821446,24
28.149999618530273,23.739999771118164,69.80000305175781,83.9000015258789,44,17,16,21,1,7,48821446,24
28.18000030517578,23.729999542236328,69.80000305175781,83.9000015258789,50,13,18,17,3,9,48821446,26
28.18000030517578,23.729999542236328,69.80000305175781,83.9000015258789,50,13,18,17,3,9,48821446,26
28.18000030517578,23.729999542236328,69.80000305175781,83.9000015258789,50,13,18,17,3,9,48821446,26
28.15999984741211,23.729999542236328,69.80000305175781,83.9000015258789,54,29,18,14,5,11,48821446,26
28.15999984741211,23.729999542236328,69.69999694824219,83.9000015258789,54,29,18,14,5,11,48821446,26
28.15999984741211,23.729999542236328,69.69999694824219,83.9000015258789,54,29,18,14,5,11,48821446,26
28.15999984741211,23.739999771118164,69.69999694824219,84,9,42,20,10,2,3,48821447,26
28.15999984741211,23.739999771118164,69.69999694824219,84,9,42,20,10,2,3,48821447,26
28.15999984741211,23.739999771118164,69.69999694824219,83.9000015258789,9,42,20,10,2,3,48821447,26
28.170000076293945,23.729999542236328,69.69999694824219,83.9000015258789,35,41,20,7,4,5,48821447,26
28.170000076293945,23.729999542236328,69.69999694824219,83.9000015258789,35,41,20,7,4,5,48821447,26
28.170000076293945,23.729999542236328,69.69999694824219,83.9000015258789,35,41,20,7,4,5,48821447,26
```

Figura 14: arquivo ARFF gerado através do script python após uma query ao banco de dados.

```
@relation 'weather-weka.filters.supervised.attribute.AttributeSelection-Eweka.attributeSelection.CfsSubsetEval -P 1 -E 1-
Sweka.attributeSelection.GreedyStepwise -T -1.7976931348623157E308 -N -1 -num-slots 1'

@attribute temperature_inside numeric
@attribute temperature_outside numeric
@attribute humidity_inside numeric
@attribute hour numeric
@attribute w_day numeric
@attribute month numeric
@attribute temperature_ideal numeric

@data
28.16,23.780001,69.800003,6,3,8,25
28.16,23.780001,69.699997,6,3,8,25
28.18,23.76,69.699997,8,0,12,22
28.18,23.76,69.800003,8,0,12,22
28.18,23.76,69.800003,8,0,12,22
28.16,23.73,69.699997,11,4,3,22
28.16,23.73,69.699997,11,4,3,22
28.16,23.73,69.699997,11,4,3,22
28.15,23.74,69.699997,16,1,7,24
28.15,23.74,69.800003,16,1,7,24
28.15,23.74,69.800003,16,1,7,24
28.18,23.73,69.800003,18,3,9,26
28.18,23.73,69.800003,18,3,9,26
28.18,23.73,69.800003,18,3,9,26
28.16,23.73,69.800003,18,5,11,26
28.16,23.73,69.699997,18,5,11,26
28.16,23.73,69.699997,18,5,11,26
28.16,23.74,69.699997,20,2,3,26
28.16,23.74,69.699997,20,2,3,26
28.16,23.74,69.699997,20,2,3,26
28.17,23.73,69.699997,20,4,5,26
28.17,23.73,69.699997,20,4,5,26
28.17,23.73,69.699997,20,4,5,26
28.15,23.77,69.699997,22,6,7,26
28.15,23.77,69.699997,22,6,7,26
28.15,23.77,69.699997,22,6,7,26
28.16,23.780001,69.699997,0,2,9,25
```

Figura 15: arquivo ARFF resultante do algoritmo de feature selection.

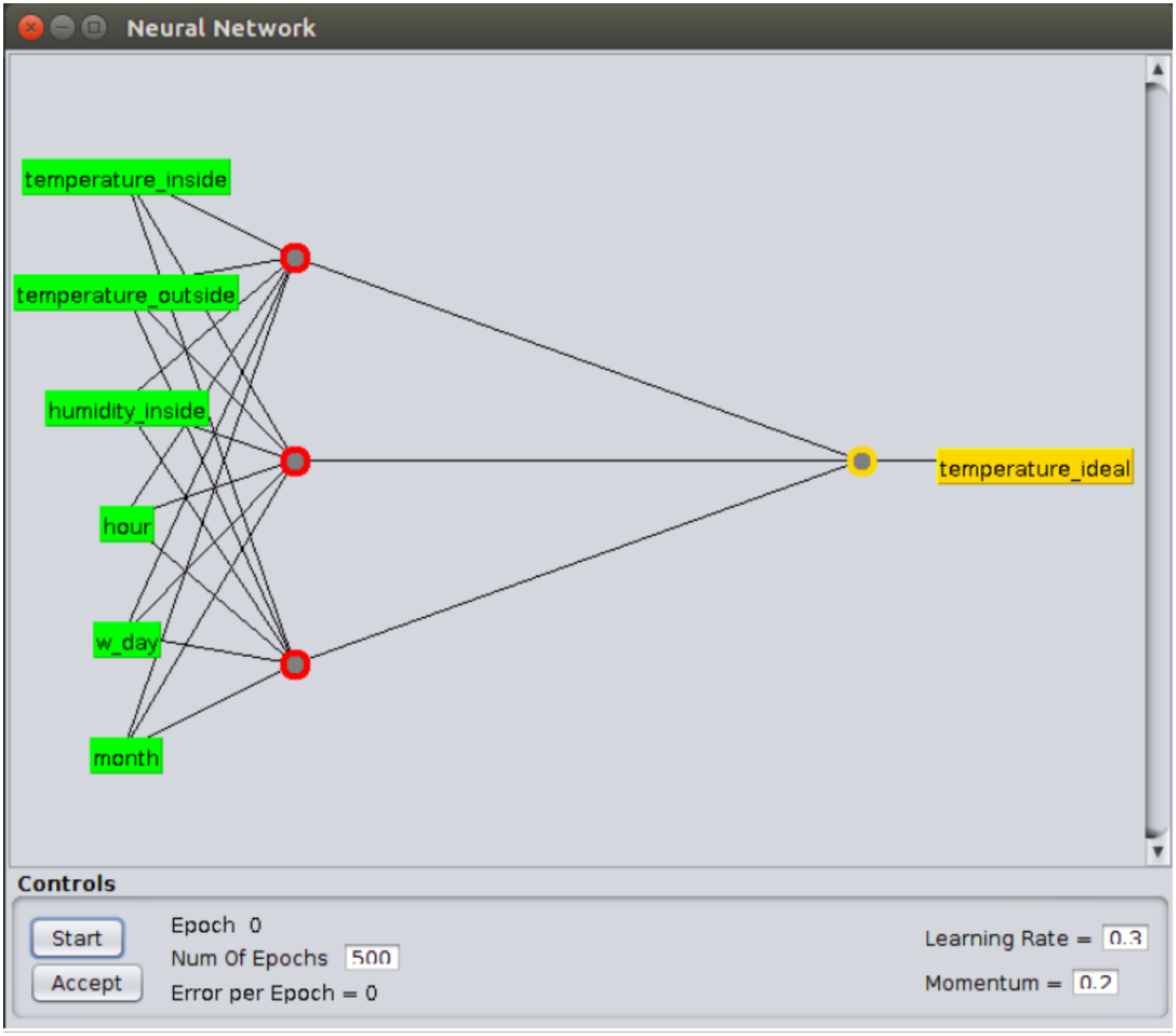


Figura 16: rede neural criada pelo Weka após o treinamento através do algoritmo MultilayerPerceptron.

Assim que o processo de aprendizado for implantado na plataforma IoT do Lisha, o arquivo ARFF, assim como a sua construção, serão substituídos por querys diretas no banco de dados e a comunicação com o daemon controlador será realizada por intermédio de streams ou pipes. Desta forma, é possível se beneficiar do desempenho superior de comunicação entre processos ao invés da penalização causada pelo uso de escritas constantes no disco.

f. Comunicação do Gateway com a Plataforma IoT

O processo de comunicação do gateway com a plataforma IoT do Lisha segue o exemplo mostrado na análise de viabilidade (implementada durante o D2).

g. Detecção do usuário no ambiente

Como dito na análise de viabilidade, a detecção do usuário seria realizada com base nos scripts dados em 3. No entanto, quando os scripts foram executados no ambiente foco de estudo (Laboratório de Computação Embarcada - ECL) verificou-se que a rede da universidade era um pouco mais complicada de

se trabalhar.

Primeiramente, a máscara de sub-rede do IP dos dispositivos variam de acordo com a conexão (Ethernet ou Wireless), de forma que o desktop onde foi conectado o gateway não poderia, através da rede, ter acesso aos endereços MAC de dispositivos em outras sub-redes e, portanto, não poderia verificar quando o usuário se conectasse nessa rede.

Como alternativa, tentou-se utilizar o notebook onde foi conectado um dos sensores, assim, como ele utiliza conexão wireless ele estaria na mesma sub-rede que o dispositivo móvel do usuário. No entanto, verificou-se que mesmo conectando-se, possivelmente no mesmo ponto de acesso, a máscara de sub-rede do IP que o notebook recebeu diferenciava-se tanto do desktop quanto do dispositivo do usuário. Além disso, foi verificado um terceiro dispositivo móvel, conectado possivelmente através do mesmo ponto de acesso que os dispositivos anteriores, e notou-se que a máscara de sub-rede para o IP desse dispositivo também era completamente diferente dos demais. A Figura 17 mostra o resultado do comando ip -o -f inet show para o desktop, notebook e dispositivo do usuário.

```
~/Downloads/arping/src $ ip -o -f inet addr show
1: lo    inet 127.0.0.1/8 scope host lo\       valid_lft forever preferred_lft forever
2: eno1    inet 150.162.57.219/26 brd 150.162.57.255 scope global dynamic eno1\       valid_lft 587sec preferred_lft 587sec
3: docker0    inet 172.17.0.1/16 scope global docker0\       valid_lft forever preferred_lft forever
4: br-cdcb0e70273d    inet 172.18.0.1/16 scope global br-cdcb0e70273d\       valid_lft forever preferred_lft forever
```

```
~ $ ip -o -f inet addr show
1: lo    inet 127.0.0.1/8 scope host lo\       valid_lft forever preferred_lft forever
3: wlp8s0    inet 150.162.207.8/23 brd 150.162.207.255 scope global dynamic wlp8s0\       valid_lft 1084sec preferred_lft 1084sec
```

```
$ ip -o -f inet addr show
1: lo    inet 127.0.0.1/8 scope host lo
30: wlan0    inet 150.162.230.252/23 brd 150.162.231.255 scope global wlan0
30: wlan0    inet 150.162.230.252/16 brd 150.162.255.255 scope global wlan0
```

Figura 17: resultado do comando ip -o -f inet show para o desktop (terminal de cima), notebook (terminal do meio) e dispositivo do usuário (terminal de baixo) onde todos mostram máscaras de sub-rede distintas.

Por fim, verificou-se que a solução previamente estabelecida para localização do usuário na rede não será possível de ser aplicada, já que o um dispositivo em uma sub-rede não pode ter acesso ao endereço MAC de dispositivos fora da sua sub-rede.

Como alternativa, foi verificada a possibilidade de utilizar a conexão bluetooth para detectar o usuário no ambiente. Foi criado um script bash (mostrado abaixo) que recebe um endereço MAC, ativa o dispositivo bluetooth do notebook, procura pelo usuário, desliga o dispositivo bluetooth e finaliza retornando se o usuário foi ou não encontrado.

```
# Endereco MAC passado por parametro
MAC=$1

# Liga o dispositivo bluetooth
rfkill unblock bluetooth

# Aguarda a inicialização do dispositivo
sleep 2

# Lista os dispositivos encontrados e procura pelo endereço MAC fornecido
USER="$(bt-device --list | grep $MAC)"
```

```
# Verifica se o usuario existe ou não.
# Caso exista limpa o historico e atribui a variavel END
# Caso não exista atribui 0 a variavel END
if [ "$USER" ]; then
        bt-device --remove $MAC > /dev/null
        END=1
else
        END=0
fi

# Desliga o dispositivo bluetooth
rfkill block bluetooth

# encerra o script com o resultado da busca pelo MAC do usuario
exit $END
```

Com a finalidade de listar todos os usuário possíveis de serem encontrados no ambiente e verificar quando os mesmos estão conectados e quando desconectam-se, foi implementado um script em python que faz uso do script bash mencionado acima. A Figura 18 mostra o resultado da execução desse algoritmo.

```
from subprocess import call
import time

def searchUser():

        # Lista de todos os usuario que podem ser localizados no ambiente
        userList = [("GT-S6812B", "C0:65:99:D3:5E:86"), ("Bonotto", "00:34:DA:A9:3F:23")]

        while True:

                MAC = ""
                connected = False

                for i in userList:

                        # Executa o script para localizar o MAC atraves do bluetooth
                        # Caso retorne 1 o usuário foi localizado. Então atualiza as
                        # variaveis connected para true e o MAC do usuario
                        # conectado e finaliza a busca.
                        # Imprime uma mensagem caso contrario
                        if call("./searchMAC.sh " + i[1], shell=True):
                                print("User " + i[0] + " found!!")
                                connected = True
                                MAC = i[1]
                                break

                        print("User " + i[0] + " not found!!")

                # Se o MAC não foi atualizado e por que não existe nenhum usuario
                if not MAC:
                        time.sleep(5)

                # Caso exista um usuario no ambiente então verifica de tempo em
```

```
                # tempo se ele ainda esta conectado.
                else:
                        time.sleep(1)
                        while call("./searchMAC.sh " + MAC, shell=True):
                                print("User still connected!!")
                                time.sleep(1)
                informServer(”“)

searchUser()
```

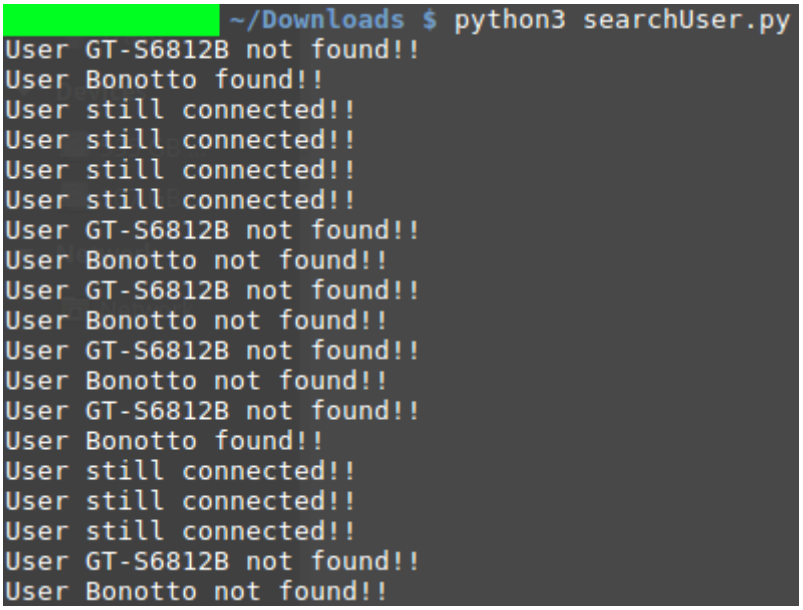


Figura 18: resultado da execução do script python para verificação do usuário. Primeiramente, o usuário Bonotto estava no ambiente e depois foi embora, voltando novamente após três verificações e deixando o ambiente novamente depois.

Para finalizar a detecção do usuário é necessário que o servidor seja informado sobre sua presença e, dessa maneira, possa realizar suas tarefas de acordo com o especificado. Nesse sentido, foi criada uma função no script em python para avisar o servidor qual o usuário que foi localizado no ambiente, processo este que é realizado através do envio de um smartdata.

```
def informServer(MAC)
        put_url ='https://iot.lisha.ufsc.br/api/put.php'
        CLIENT_CERTIFICATE = ['client-xx-xxx.pem',
 'client-xx-xxx.key']
        query = {
            'smartdata': [ {
                'version': 0,
                'confidence': 0,
                't': 0,
                'unit': 0,
                'error': 0,
                'dev': 1,
                'y': 300,
                'x': 300,
                'z': 0,
                'value': 0,
                'dev': 0,
        'mac': MAC
```

```
            } ],
            'credentials': {
                'domain': 'grupo2'
            },
            'workflow': 106,
        }
        session = requests.Session()
        session.cert = CLIENT_CERTIFICATE
        session.headers = {'Content-type' : 'application/json'}
        response = session.post(put_url, json.dumps(query))
        print("Put [", str(response.status_code), "]", sep='')
        if response.status_code == 204:
            print('Put: OK!\n')
            print(json.dumps(query, indent=4, sort_keys=False))
        else:
            print("Put: Failed!")
searchUser()
```

Embora esteja implementada, a função ainda não pode ser utilizada devido à incompatibilidade com o formato do JSON enviado ao servidor, sendo necessário a inclusão do campo MAC que, atualmente, ainda não existe.

h. Estrutura do Controlador (*daemon*)

O controlador (daemon) foi projetado para utilizar pipes (Named Pipes ou Fifos) como meio para a comunicação entre processos. Desta forma, o daemon ganha uma postura serverless em relação a comunicação, não consumindo CPU quando estiver bloqueado ou aguardando novas requisições.

O controlador é composto por quatro threads fixas para tarefas periódicas e uma criada em um contexto específico. A thread fachada, denominada Daemon, efetiva a comunicação de dados e de comandos através da fila implementada sobre o pipe nomeado do sistema operacional com o objetivo de ser o mais eficiente possível.
Internamente, os dados são entregues a thread learning responsável por lidar com os dados e a criação de contextos através de uma fila sincronizada de mensagens. A thread learning é responsável por instanciar duas outras threads. A thread watchmaker é inicializada apenas uma vez e fica responsável por atualizar a temperatura ideal com base na predição do modelo, caso o usuário não tenha solicitado uma alteração no intervalo de 30 minutos. A outra thread, denominada worker, é criada apenas para atualizar o modelo de predição para que liberar a learning para atender novas solicitações.

Os comandos seguem a mesma ideia anterior, sendo entregues por intermédio de uma fila sincronizada à thread controlling, porém seu funcionamento é simples, sendo responsável apenas de atualizar o comando ideal do usuário no contexto atual.

Sincronização

A sincronização das threads é realizada através da primitiva synchronized disponibilizada pela linguagem Java protegendo métodos e objetos que possuem acesso concorrente. Basicamente, todos os objetos em Java possuem um mutex implícito, o que nos permite criar seções críticas distintas facilmente associando-a a um objeto alvo.

Requisições

Como pode ser visto no código, Daemon.java, o loop principal da thread Daemon contém um switch case que, quando recebe uma nova requisição, decide qual operação deve ser realizada. Vale notar que as operações que envolvem a manipulação da cache (de dados ou de controle) é executada em paralelo permitindo que o controlador possa voltar a lidar com novas requisições.

A mensagem da requisição em si é simples, contendo apenas um enumerador que identifica o tipo de requisição e um SmartData associado, como pode ser visto no código abaixo. O objeto SmartData segue a mesma ideia, sendo apenas um wrapper para as informações do JSON passado para o controlador. O código do objeto Message está disponível em Message.java, enquanto o do SmartData está em SmartData.java.

Mensagem em formato JSON:

```
{
 "type" : 3, //! DATA
"smartdata" : { "version" : "1.7", "unit" : 2224179500, "value" : 23, "error" : 0,
     "confidence" : 3, "x" : 298, "y" : 302, "z" : 0, "t" : 1641568216256, "dev" : 0 }
}
```

Named Pipe

Um objeto da classe NamedPipe, como pode ser visto no código, NamePipeReader.java, utiliza o programa mkfifo (sistemas unix) para criação do arquivo pipe. A abertura para leitura do pipe acontece em toda chamada da função receive porque, ao deixá-lo aberto, o próximo processo não bloquearia ao tentar ler do buffer.

O exemplo a seguir exemplifica o envio de uma mensagem para o armazenamento de um SmartData de dados. Para simplificação da comunicação foi utilizado o comando cat para enviar a mensagem previamente construída. A demonstração da persistência da cache e de sua atualização também foi simplificada para realizar as operações necessárias quando apenas uma nova instância do contexto for construída.

```
Daemon up ...
Configuring ...
Services up ...
Waiting requisitions ...
```

Figura 17: Controlador esperando requisições.

```
ContextDaemon on ⊱ dev3 [!]
➜ cat smartdata > myfifo
```

Figura 18: Envio do dado para o controlador.

```
Daemon up ...
Configuring ...
Services up ...
Waiting requisitions ...
Requisição de dados:
Mensagem: Message [type=DATA, smartdata=SmartData [unit=2224179500, value=23.0, x=298, y=302, z=0, t=1641568216256, error=0, con
Waiting requisitions ...
Thread inicializada para atualização da cache de dados!
Atualização das médias 0
avg_internal_temps: 0.0
avg_external_temps: 0.0
avg_internal_hums: 23.0
avg_external_hums: 0.0
n_in_temp: 0
n_out_temp: 0
n_in_hum: 1
n_out_hum: 0
```

Figura 19: Primeiro dado recebido.

```
Requisição de dados:
Mensagem: Message [type=DATA, smartdata=SmartData [unit=2224179500, value=23.0, x=302, y=302, z=0, t=1641568216256, error=0, con
Waiting requisitions ...
Thread inicializada para atualização da cache de dados!
Atualização das médias 2
avg_internal_temps: 0.0
avg_external_temps: 0.0
avg_internal_hums: 23.0
avg_external_hums: 23.0
n_in_temp: 0
n_out_temp: 0
n_in_hum: 2
n_out_hum: 1
```

Figura 20: Terceiro dado recebido.

i. Cache de dados e de comandos de usuário

A implementação da cache de dados e comandos, assim como as políticas de atualização foram realizar dentro da classe CacheController onde serão realizadas todas as manipulações existentes, tanto para dados como para comandos do usuário.

A política de cache para os dados utilizada foi manter um histórico das últimas 24 horas em que o usuário esteve no ambiente com instâncias que representam um intervalo de 30 segundos. A escolha desse intervalo se justifica pelo grande volume de dados coletados e a pouca variação existente entre curtos períodos de tempo. Desta forma, mantemos um histórico médio do usuário sem perder o contexto dos dados.

A política para comandos mantém fixo um comando feito pelo usuário por 30 minutos, dando tempo ao modelo aprender o novo contexto. Após esse período, é realizada a predição a partir do modelo com o mesmo peso de um comando do usuário. Caso o usuário não altere novamente, essa predição vai ser atualizada no mesmo intervalo de 30 minutos.

Optou-se por utilizar um arquivo .arff para armazenar de forma persistente os dados pela facilidade de carregamento na inicialização dos dados, além da garantia de contexto fornecida pela estrutura Instance do Weka. Uma Instance armazena um contexto completo de aprendizado do usuário, contendo as entradas associadas a um período do tempo em virtude a uma temperatura ideal do usuário. Desta forma, o aprendizado contínuo e a sua contra-parte em disco, são facilmente complacente com o uso proposto.

O código disponível no github, CacheController.java, mostra a implementação para o controlador das caches. Como continuação do exemplo sobre a comunicação do controlador pelo pipe, as imagens a seguir exemplificam a atualização em disco da cache de dados.

```
@relation weather

@attribute temperature_inside numeric
@attribute temperature_outside numeric
@attribute humidity_inside numeric
@attribute humidity_outside numeric
@attribute minute numeric
@attribute hour numeric
@attribute w_day numeric
@attribute temperature_ideal numeric

@data
18,16,75,70,11,12,1,19
19,18,75,76,30,13,1,19
20,20,75,80,20,14,1,19
21,22,75,78,55,15,1,19
22,23,75,62,32,16,1,20
23,23,75,62,54,17,1,20
24,24,75,60,21,18,1,22
25,27,75,55,01,19,1,22
26,28,75,50,23,20,1,22
```

Figura 21: Dados da cache de dados reais previamente armazenados.

```
1   @relation weather
2
3   @attribute temperature_inside numeric
4   @attribute temperature_outside numeric
5   @attribute humidity_inside numeric
6   @attribute humidity_outside numeric
7   @attribute minute numeric
8   @attribute hour numeric
9   @attribute w_day numeric
10  @attribute temperature_ideal numeric
11
12  @data
13  18,16,75,70,11,12,1,19
14  19,18,75,76,30,13,1,19
15  20,20,75,80,20,14,1,19
16  21,22,75,78,55,15,1,19
17  22,23,75,62,32,16,1,20
18  23,23,75,62,54,17,1,20
19  24,24,75,60,21,18,1,22
20  25,27,75,55,1,19,1,22
21  26,28,75,50,23,20,1,22
22  18,16,75,70,11,12,1,19
23  19,18,75,76,30,13,1,19
24  20,20,75,80,20,14,1,19
25  21,22,75,78,55,15,1,19
26  22,23,75,62,32,16,1,20
27  23,23,75,62,54,17,1,20
28  24,24,75,60,21,18,1,22
29  25,27,75,55,1,19,1,22
30  26,28,75,50,23,20,1,22
31  0,0,23,23,10,15,6,22
```

Figura 22: Cache após a nova instância ser criada (linha 31), vale ressaltar que existem dois arquivos de cache, um com valores padrão e outro com os dados reais. Na imagem acima houve a concatenação desses dois conjuntos por causa da pequena quantidade de amostras.

j. Processo de Aprendizagem e Predição

A primeira rede neural é criada na inicialização do Daemon. Como entrada para o algoritmo SGD, é utilizado o conjunto de dados que está disponível na cache, tanto do próprio usuário como valores padrões caso os mesmos sejam insuficientes. Quando o tipo da mensagem enviada ao Daemon for PREDICT, será solicitado ao objeto LearningRunnable a predição da temperatura do ambiente ideal ao usuário. Se a predição dada pela rede neural não foi a desejada, o usuário enviará um comando com a temperatura ideal. Isso modificará a variável _user_mode, mantendo, pelos próximos 30 minutos, a criação das instâncias com a temperatura que o usuário solicitou. Durante esse período de tempo não serão realizadas predições utilizando o modelo, mas utilizando a temperatura ideal fornecida pelo usuário. O objetivo dessa lógica é aprender o contexto do usuário, depois de detectar um erro na predição.

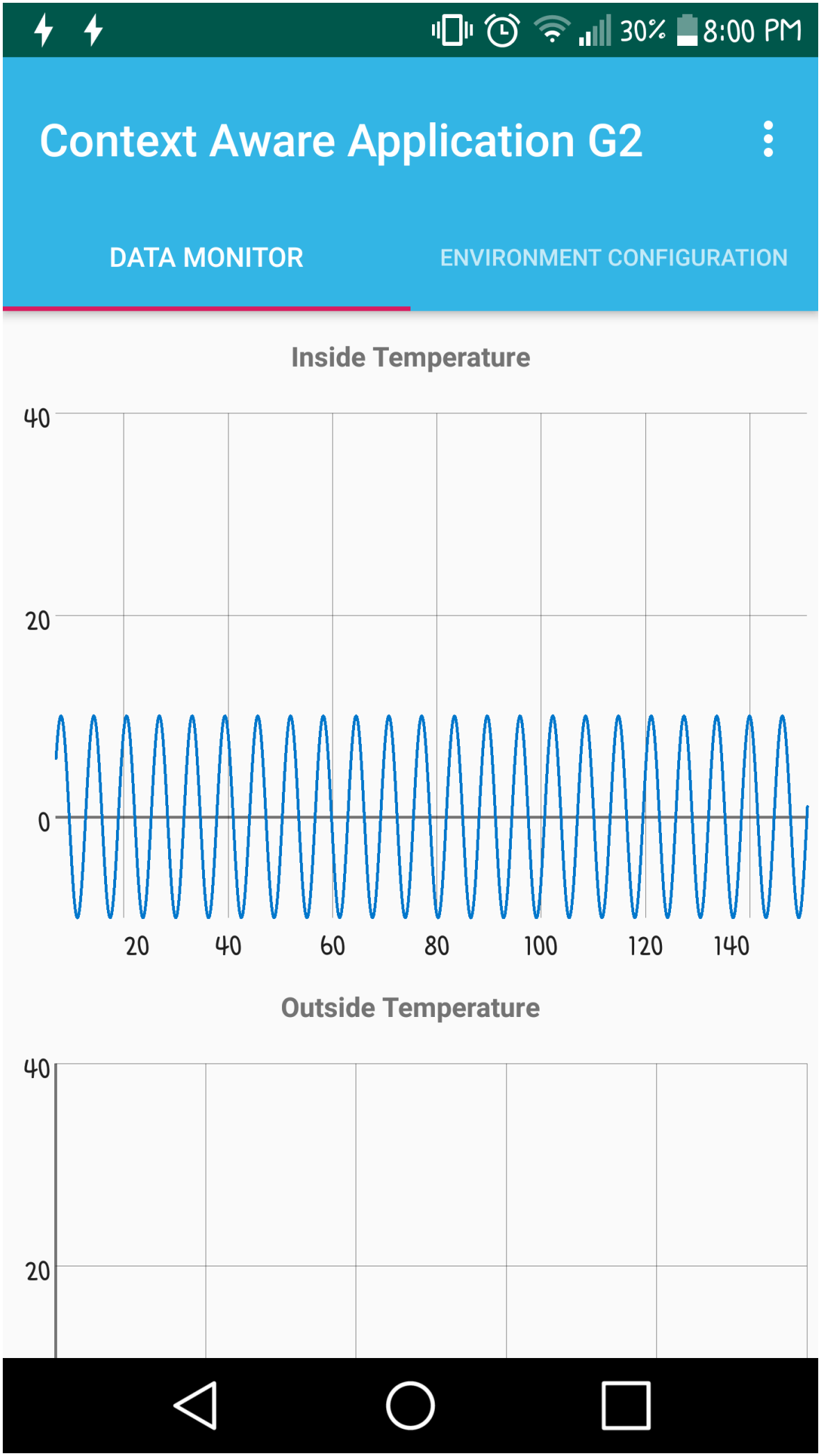
Passados 30 minutos, o mesmo modelo é treinado com as instâncias criadas, e voltando a predizer utilizando a rede neural. Enquanto as predições do modelo estiverem certas, o modelo vai sendo aprendendo através de dados recebido dos sensores com o contexto predito anteriormente. Esse fluxo pode ser visualizado nos arquivos main, learning e runnable no github, github Daemon.

h. Aplicação móvel para controle e monitoramento

A visualização das configurações ideais do ambiente e a monitoração da coleta de dados são importantes para a demonstração do funcionamento correto do sistema. Para sua implementação foi utilizada a IDE para desenvolvimento de aplicações para dispositivos Android chamada Android Studio.

Devido a inexperiência, sua configuração foi realizada tardiamente, não permitindo a finalização da aplicação. No entanto, a aplicação está basicamente finalizada, com a interface gráfica pronta e, portanto, basta apenas implementar uma função em Java para acessar os dados do servidor IoT do Lisha e construir os gráficos de acordo com os dados recebidos. A Figura abaixo mostra a tela inicial da aplicação, onde foi construído um gráfico com dados não reais.

A aplicação possui duas abas, uma para visualização dos dados e outra para a configuração do ambiente, a qual mostrará apenas um texto informando o usuário sobre a configuração do ambiente, sendo esses dados sendo pegos de forma semelhante à um get.


30%
8:00 PM
Context Aware Application G2
DATA MONITOR
ENVIRONMENT CONFIGURATION
Inside Temperature
40
20
0
20
40
60
80
100
120
140
Outside Temperature
40
20

Legenda: tela inicial da aplicação móvel para visualização dos dados monitorados e configuração ideal do ambiente.

Validação

A validação do projeto será realizada através de pequenos testes para cada objetivo específico com o intuito de demonstrar o correto funcionamento dos módulos desenvolvidos. Por fim, será demonstrado o funcionamento da arquitetura integrada e funcionando corretamente.

Cronograma

Tarefa	01/10	08/10	22/10	05/11	19/11	26/11	21-28/11
Planejamento	D0						
Revisão do Planejamento		D1					
Implementação I			D2				
Implementação II				D3			
Implementação III					D4		
Validação						D5	
Apresentação							Apres.

Tarefas e Avaliação

D0: Planejamento

- Definição do tema;
- Planejamento inicial do escopo do projeto.

D1: Revisão do Planejamento

- Detalhamento do plano inicial do projeto;
- Definição do escopo;
- Definição das tecnologias a serem utilizadas;
- Definição das atividades;
- Definição dos principais marcos do projeto.

D2: Implementação I

- Instalação e configuração dos sensores para coleta de dados.
    - Critérios de avaliação:
        - Corretude do código e configuração dos sensores para coleta e envio dos dados;
        - Corretude do código e configuração do gateway para recebimento dos dados coletados;
        - Demonstração da captura dos dados através de logs gerados pelos sensores e gateway;
        - Corretude na definição dos parâmetros dos Smart Data.

- Implementação do algoritmo de seleção de dados relevantes.
    - Critérios de avaliação:
        - Corretude da definição e relevância da justificativa dos algoritmos e parâmetros utilizados;
        - Demonstração da seleção das entradas realizada sobre dados reais.
- Implementação da rede neural para aprendizagem com o contexto.
    - Critérios de avaliação:
        - Corretude da definição dos algoritmos e parâmetros utilizados;
        - Demonstração do treinamento da rede neural usando os dados reais e geração do modelo de predição.

D3: Implementação II

- Instalação e configuração do gateway para comunicação com o servidor do Lisha.
    - Critérios de avaliação:
        - Corretude do código, da configuração e protocolo (script) utilizados para comunicação entre o gateway e a máquina host;
        - Corretude do código e da configuração para envio dos dados coletados para plataforma IoT do Lisha;
        - Demonstração do envio e recuperação dos dados para a plataforma IoT Lisha.
- Implementação da detecção de usuário no ambiente.
    - Critérios de avaliação:
        - Corretude do código para descoberta e monitoramento do usuário no ambiente;
        - Demonstração da descoberta da entrada do usuário no ambiente;
        - Demonstração da descoberta da saída do usuário do ambiente.
- Estrutura base do controlador (daemon).
    - Critérios de avaliação:
        - Corretude dos scripts e códigos desenvolvidos para o controlador;
        - Demonstração da comunicação inter-processos utilizada;
        - Demonstração da inicialização e persistência do processo daemon do controlador.
- Implementação da cache de dados.
    - Critérios de avaliação:
        - Corretude da implementação da política de cache para armazenamento de dados;
        - Demonstração do armazenamento e recuperação de dados da cache.
- Implementação da aplicação móvel para controle e monitoramento.
    - Critérios de avaliação:
        - Corretude no código e protocolos de comunicação utilizados;
        - Demonstração do envio de comandos e o monitoramento do ambiente.

D4: Implementação III

- Estrutura definitiva e política de controle do controlador (daemon).
    - Critérios de avaliação:
        - Corretude do código e protocolos utilizados pelo controlador;
        - Demonstração do controle sendo executado e do fluxo dos dados/comandos na arquitetura.

- Implementação da cache de comandos do usuário.
    - Critérios de avaliação:
        - Corretude da implementação da política de cache para armazenamento de comandos do usuário;
        - Demonstração do armazenamento e recuperação de comandos da cache.
- Implementação do processo de aprendizado e atualização do modelo.
    - Critérios de avaliação:
        - Corretude da implementação do programa de aprendizado e sua integração na plataforma IoT;
        - Demonstração da atualização e retreinamento de um modelo.
- Implementação de contextualização multiusuário (opcional).
    - Critérios de avaliação (acréscimo da nota final à critério do professor):
        - Corretude da implementação da política multiusuário;
        - Demonstração do funcionamento de predição para um usuário “composto”.

D5: Validação

- Finalização de possíveis módulos ainda incompletos.
- Avaliações do projeto.

D6: Apresentação

- Elaboração da Apresentação do projeto.

Referências

1. Abayomi Otebolaku and Gyu Myoung Lee, “A Framework for Exploiting Internet of Things for Context-Aware Trust-Based Personalized Services”, Mobile Information Systems, vol. 2018, Article ID 6138418, 24 pages, 2018. https://doi.org/10.1155/2018/6138418.
2. O. B. Sezer and E. Dogdu and A. M. Ozbayoglu, “Context-Aware Computing, Learning, and Big Data in Internet of Things: A Survey”, IEEE Internet of Things Journal. vol. 5, no. 1, Feb. 2018.
3. Rodrigo Schmitt Meurer, “Adaptive Context-Aware Control and Monitoring System for Smart Environments”, 2017. p.24. Trabalho de Conclusão de Curso - Universidade Federal de Santa Catarina, 2017.
4. Rodrigo Schmitt Meurer and Antônio Augusto Fröhlich and Jomi Fred Hübner, “Ambient Intelligence for the Internet of Things through Context-Awareness”, ainda não publicado, 2018.
5. Jundong Li et. al. “Feature Selection: A Data Perspective”. Journal ACM Computing Surveys, 50.6 (2018): 94. Web. 05 Oct. 2018.
6. R. Sheikhpour et. al. “A Survey on semi-supervised feature selection methods”. Pattern Recognition, 64 (2017): 141-158. Web. 05 Oct. 2018.
7. Shuochao Yao et al. "QualityDeepSense: Quality-Aware Deep Learning Framework for Internet of Things Applications with Sensor-Temporal Attention". 2nd International Workshop on Embedded and Mobile Deep Learning (Munich, Germany), (2018): 6. Web. 05 Oct. 2018.
8. WEKA: Data Mining with Open Source Machine Learning Software in Java. 21 de Outubro de 2018. Disponível em <http://weka.sourceforge.net/doc.dev/>.
9. Department of Computer Science: University of Waikato. 21 de Outubro de 2018. Disponível em

<https://www.cs.waikato.ac.nz/ml/weka/arff.html>.
10. Wikipédia. 28 de Outubro de 2018. Disponível em <https://en.wikipedia.org/wiki/Multilayer_perceptron>.
11. Wikipédia. 28 de Outubro de 2018. Disponível em<https://en.wikipedia.org/wiki/Perceptron>.