

**ADRIANO DENNANNI
RICARDO NAGANO
THIAGO LIRA**

**NET.MAP - SISTEMA DE POSICIONAMENTO
INDOOR**

São Paulo
2016

**ADRIANO DENNANNI
RICARDO NAGANO
THIAGO LIRA**

NET.MAP - SISTEMA DE POSICIONAMENTO INDOOR

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Eletricista com ênfase em Computação.

São Paulo
2016

**ADRIANO DENNANNI
RICARDO NAGANO
THIAGO LIRA**

NET.MAP - SISTEMA DE POSICIONAMENTO INDOOR

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Eletricista com ênfase em Computação.

Área de Concentração:
Engenharia de Computação

Orientador:
Prof. Dr. Reginaldo Arakaki

Co-orientador:
Eng. Marcelo Pita

São Paulo
2016

“Anything one man can imagine, other men can make real”

-- Jules Verne

“Enquanto sentir vontade de competir, buscar desafios e correr atrás de torneios, vou jogar”

-- Gustavo Kuerten

“Not all those who wander are lost”

-- J. R. R. Tolkien

RESUMO

Mapas físicos tornam-se cada vez menos utilizados com o desenvolvimento progressivo de sistemas de posicionamento cada vez melhores. O sistema americano GPS é possivelmente o mais utilizado, sendo que ele possibilita qualquer um ter informações sobre sua localização, dando apoio, por exemplo, à praticantes de trilhas e acampamentos, principalmente em casos de emergência. Porém, em ambientes fechados, as ondas eletromagnéticas utilizadas pelos satélites sofrem atenuações e interferências devidos aos materiais de construção, e assim o sistema perde precisão e não funciona com toda a precisão esperada. Como uma alternativa para esta dificuldade, procurou-se desenvolver um sistema, que consegue obter a posição do usuário em um ambiente fechado com precisão, sendo usado para isso técnicas de machine learning, aliadas com dados obtidos de redes em fio já instaladas no local. O sistema consistirá de um servidor central, onde serão enviados os dados e os mesmos serão processados. Os dados serão coletados por meio de um aplicativo de Android, este possuirá duas versões. A versão usuário usará os dados do servidor para localizar o usuário, a versão administrador irá coletar dados novos para serem usados em futuras medições.

Palavras-Chave – Localização Indoor, Wi-Fi, Machine Learning.

ABSTRACT

Physical maps are becoming each day less used due to constant evolution of positioning systems, better each day as well. The American system GPS probably is the most used and the most famous. It allows everyone to have their location information, giving support to hikers and campers, specially in emergency situations. On the other hand, in indoor environments, electromagnetic waves used by the satellites suffer with interference and mitigations and the systems loses precision and does not work as expected. As an alternative for this difficulty, it was developed a system that can locate the user position in an indoor environment with precision, using machine learning algorithms and data of wireless signals collected from the networks already existing on the place. The system consists on a main server that will receive the data and process it. The data will be collected with a Android app that will have two versions. The user version will use the server data to locate the user. The admin version will collect new data to be user on future measures.

Palavras-Chave – Indoor Location, Wi-Fi, Machine Learning.

LISTA DE FIGURAS

1	Algoritmo KNN aplicado com diferentes valores de K	16
2	Árvore de Decisão para prever a sobrevivência de passageiros do Titanic	17
3	SVM aplicado para classificar dados linearmente separáveis	18
4	SVM aplicado para classificar dados de maneira não linear	19
5	Arquitetura do Sistema net.map	20
6	Diagrama das classes na API	23
7	Erros para uso de Diferentes Distâncias	27
8	Erro em função do número de vizinhos	27
9	Erro para diversas quantidades de neurônios na rede neural.	28
10	Teste de Validação do Algoritmo C4.5	29
11	Comparação do C4.5 com o uso do AdaBoost	30
12	Erro do algoritmo SMO para uma quantidade crescente de zonas de classificação.	31
13	Erro do algoritmo de Redes Neurais para uma quantidade crescente de zonas de classificação.	32
14	Erro do algoritmo KNN para uma quantidade crescente de zonas de classificação.	32
15	Erro do algoritmo de Árvore de Decisão com e sem o Adaboost para uma quantidade crescente de zonas de classificação.	33
16	Erro do Voto Simples e Voto Ponderado para uma quantidade crescente de zonas de classificação.	34
17	Esquema mostrando como as aplicações no servidor interagem.	36
18	Tela do EletricGO mostrando um <i>Pokémon</i> na sala C2-59 do prédio de Engenharia Elétrica da POLI-USP	38
19	Acerto de Classificação usando-se a Média Aritmética como método de normalização dos dados	40

20	Acerto de Classificação usando-se o Filtro de Kalmann como método de normalização dos dados	40
21	Tela para o usuário efetuar o login	44
22	Tela informando as instalações do usuário	44
23	Tela para criar nova instalação	45
24	Tela para criar nova zona no sistema	45
25	Tela para iniciar aquisição de sinais	45
26	Tela do aplicativo adquirindo os sinais	45
27	Tela informando que os dados estão sendo treinados	46
28	Tela do aplicativo informando a zona atual	46
29	Representação do 1º andar do bloco B	47
30	Representação do andar térreo do bloco B	48

LISTA DE TABELAS

1	Comparativo entre diferentes métodos de localização	13
2	Exemplo de divisão de instalação em zonas	22

LISTA DE SIGLAS

AP	Access Point
API	Application Programming Interface
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JSON	JavaScript Object Notation
KNN	k-Nearest Neighbors
ML	Machine Learning
NoSQL	Not Only SQL
RFID	Radio-Frequency Identification
SMO	Sequential Minimal Optimization
SQL	Structured Query Language
SVM	Support Vector Machine
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

SUMÁRIO

1	Introdução	12
1.1	Motivação	12
1.2	Objetivo	13
1.3	Justificativa	13
1.4	Organização	14
2	Aspectos Conceituais	15
2.1	<i>Bias vs. Variância: Um tradeoff</i>	15
2.2	Algoritmos de Machine Learning	16
2.2.1	KNN	16
2.2.2	Árvore de Decisão	16
2.2.3	Support Vector Machines	17
2.3	Aplicação na Prática	19
3	Especificação	20
3.1	Arquitetura	20
3.2	Requisitos	21
3.2.1	Requisitos Funcionais	21
3.2.1.1	Aplicativo	21
3.2.1.2	Servidor	21
3.2.2	Requisitos Não-Funcionais	21
3.2.2.1	Aplicativo	21
3.2.2.2	Servidor	21
3.3	Casos de Uso	22

3.3.1	Captura e Treinamento	22
3.3.2	Localização	23
3.4	Modelo de classes	23
4	Metodologia	24
4.1	Gerenciamento de tarefas	24
4.2	Gerenciamento de código	24
4.3	Pesquisa	24
5	Projeto e Implementação	25
5.1	Machine Learning	25
5.1.1	Tratamento dos dados	25
5.1.1.1	Dados obtidos do Repositório UCI	25
5.1.1.2	Dados do Servidor	25
5.1.2	Formato dos dados tratados	26
5.1.3	Modelos Usados e <i>Cross-Validation</i> de parâmetros	26
5.1.3.1	KNN : K-Nearest-Neighbors	26
5.1.3.2	Rede Neural	28
5.1.3.3	Arvore de decisão	29
5.1.3.4	Comparação dos Modelos Usados	31
5.1.4	Métodos de Votação	33
5.1.5	Testes com os Métodos de Votação	34
5.2	Aquisição de dados	34
5.2.1	Filtro de Kalman	35
5.3	Estrutura do servidor	35
6	Aplicações do net.map	37
6.1	APScanner	37

6.2 EletricaGO	37
7 Testes in loco e Avaliação de Resultados	39
7.1 Testes <i>in loco</i>	39
7.1.1 Método de Teste	39
7.2 Objetivos e Resultados	41
8 Considerações Finais	42
Referências	43
Anexo A – Telas do aplicativo APScanner	44
Anexo B – Plantas do prédio de Engenharia Elétrica da Poli-USP com medições	47

1 INTRODUÇÃO

1.1 Motivação

Com a modernização das tecnologias de telefonia móvel torna-se cada vez maior o número de pessoas com acesso à *Internet*, através de tecnologias como *Wi-fi*, 3G e 4G. Essas formas de acesso à rede fornecem informações a provedores sobre o usuário a todo momento, como o conteúdo acessado por seus navegadores ou aplicativos e informações sobre posição e deslocamento. Dados de localização por si possuem pouco valor, mas quando aliados a outros conteúdos, é possível fornecer conteúdo personalizado em tempo real, reativo ao ambiente, passando a oferecer valor real à empresas e entidades.

Sistemas de posicionamento por satélite como GPS conseguem localizar um dispositivo na Terra com uma precisão na casa dos centímetros em ambientes abertos. Porém, o mesmo não ocorre em lugares fechados, como residências e edifícios. Isso ocorre devido à atenuação dos sinais dos satélites causada pelas paredes e tetos das estruturas. Tendo em vista o crescimento das cidades e o consequente aumento no número de construções, as pessoas cada vez passam mais tempo em ambientes fechados. A necessidade de serviços de localização *indoor* tem se tornado cada vez mais evidente.

Respondendo a essa necessidade, surgiram alternativas para o posicionamento em ambientes fechados, tais como o emprego de *tags RFID* (*Radio Frequency Identification*) e do *Bluetooth*. Tendo em vista este cenário e as condições tecnológicas atuais, este projeto procura apresentar uma solução alternativa para localização de pessoas em ambientes fechados, como shoppings e eventos em galpões, sem ter que investir altos valores em infraestrutura. Para tal, será utilizada a tecnologia *Wi-Fi* combinada a técnicas de *Machine Learning*.

A tabela 1 compara as tecnologias de localização citadas acima.

Tabela 1: Comparativo entre diferentes métodos de localização

	GPS	Triangulação por Bluetooth	Wi-Fi e Machine Learning
Localização em ambientes fechados	X	✓	✓
Precisão	Boa (Quando o sinal é estável)	Boa ~ Média (Depende de como foi instalado)	Boa ~ Média (Depende da quantidade de Access Points e medidas no ambiente)
Custo	Uso gratuito	É necessário adquirir, instalar e arcar com a manutenção de Beacons Bluetooth	Custo somente no acesso aos servidores. Leva em consideração que o ambiente já possui Access Points.
Gasto de bateria para o usuário	Alto	Médio	Baixo

Esta abordagem se mostra interessante ao ponto de que sua implementação não necessita de configurações particulares nas redes ao redor, uma vez que se baseia em leituras feitas pelo aparelho móvel e no processamento dos dados feitos em um servidor em nuvem. Tampouco será necessário se conectar a uma dessas redes *Wi-Fi* no ambiente.

1.2 Objetivo

O objetivo deste projeto é desenvolver um conjunto de ferramentas que possibilitem o mapeamento e a identificação de áreas dentro de ambientes fechados. Estas ferramentas serão utilizadas em dispositivos móveis, possibilitando que os usuários possam se localizar em locais fechados. Para chegar a esse objetivo, um sistema de Machine Learning utilizará os valores das potências das redes *Wi-Fi* presentes nos arredores para aprender a mapear diversas zonas no ambiente.

1.3 Justificativa

Levando em conta a falta de alternativas práticas para sistemas de posicionamento *indoor*, o *net.map* se mostra ideal para suprir essa demanda. O sistema pode ser utilizado por museus

(para tornar a experiência mais interativa) ou por *Shopping Centers* (para sugerir produtos diferentes de acordo com a localização do cliente). Esses dois exemplos de ambientes são tipicamente instalados em ambientes fechados, onde o GPS não funciona bem, e como consequência o *net.map* pode preencher essa lacuna funcionando como o sistema de localização padrão para esses lugares.

1.4 Organização

O restante do documento tem a seguinte estrutura: Na sessão 2 temos uma breve explicação de alguns conceitos fundamentais para o desenvolvimento do trabalho, e uma breve explicação de cada um dos modelos de *Machine Learning* usados. Na sessão 3 é detalhada a especificação do projeto. Na sessão 5 é documentado todo o estudo com o *Machine Learning* e seus respectivos resultados. São apresentados gráficos e justificativas para todo o tratamento e treinamento dos dados.

2 ASPECTOS CONCEITUAIS

Grande parte da pesquisa e fundamentação teórica feita sobre *machine learning* foi feita no livro "An Introduction to Statistical Learning" (JAMES et al., 2014) e no curso online ministrado por Andrew Ng (NG, 2008). A seguir são explicados detalhadamente alguns pontos teóricos importantes usados no trabalho.

2.1 Bias vs. Variância: Um tradeoff

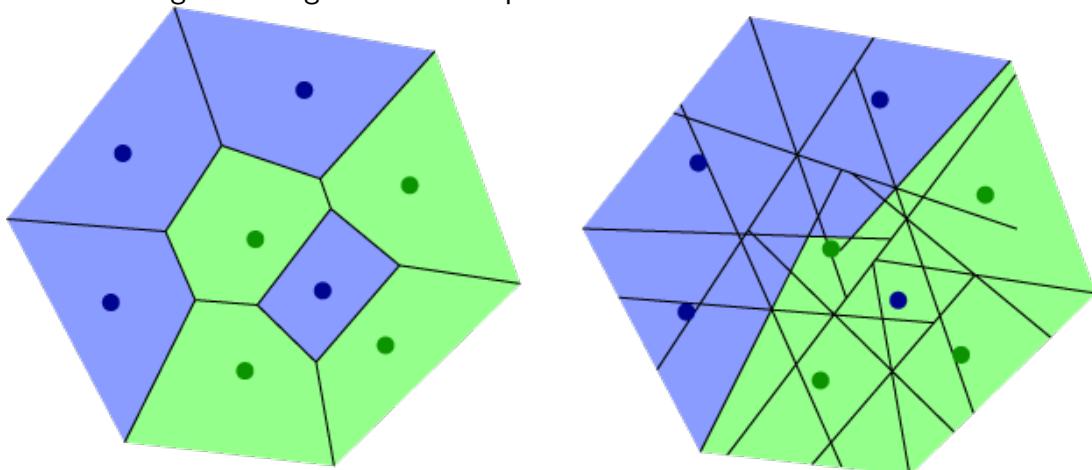
Um dilema comum que se aparece ao usar *Machine Learning* é o de *Bias* contra Variância. *Bias* é a medida do erro do modelo treinado e da realidade que nós tentamos modelar. Variância é o erro decorrente de pequenas flutuações nos pontos de treino, ou seja, diversos modelos treinados com pequenas mudanças dos pontos de treino irão gerar erros drasticamente diferentes. Uma maneira comum de generalizar o erro de modelos de *Machine Learning* é em dividi-lo em uma parcela de erro proveniente de *bias*, outra de variância e uma terceira parcela de ruído com média nula. Esse dilema aparece pela contradição clara entre desejar reduzir *bias* e variância ao mesmo tempo. Um modelo com baixo *bias* é mais complexo, (e.g. um polinômio de ordem alta) de modo que a curva representa de maneira mais próxima os pontos de treino, porém, isso também acaba por fazer que o ruído desse conjunto particular de pontos de treino seja capturado pelo modelo, o que não é desejado, visto que queremos que o modelo se adeque a *qualquer* conjunto de pontos, não apenas os que foram usados para treiná-lo. A solução poderia ser diminuir então a complexidade do modelo, mas isso invariavelmente aumenta nosso erro de *bias*, por mais que diminuia o de variância.

2.2 Algoritmos de Machine Learning

2.2.1 KNN

O Algoritmo KNN busca criar regiões de classificação no espaço pelo voto majoritário dos K pontos de treino mais próximos, sendo K um parâmetro do algoritmo. Na imagem a seguir podemos ver essa classificação em duas dimensões, supondo que os pontos roxos representam uma classe, e os verdes, outra. A imagem da esquerda representa a saída do algoritmo para um valor de K escolhido pequeno, e a da direita, para um valor grande K , com o risco de *overfit*. Pode-se reparar que com um K maior, um ponto solitário que pode representar um erro dos dados de treino é ignorado por estar cercado de pontos que representam outra classe.

Figura 1: Algoritmo KNN aplicado com diferentes valores de K

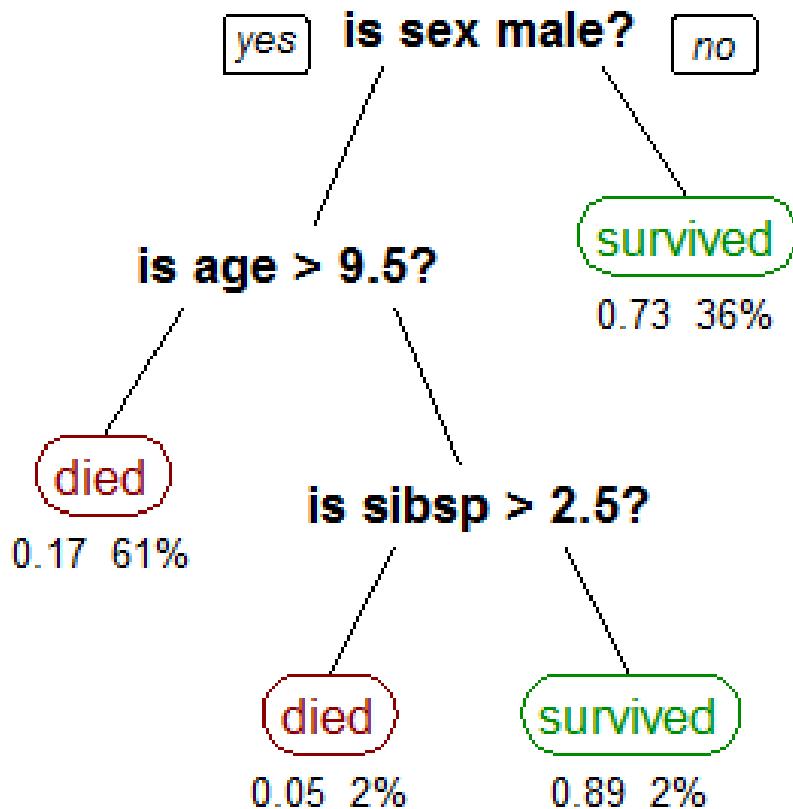


2.2.2 Árvore de Decisão

Em sua forma mais simples, árvores de decisão são uma classe de algoritmos que buscam (no caso de classificação) achar a classe de um ponto de treino testando intervalos de suas *features*. Vejamos o exemplo de uma árvore treinada a seguir:

Essa árvore busca prever se um determinado passageiro do Titanic sobreviveu ou não ao acidente testando diversas características desse indivíduo. Por exemplo, podemos ver que caso o sexo do passageiro seja feminino, ela tem uma alta chance de ter sobrevivido. Caso contrário, já são testadas outras duas cláusulas referentes à idade ao número de familiares também a bordo do navio. Por construção, temos um *trade-off* para árvores, podendo trocar **legibilidade** por **precisão** (menos *bias*). É possível construir árvores mais complexas que poderão a sua facilidade de interpretação por uma pessoa, porém poderão se adequar melhor aos dados treinados, e é essa ideia que é explorada por algoritmos de *ensemble learning* como

Figura 2: Árvore de Decisão para prever a sobrevivência de passageiros do Titanic

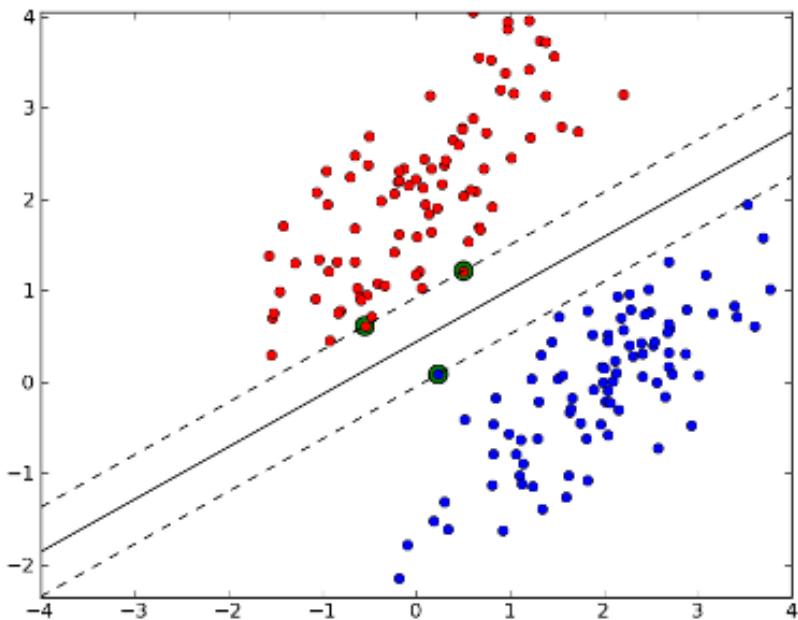


Random Forests ou *Adaboost*, que serão explicados em outra parte desse documento, mas que basicamente combinam o poder de diversas árvores treinadas iterativamente, para aumentar seu poder de classificação.

2.2.3 Support Vector Machines

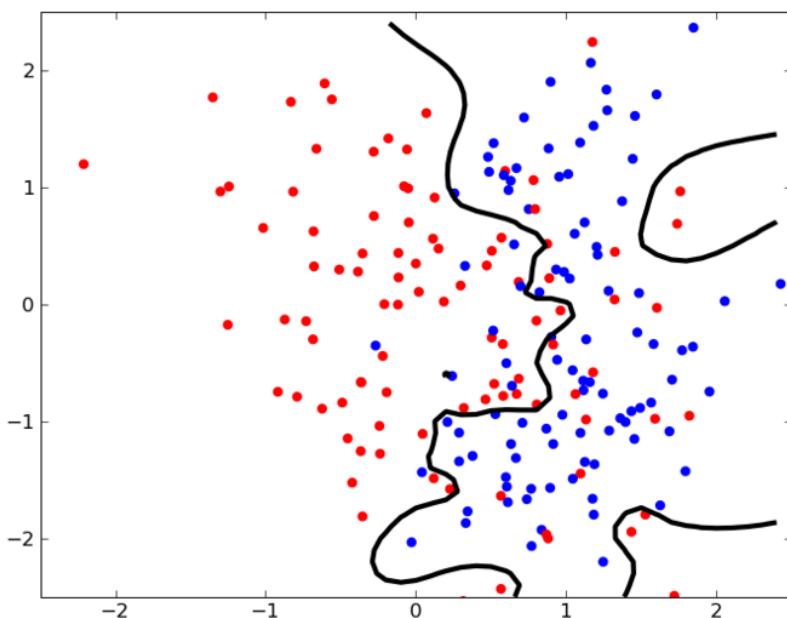
Essa classe de algoritmos busca, para um conjunto de dados N-dimensionais, encontrar um hiperplano que separa o espaço dos dados em regiões de classificação. Vejamos um exemplo simples a seguir:

Figura 3: SVM aplicado para classificar dados linearmente separáveis



A reta na figura 3 foi encontrada de modo a separar o espaço de classificação da classe "azul" do espaço de classificação da classe "vermelha". A linha pontilhada representa a margem de classificação, o algoritmo busca chegar a um hiperplano que tenha ainda uma distância dos pontos mais próximos da região de separação. A seguir, um exemplo mais complexo usando um *kernel* para poder classificar dados não linearmente separáveis:

Figura 4: SVM aplicado para classificar dados de maneira não linear



2.3 Aplicação na Prática

No estudo e desenvolvimento da pesquisa, foi decidido finalmente combinar esses três algoritmos. Usaremos as conclusões levantadas por (MACLIN; OPITZ, 2011), onde foi averiguado empiricamente que:

1. Combinar as saídas de diversos classificadores pode reduzir o risco de selecionar um único que possa funcionar mal naquele caso específico
2. Os erros cometidos advindos de um classificador são geralmente compensados pela classificação correta de outro algoritmo no conjunto, de modo que a taxa de acerto *do sistema* melhore
3. Classificadores base devem ser diversos em natureza para que a decisão final não sofra de *bias*

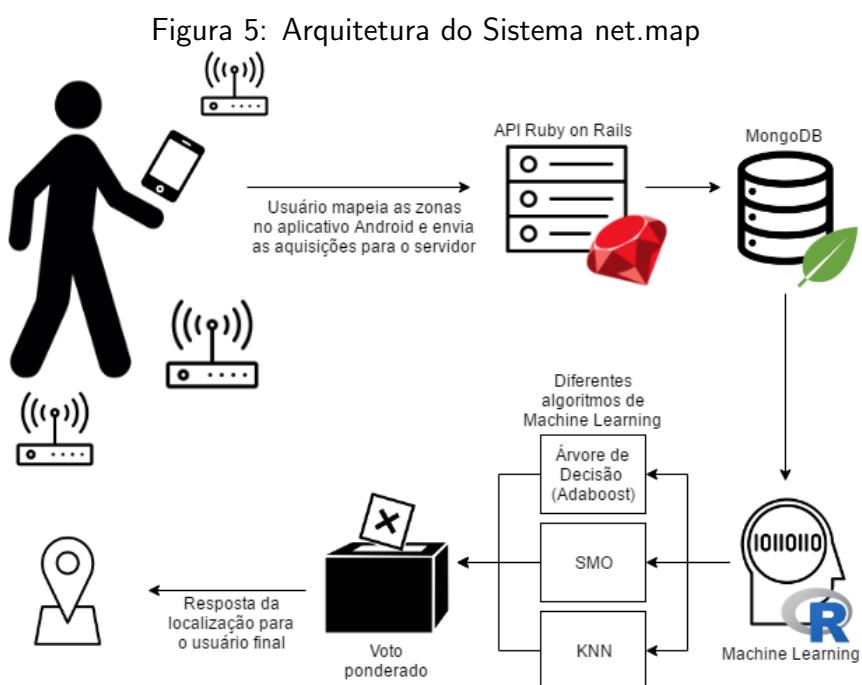
3 ESPECIFICAÇÃO

3.1 Arquitetura

O sistema net.map precisa adquirir dados de potência dos sinais de redes *Wi-Fi* para funcionar. Sendo assim, é necessário um dispositivo físico capaz de conseguir esses dados. Tendo em vista que os *smartphones* possuem a capacidade de ler informações sobre as redes *Wi-Fi* no ambiente ao seu redor, se tornam o dispositivo perfeito tanto para fazerem o mapeamento do ambiente quanto para ser utilizado pelo usuário final.

Tendo isso em mente, foi desenvolvido um aplicativo para dispositivos Android capaz de mapear ambientes e de verificar a localização atual. Esse *app* se comunica com uma API rodando em um servidor na nuvem, responsável por fazer processamento dos dados utilizando algoritmos de *Machine Learning*. Após processados, o resultado solicitado pelo usuário é retornado ao seu *smartphone Android*.

O diagrama a seguir ilustra bem a arquitetura básica do sistema:



3.2 Requisitos

3.2.1 Requisitos Funcionais

3.2.1.1 Aplicativo

- Enviar dados sobre a intensidade dos sinais de *Wi-Fi* ao servidor.
- Retornar a posição do usuário baseado nos dados enviados.
- O usuário deve inserir seu usuário e senha para poder usar o serviço.

3.2.1.2 Servidor

- A API deve funcionar não apenas com o aplicativo, mas com qualquer outra aplicação que utilize o protocolo HTTP.
- A API deve receber dados no formato JSON.
- Cada *Facility* deve ter ao menos 2 zonas.
- Cada Zona deve ter ao menos 4 medidas.

3.2.2 Requisitos Não-Funcionais

3.2.2.1 Aplicativo

- O usuário deve ter um *smartphone Android* com pelo menos a versão 4.1 instalada, além do aplicativo net.map.
- O usuário deve ter a capacidade de se conectar a *internet* enquanto realiza suas medições.

3.2.2.2 Servidor

- A API deve ser transparente ao usuário.
- O usuário deve solicitar ao administrador do sistema um login e senha.
- O conjunto de algoritmos de *Machine Learning* usados devem ser uma taxa de acerto melhor que cada um individualmente (MACLIN; OPITZ, 2011)

3.3 Casos de Uso

3.3.1 Captura e Treinamento

No primeiro caso de uso do sistema net.map, temos um usuário desejando mapear uma certa **instalação** (*facility*, como é chamada no sistema). Ele deve planejar um mapa dividindo a instalação em seções menores, chamadas de **zonas**. Exemplo:

Tabela 2: Exemplo de divisão de instalação em zonas

Instalação: Museu Paulista da USP	
Zona 1	Saguão de entrada
Zona 2	Exposição sobre Dom Pedro I
Zona 3	Sala do quadro <i>Independência ou Morte</i>
Zona 4	Exposição sobre maquinário agrícola do séc. XIX
	...
Zona n	Exposição x

As zonas idealmente são melhores definidas como salas separadas por divisões físicas, tais como paredes ou andares. O confinamento dos sinais eletromagnéticos e a atenuação desses sinais ao atravessar paredes garante um resultado mais preciso. Porém, é possível delimitar linhas imaginárias para essas zonas, podendo demarcar zonas em grandes áreas sem paredes. O ideal é que essas zonas tenham uma área maior, garantido assim um resultado preciso.

Após demarcar as zonas, é necessário configurar alguns parâmetros no aplicativo: número de aquisições por ponto e método de normalização.

Número de aquisições por ponto

Para mapear uma zona, andamos por ela com o aplicativo de celular aberto, escaneando pontos discretos dentro dela. Como a potência do sinal das redes *Wi-fi* é muito flutuante, o ideal é fazer uma média de várias aquisições para normalizar o valor lido naquele ponto. Isso é feito automaticamente pelo aplicativo.

Método de normalização

Conforme explicado no parágrafo acima, é necessário normalizar os dados adquiridos devido à flutuação da potência de sinal. Esses dados podem ser normalizados através de uma média aritmética simples ou através de um método matemático chamado **Filtro de Kalman**. Este último será detalhado mais a frente. Um comparativo dos dois existe no capítulo 6.

Terminando a configuração, é necessário percorrer a zona desejada, fazendo aquisições em

pontos discretos dentro da zona. Idealmente, quantos mais pontos são capturados, melhor é a precisão da resposta final do sistema.

3.3.2 Localização

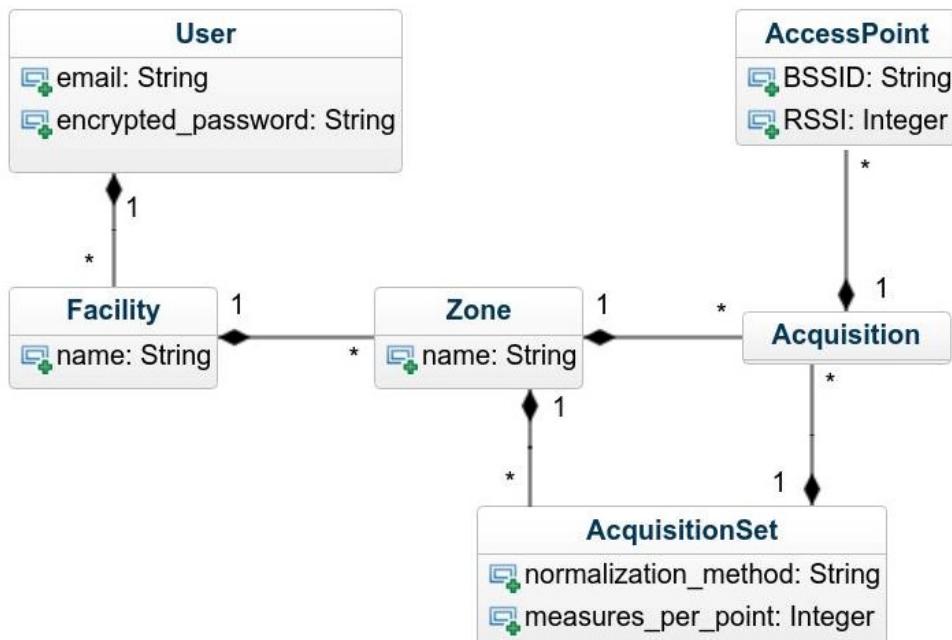
O modo de localização de usuário interage com a API instalada no servidor em nuvem, podendo assim ser implementado em qualquer dispositivo capaz de receber sinais *Wi-Fi*. Utilizando uma aplicação que se comunica com a API, o usuário recebe a localização do servidor a cada novo escaneamento que é feito pela aplicação.

3.4 Modelo de classes

Como neste projeto utilizamos o MongoDB (banco de dados não-relacional orientado a documentos JSON), o conceito de entidade-relacionamento existente em bancos de dados SQL não faz muito sentido.

Uma boa maneira de representar quais dados persistentes fazem parte do projeto é gerar um diagrama de classes da API no servidor:

Figura 6: Diagrama das classes na API



4 METODOLOGIA

4.1 Gerenciamento de tarefas

O projeto desde a sua idealização foi gerenciado com metodologia *Kanban*, com a equipe sempre desenvolvendo *features* incrementais com pequenos protótipos entregáveis. A ferramenta de gestão utilizada pelo grupo foi o Trello, aplicação de gerenciamento de projetos baseado na web. Foi sempre feita a distinção entre pesquisa, documentação e implementação. Como é típico de metodologias ágeis, os testes sempre foram realizados em paralelo com a implementação de cada nova *feature*, de modo que para cada nova parte do projeto implementada, foram realizados testes para garantir que as funções antigas ainda funcionam individualmente.

4.2 Gerenciamento de código

Para hospedar e versionar os códigos-fonte dos programas foram utilizados repositórios do GitHub, serviço Git baseado em nuvem, gratuito para softwares *Open-Source*. O Git consiste num sistema de controle de versão de software distribuído, com objetivo de minimizar conflitos entre códigos de diferentes contribuidores em um mesmo repositório.

4.3 Pesquisa

A pesquisa foi feita simultaneamente com o desenvolvimento, com novas possibilidades sendo estudadas e implementadas, às vezes ao mesmo tempo. Foram de vital importância as ideias obtidas de *papers* nas partes de *ensemble learning* (MACLIN; OPITZ, 2011), normalização e tratamentos dos dados, bem como a estrutura dos testes de *cross validation* (BOZKURT et al., 2015), e também na fundamentação teórica do método *Adaboost* (SCHAPIRE, 2013) e (FREUND; SCHAPIRE, 1996).

5 PROJETO E IMPLEMENTAÇÃO

5.1 Machine Learning

5.1.1 Tratamento dos dados

5.1.1.1 Dados obtidos do Repositório UCI

Para grande parte dos testes de *cross-validation*, foi usado o *dataset* *UJIIndoorLoc* apresentado em (TORRES-SOSPEDRA et al., 2014). O *dataset* consiste em diversas medidas de potência de sinal *Wi-Fi* medidas com diversos aparelhos celulares em 3 prédios diferentes de uma faculdade. As medidas estão separadas por prédio, andar e sala. Para os fins dessa monografia, serão realizados testes com medidas sempre de um mesmo andar, afim de classificar as zonas de um mesmo ambiente. Esse *dataset* usa a constante 100 para designar valores não medidos. Para começar o tratamento os substituímos pelo valor de -120 (dB), que para todos os fins práticos, é uma medida de potência que representa um valor nulo, já que representa um valor de potência insignificante. E finalmente, antes de serem usados para treinar os modelos, os dados são transformados de modo a ficarem com média 0 e variância 1 por coluna, o que é necessário para evitar comportamentos indesejados dos algoritmos de ML.

A transformação é realizada da seguinte maneira, com X_j representando a j -ésima coluna da matriz de dados, com média μ e desvio-padrão σ , para cada elemento i dessa coluna:

$$X_{ij} = \frac{X_{ij} - \mu}{\sigma}$$

5.1.1.2 Dados do Servidor

De modo análogo aos dados pegos do Repositório UCI, a matriz de dados é transformada para ter média nula e variância unitária em cada coluna, sendo que antes disso valores nulos são substituídos por -120 .

5.1.2 Formato dos dados tratados

A matriz de dados, depois de tratada, tem o seguinte formato:

$$\begin{array}{c|ccccc|c} & ZoneID & BSSID_1 & BSSID_2 & \dots & BSSID_n \\ \hline & 1 & -70 & -92 & \dots & -87 & Measure_1 \\ & 2 & -89 & -80 & \dots & -63 & Measure_2 \\ & 3 & -28 & -120 & \dots & -35 & Measure_3 \\ & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ & 1 & -48 & -36 & \dots & -29 & Measure_n \end{array}$$

Cada coluna representa a medida de uma *BSSID* (i.e. um Roteador), ou seja, uma *feature* para o ML, e cada linha é um ponto de treino (i.e. uma medida para cada *BSSID*, nossas *features*).

5.1.3 Modelos Usados e Cross-Validation de parâmetros

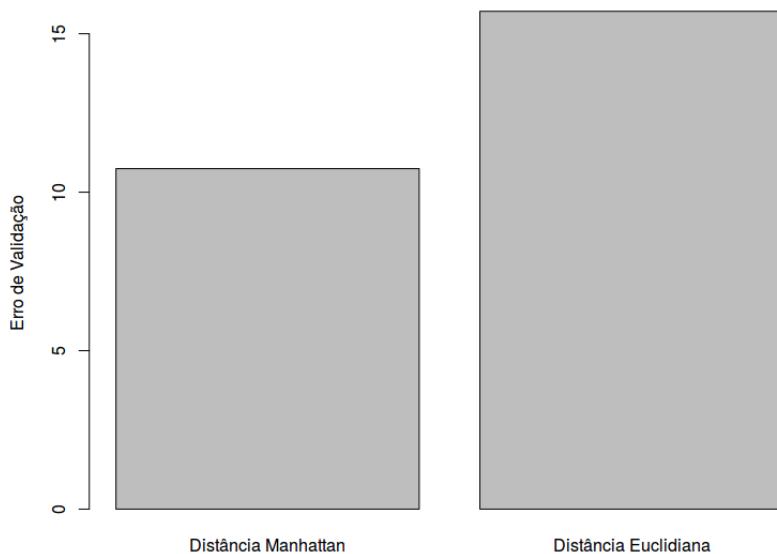
A chamada *cross-validation* dos modelos de ML é o teste para encontrar os parâmetros ótimos para o treinamento. Um dos métodos escolhidos de *cross-validation* foi o *k-fold*. O método *k-fold* divide o dataset em K subconjuntos de igual tamanho e então um dos conjuntos é usado como validação do treinamento feito pelos $K - 1$ subconjuntos restantes. O processo é repetido K vezes e em cada subdivisão possível podem ser usados valores de parâmetros de treino distintos. Para a comparação dos modelos, serão feitos diversos testes com um número crescente de zonas de classificação. Nesse caso, são escolhidas iterativamente e aleatoriamente n zonas do dataset *UJIIndoorLoc*, são pegos todos os pontos associados as n zonas selecionadas. Então, o dataset é separado em 80% de pontos de treino e 20% para testes. Para valores de n de 1 a 30 são finalmente medidos as taxas de erro de cada algoritmo e essa informação é apresentada em gráficos.

5.1.3.1 KNN : K-Nearest-Neighbors

Para o algoritmo de KNN, foram feitos dois testes de *cross-validation*. No primeiro, a métrica de distância foi decidida, e no segundo, o número de vizinhos (K). Para que fosse definida a métrica de distância, foi usado o método *k-fold* com $K = 2$. E em cada *fold* o modelo foi treinado com uma métrica diferente. Foram calculados 20 conjuntos diferentes de *folds* e foi tirada a média do erro de validação para cada uma das métricas.

Figura 7: Erros para uso de Diferentes Distâncias

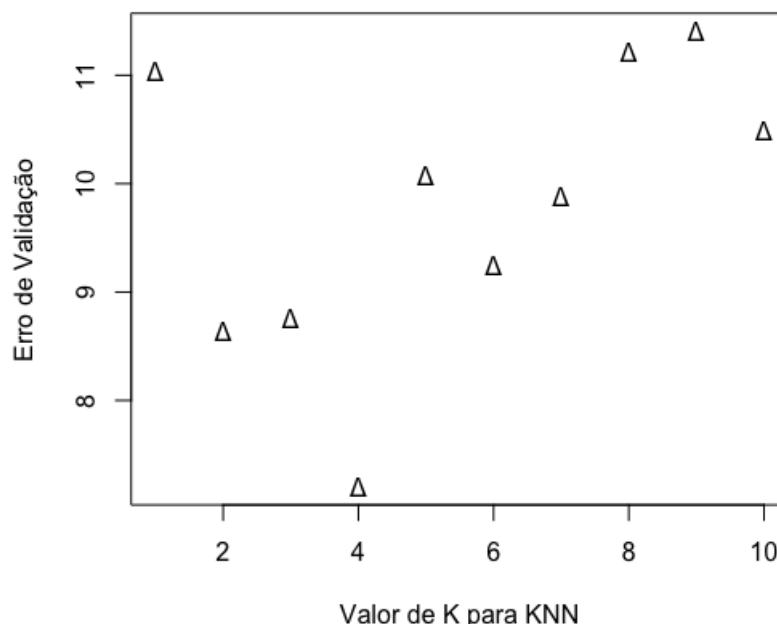
Diferença do Erro de Validação para Diferentes Métricas de Distância KNN



Depois, foi usado o método *k-fold* com $K = 10$ para a decisão do número de vizinhos. Foram testados os valores de 1 até 10 para o número de vizinhos. Na imagem a seguir vemos detalhes desse teste.

Figura 8: Erro em função do número de vizinhos

**Média de 20 testes de Cross-Validation 10-Fold para KN
25 Zonas Usadas**



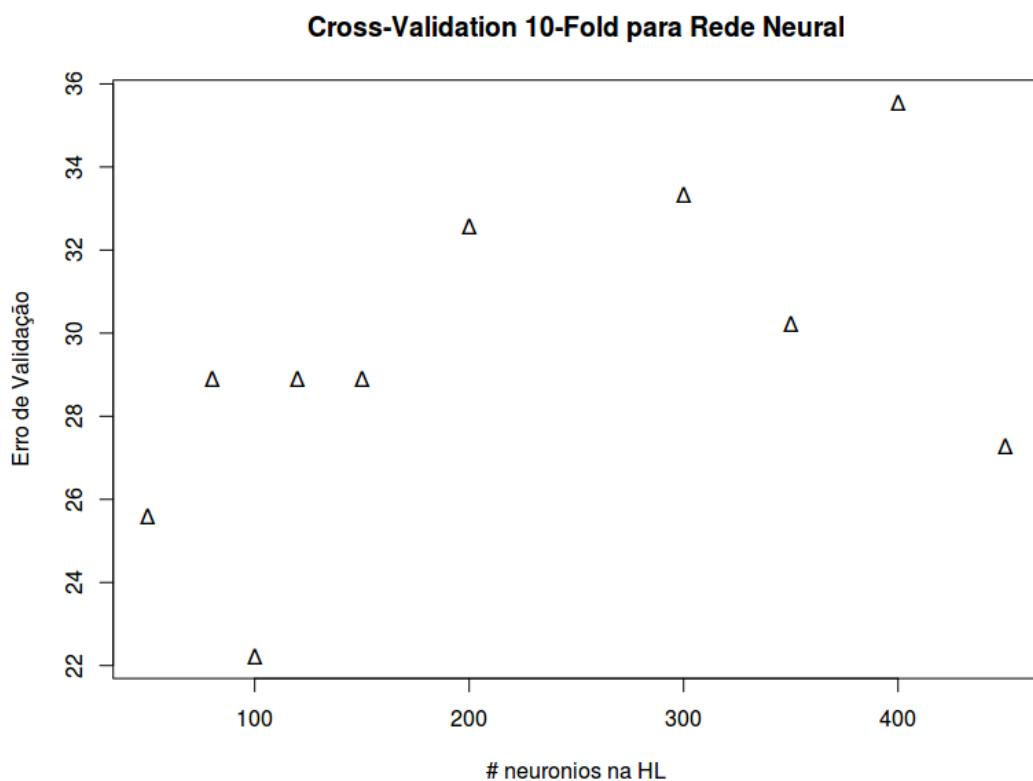
Podemos concluir então que (1) o erro é menor no geral com a distância de *Manhattan* e (2) embora tenha muita variância envolvida no teste, o valor de K que minimiza o erro fica próximo de 4 vizinhos.

No restante do trabalho foi escolhido o valor de $K = 4$ vizinhos.

5.1.3.2 Rede Neural

Para a *cross-validation* das Redes Neurais, também foi usado o método de *k-fold* com $K = 10$. Para cada uns dos *folds* foi testado um número de neurônios em uma *hidden-layer* única. (Outros testes mostraram não valer a pena para o nosso problema usar mais de uma camada de *hidden layer* ou *deep learning*). Os resultados são mostrados a seguir:

Figura 9: Erro para diversas quantidades de neurônios na rede neural.



Concluímos que o número ideal de neurônios na *hidden-layer* está próximo de 200. Porém, com estudos e testes mais detalhados ficou claro que as Redes Neurais não são ideias para o nosso projeto, por não conseguirem capturar de forma satisfatória o modelo do nosso problema. Além disso, o algoritmo de *back propagation* se mostra muito mais custoso em relação a memória e processamento se comparado aos outros algoritmos usados no projeto, demorando cerca de 10 vezes mais para o treinamento que todos os outros modelos juntos. Portanto, seu

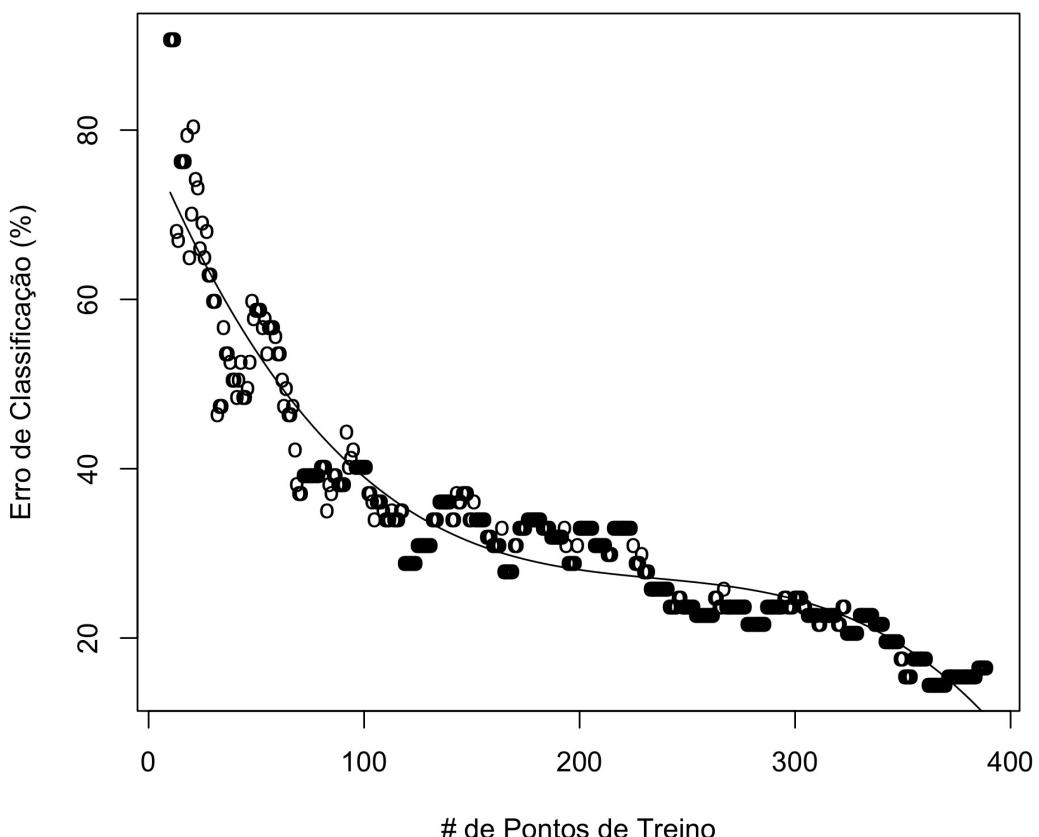
uso foi descartado.

5.1.3.3 Árvore de decisão

Baseado no trabalho apresentado por (BOZKURT et al., 2015), foi escolhida uma implementação do algoritmo C4.5 (QUINLAN, 1993) em Java, da biblioteca Weka, para serem geradas árvores de decisão. Os métodos porém foram chamados pela interface da Weka para R, chamada RWeka. A função J48 do Weka foi testada com os pontos de treino referentes ao primeiro andar do prédio 1 (i.e. BuildingID 1, FloorID 0) do dataset *UJIIndoorLoc*. Foi levantada a curva de erro de classificação para uma quantidade crescente de pontos de treino (como explicado em 2.3).

Figura 10: Teste de Validação do Algoritmo C4.5

Erro de Classificação para J48 (C4.5)



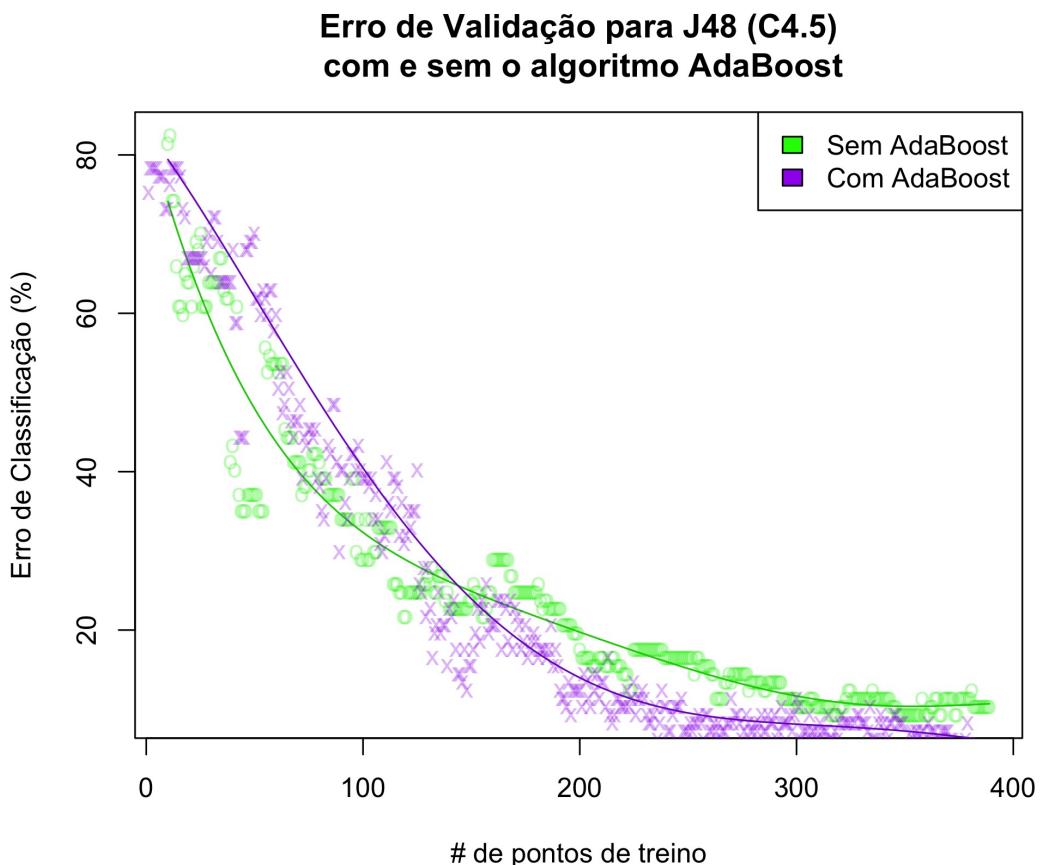
Acoplamento com o Algoritmo AdaBoost

Boosting é a ideia em ML de criar uma predição forte e precisa combinando diversas

previsões mais fracas. O algoritmo AdaBoost (FREUND; SCHAPIRE, 1996) é um método para melhorar a precisão de classificadores fracos (i.e. *Weak Learners*) por meio de uma votação ponderada que leva em conta o erro de diversos votadores fracos treinados no processo. É provado que o modelo final se torna um classificador forte (SCHAPIRE, 2013). Por definição, um classificador fraco é aquele que consistentemente consegue ser melhor que um chute para a classificação (e.g. o erro é menor que 50% para o caso de duas classes possíveis de classificação).

Também baseados em (BOZKURT et al., 2015) e (MACLIN; OPITZ, 2011). Usaremos o algoritmo AdaBoost e sua implementação na biblioteca Weka para melhorar a precisão das árvores de decisão C4.5. A seguir vemos um gráfico comparando a classificação para o mesmo andar do dataset *UJIIndoorLoc* com os mesmos pontos de teste e uma quantidade crescente de pontos de treino, assim como feito no tópico anterior.

Figura 11: Comparação do C4.5 com o uso do AdaBoost



Podemos notar que com um número grande de pontos de treino, obtemos uma melhora sensível de precisão para a classificação, e portanto, no restante do trabalho, iremos usar o algoritmo C4.5 acoplado com o algoritmo AdaBoost.

5.1.3.4 Comparação dos Modelos Usados

Reiterando o que foi explicado em outra sessão, os testes mostrados nas imagens 12 à 15 irão contemplar todos os modelos usados. Os modelos tem seu erro testado para *datasets* com um número crescente de zonas aleatórias a serem classificadas. Todos os testes foram feitos com os dados do primeiro andar do prédio 1 do *dataset UJIIndoorLoc* (i.e. BuildingID 1, FloorID 0).

Figura 12: Erro do algoritmo SMO para uma quantidade crescente de zonas de classificação.

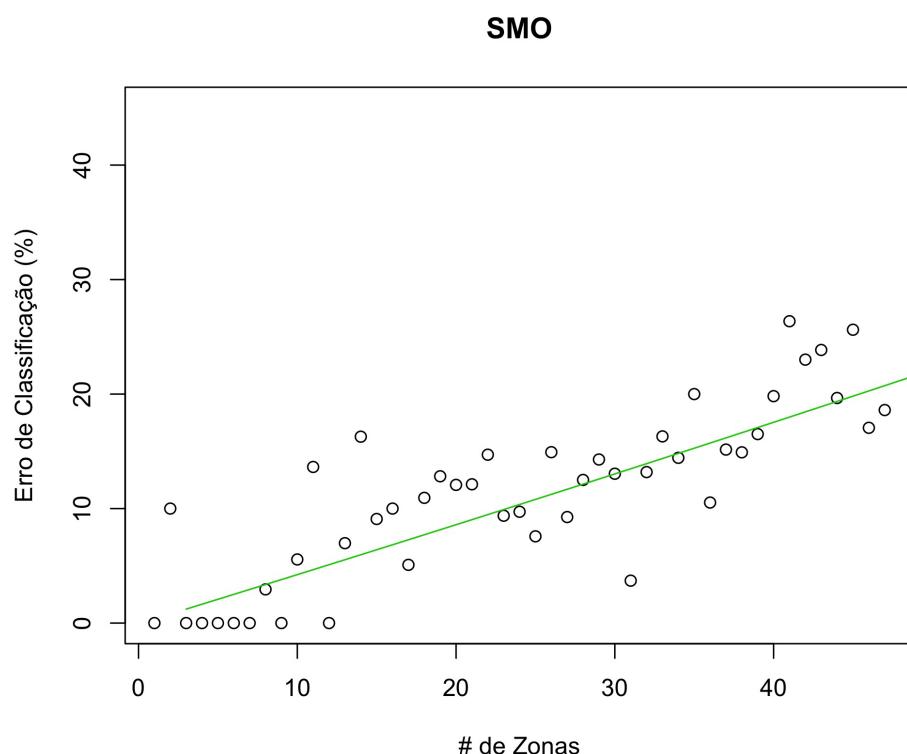


Figura 13: Erro do algoritmo de Redes Neurais para uma quantidade crescente de zonas de classificação.

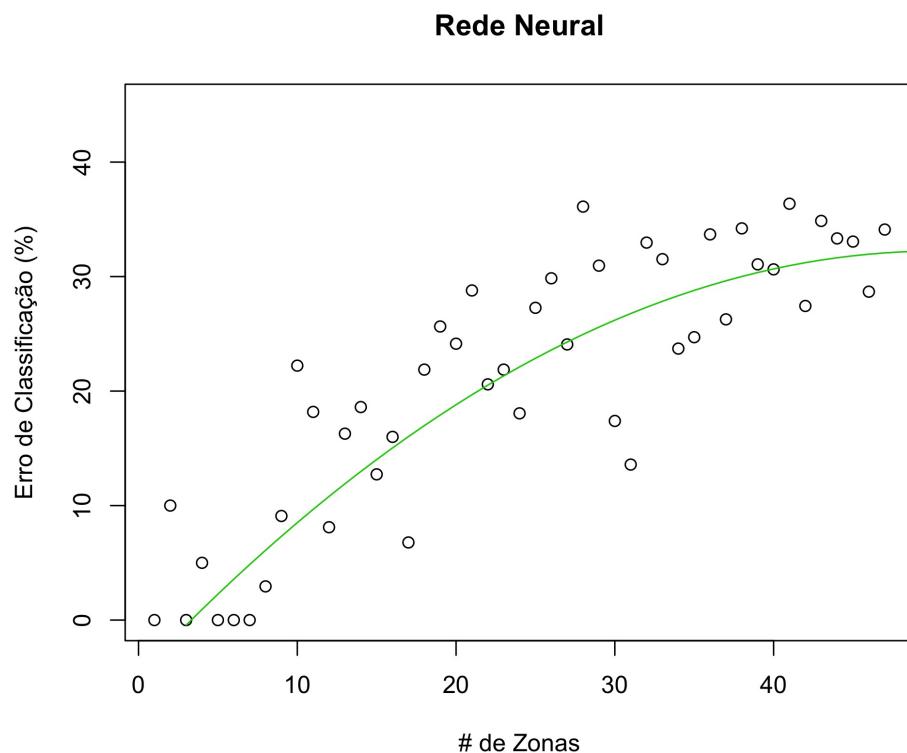


Figura 14: Erro do algoritmo KNN para uma quantidade crescente de zonas de classificação.

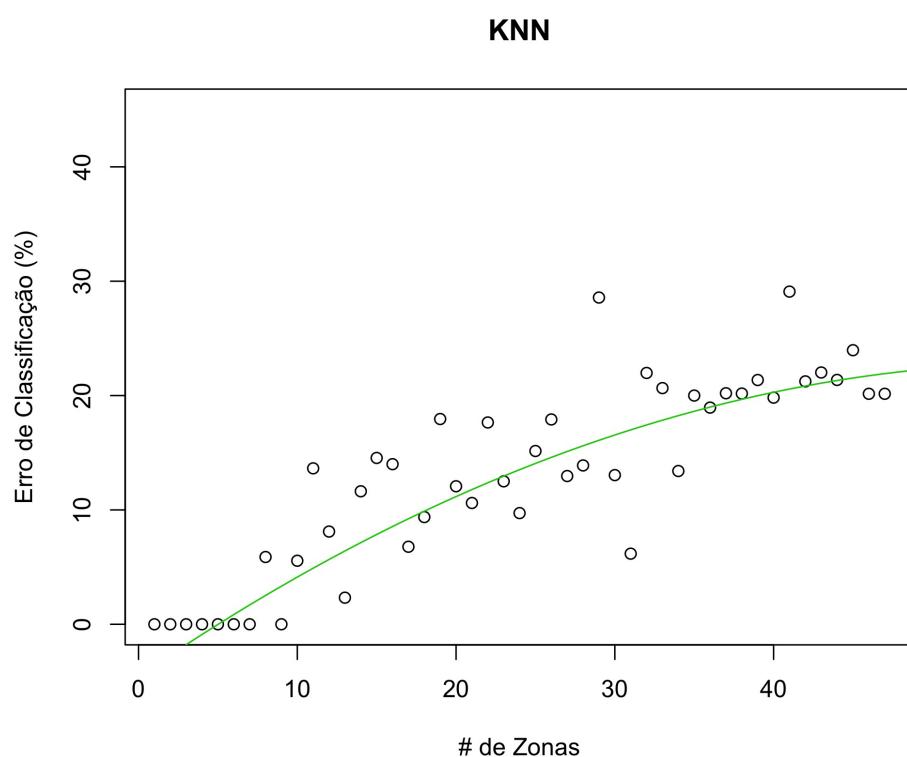
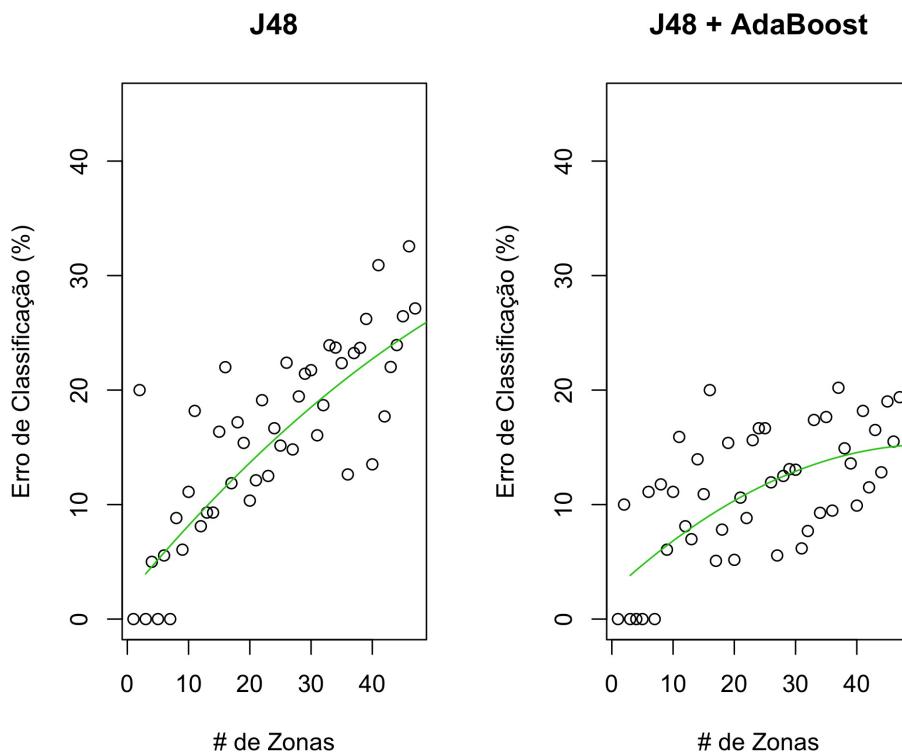


Figura 15: Erro do algoritmo de Árvore de Decisão com e sem o AdaBoost para uma quantidade crescente de zonas de classificação.



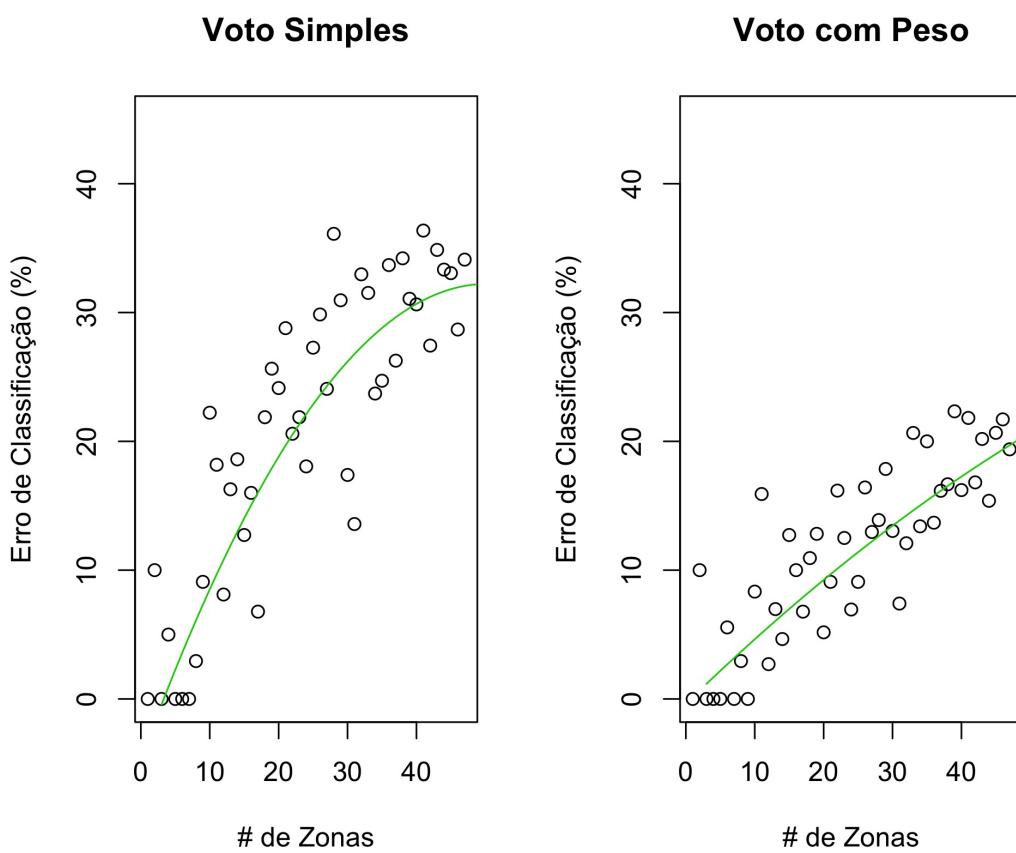
5.1.4 Métodos de Votação

Os chamados sistemas de *Ensemble Learning* são aqueles em que uma classificação é feita com base em diversos modelos treinados. Em tópicos anteriores foi discutido o uso do algoritmo AdaBoost, que é um método desse tipo para melhorar o poder de classificadores fracos treinando iterativamente diversos modelos fracos progressivamente com os erros do anterior (SCHAPIRE, 2013). Porém, para agregar os modelos usados até agora, iremos implementar algo mais simples, porém seguindo a mesma lógica: Sistemas de Votação. Baseados em (NAGI; BHATTACHARYYA, 2013), usaremos diversos métodos para combinar as classificações dos nossos modelos treinados. Em um, foi usado apenas a classe de saída dos modelos, e posteriormente, as probabilidades (*supports*) para cada uma das classes, que também são saídas dos modelos. Esses *supports* podem ser, dependendo do modelo, a probabilidade posterior ou o grau de confiança fornecido pelos algoritmos. Por exemplo, nas redes neurais a saída de um neurônio é a função sigmoide aplicada nas entradas multiplicada pela matriz de pesos da rede, que, por definição, é um número entre 0 e 1 que representa uma probabilidade.

5.1.5 Testes com os Métodos de Votação

Como na sessão de comparação dos modelos, os métodos tem seu erro testado para datasets com um número crescente de zonas aleatórias a serem classificadas. Todos os testes foram feitos com os dados do primeiro andar do prédio 1 do dataset *UJIIndoorLoc* (i.e. BuildingID 1, FloorID 0).

Figura 16: Erro do Voto Simples e Voto Ponderado para uma quantidade crescente de zonas de classificação.



5.2 Aquisição de dados

Conforme explicado no capítulo 3, as informações de redes *Wi-Fi* são obtidas através de um dispositivo Android. Ao tocar no botão de captura num aplicativo instalado no dispositivo, o sistema aguarda o sensor *Wi-Fi* atualizar sua lista de redes disponíveis para conexão, e assim que essa lista é atualizada, os dados obtidos são armazenados. O processo é repetido n vezes, n sendo o número de amostras por ponto determinado pelo usuário. Após isso, os dados são normalizados, seja com uma média simples ou com o uso filtro de Kalman, explicado em 5.2.1.

5.2.1 Filtro de Kalman

O filtro de Kalman é um método estatístico cujo propósito é utilizar medições de grandezas realizadas ao longo do tempo (contaminadas com ruído e outras incertezas) e gerar resultados que tendem a se aproximar dos valores reais (média da população) das grandezas medidas.

Apesar de normalmente ser utilizado em tempo real, nada impede que o filtro seja aplicado em valores já medidos. Como é possível entender em (BULTEN; ROSSUM; HASELAGER, 2016), o filtro de Kalman é um método muito utilizado na obtenção de valores de ondas eletromagnéticas (como as ondas geradas por *Access Points Wi-Fi*, pois estas apresentam muita flutuação nos seus valores associados).

O filtro de Kalman é capaz de fazer uma média ponderada dos sinais lidos, dando um peso menor para aquelas amostras que se afastam muito da média de todos os sinais.

Um comparativo entre o uso do filtro de Kalman e uma média aritmética simples existe na sessão 7.1.

5.3 Estrutura do servidor

O *back end* do net.map está hospedado numa instância em nuvem do *Amazon AWS EC2*, serviço de cloud gratuito para estudantes. Nesse servidor se encontram o banco de dados MongoDB, a API Rails e os *scripts* de *Machine Learning* escritos em R, rodando no sistema operacional Ubuntu 14.04, baseado em Linux.

A API Rails, que opera em TCP na porta 3000 é responsável por:

1. Criar novos usuários
2. Enviar *token* de acesso após o *login* do usuário
3. Criar, modificar e deletar instalações e zonas enviadas pelos usuários
4. Armazenar dados de captura enviados pelo aplicativo Android

Todos esses dados são salvos na instância do MongoDB rodando no mesmo servidor. As senhas criadas pelos usuários são criptografadas utilizando o Bcrypt, um esquema de *hashing* baseado no Blowfish (PROVOS; MAZIERES, 1999).

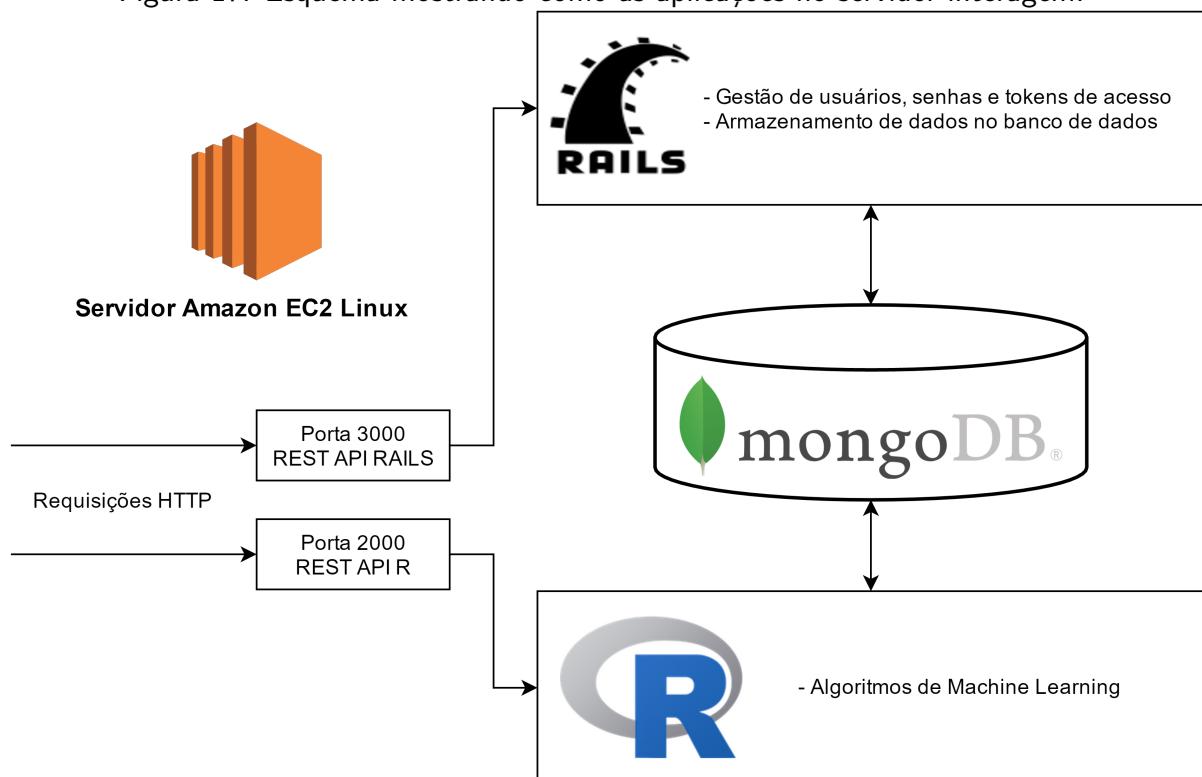
Uma biblioteca *open-source* chamada Devise¹ é responsável pelo gerenciamento dos usuários,

¹Disponível em <https://github.com/plataformatec/devise>

suas senhas e seus tokens de acesso. Tudo isso pode ser feito se comunicando com a API através de requisições HTTP. Pode-se considerar que a API aqui apresentada é RESTful, ou seja, implementa todas as suas operações através de uma API REST.

Os scripts de Machine Learning escritos em R rodam em segundo plano como processos *daemonizáveis*, com a ajuda de uma biblioteca *open-source* chamada Plumber². Essa mesma biblioteca também atua como uma API RESTful na porta TCP 2000, possibilitando que o usuário treine os *datasets* no banco de dados e que faça uma teste da zona atual após enviar uma requisição POST contendo dados de potência de sinais *Wi-Fi*. A imagem 17 mostra como o módulo de ML e a API interagem entre si e com o ambiente externo.

Figura 17: Esquema mostrando como as aplicações no servidor interagem.



²<https://github.com/trestletech/plumber>

6 APLICAÇÕES DO NET.MAP

Este capítulo tem por objetivo mostrar duas implementações reais do sistema apresentado neste documento, mostrando que o sistema não existe apenas no campo teórico e afirmando seu potencial impacto social e comercial.

6.1 APScanner

APScanner é um aplicativo Android feito para a aquisição e treinamento dos dados para o net.map. Nele é possível criar instalações e zonas, além de capturar os sinais de *access points Wi-Fi* ao redor e enviá-los para o servidor do sistema. Também é capaz de mostrar a zona na qual o usuário se encontra no momento.

As telas do aplicativo APScanner estão disponíveis no anexo A.

6.2 EletricaGO

Em julho de 2016 foi lançado ao redor do mundo o jogo de realidade aumentada Pokémon GO. Se trata de um jogo eletrônico *free-to-play* voltado para *smartphones*, onde os usuários andam pelas ruas buscando *pokémons*, as criaturas existentes no jogo, utilizando para isso o GPS de seus *smartphones*. Com mais de 500 milhões de downloads até novembro de 2016, o jogo se provou um sucesso comercial, e abriu as portas para que mais aplicações de realidade aumentada sejam desenvolvidas (PIMENTA, 2016).

Tendo em vista a popularidade do Pokémon GO, decidiu-se replicá-lo parcialmente para ser apresentado em conjunto com o resto deste projeto, mudando o contexto da aplicação. Chamado de **EletricaGO**, ao invés de funcionar em ruas e parques, funcionará dentro do prédio e Engenharia Elétrica da Escola Politécnica da USP. E ao invés de utilizar o GPS, utilizará o net.map, sistema apresentado neste documento.

No EletricaGO, o usuário terá uma lista de algumas salas e ambientes do prédio de En-

genharia Elétrica, e ao lado do nome de cada zona, um nome de um *pokémon* que pode ser encontrado nessa zona. Ao chegar na zona, o usuário pode iniciar a captura *pokémon*, conforme mostra a figura 18.

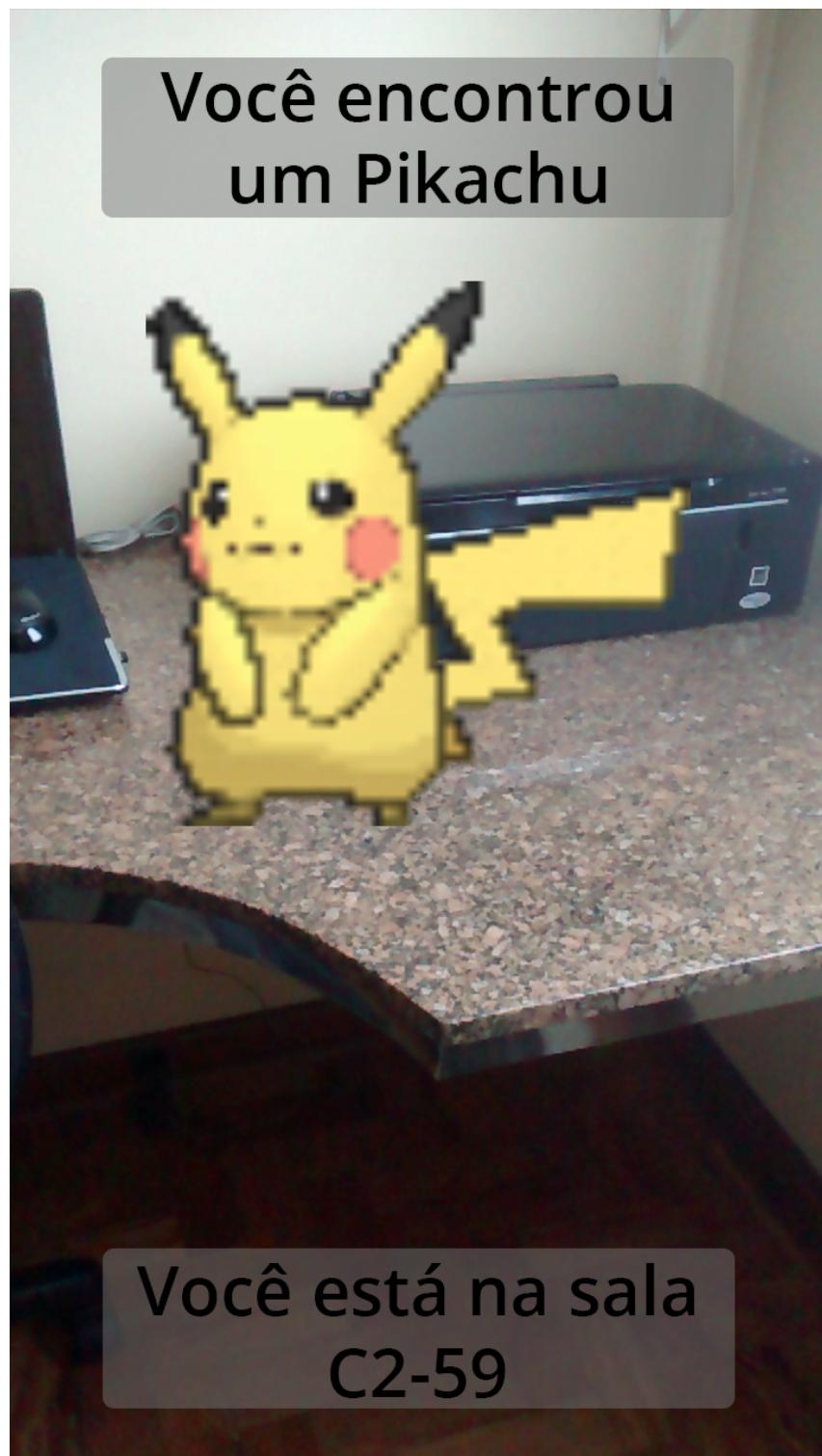


Figura 18: Tela do EletricGO mostrando um *Pokémon* na sala C2-59 do prédio de Engenharia Elétrica da POLI-USP

7 TESTES IN LOCO E AVALIAÇÃO DE RESULTADOS

7.1 Testes in loco

O sistema foi testado no bloco B do prédio da Engenharia Elétrica da Escola Politécnica, e a seguir são reproduzidos os resultados de precisão que foram obtidos. Um detalhe a ser mencionado é que os testes foram feitos com os dois métodos de normalização de dados que foram estudados (média aritmética e filtro de Kalman, cada um com um mapeamento e testes independentes).

No anexo B se encontram as plantas do primeiro e segundo andar do prédio, junto com os pontos onde foram efetuadas medições.

7.1.1 Método de Teste

Para os testes, foi usado (de maneira análoga ao tópico 5 desse documento) o conceito de *cross validation*. Os dados capturados foram tirados do servidor e divididos em dois conjuntos: Um com 80% dos pontos e outro com 20%. O primeiro conjunto foi usado para treinar os modelos e o restante para validá-los. As imagens 19 e 20 contém os gráficos com a precisão do sistema e dos algoritmos individualmente para cada método de normalização usados na captura dos pontos no aplicativo *APScanner*.

Figura 19: Acerto de Classificação usando-se a Média Aritmética como método de normalização dos dados

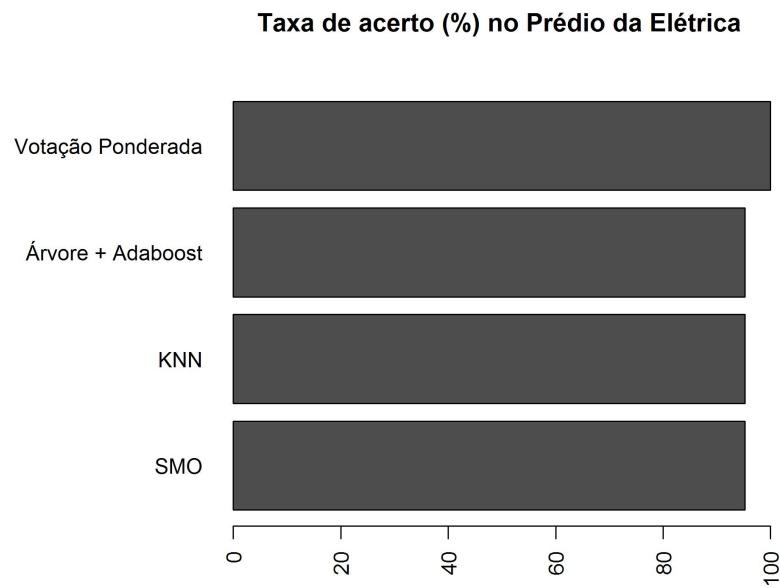
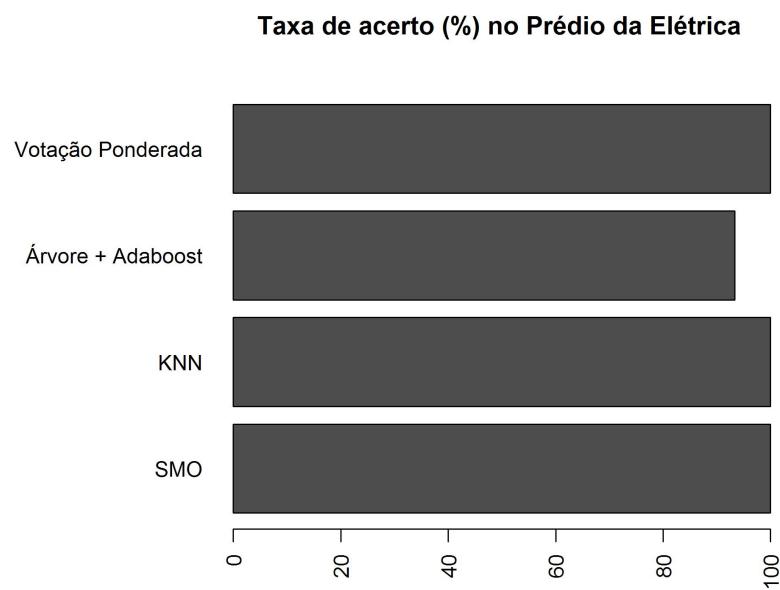


Figura 20: Acerto de Classificação usando-se o Filtro de Kalmann como método de normalização dos dados



Como é possível observar, tanto o uso da média aritmética quanto o filtro de Kalman como

método de normalização acabaram apresentando acerto máximo (100%) nos testes dentro da Poli-Elétrica.

7.2 Objetivos e Resultados

Tendo em vista o principal objetivo proposto para o projeto, que era desenvolver um conjunto de recursos que possibilitasse o mapeamento e identificação de áreas dentro de ambiente fechados, podemos dizer que o sistema desenvolvido conseguiu alcançá-lo parcialmente.

Nas fases iniciais do projeto, as primeiras concepções desenhadas previam a criação de um sistema muito semelhante ao GPS, só que para ambientes fechados. Ou seja, havia a ideia de que o sistema seria capaz de determinar a posição do usuário com a precisão de metros dentro de uma sala. Por exemplo, o sistema seria capaz de determinar se ele está mais perto da janela ou da porta da sala.

Porém, como foi constatado posteriormente durante a implementação do projeto, notou-se que existia um *trade-off* claro: Poderíamos abandonar a ideia de localizar o usuário com a precisão geométrica, para localizá-lo com mais precisão a respeito da zona que ele se encontra no prédio. E nesse escopo foram obtidos ótimos resultados.

Além do objetivo principal, durante o desenvolvimento do trabalho surgiu o objetivo da demonstração de uma prova de conceito, para o qual foram desenvolvidos os aplicativos *APScanner* e o *EletricaGO*. Os resultados obtidos a partir destes aplicativos foram usados para demonstrar o funcionamento das ferramentas desenvolvidas e uma das aplicações possíveis para o futuro.

O *APScanner* é importante para ilustrar o processo de aquisição dos sinais e seu envio para o servidor, com o posterior treinamento e a possibilidade imediata de testar o mapeamento feito. Por sua vez, o *EletricaGO* serve para apresentar uma aplicação contextual para o sistema, que usa as ferramentas desenvolvidas por meio da nossa API.

8 CONSIDERAÇÕES FINAIS

REFERÊNCIAS

- BOZKURT, S. et al. A comparative study on machine learning algorithms for indoor positioning. In: *Innovations in Intelligent SysTems and Applications (INISTA), 2015 International Symposium on*. [S.I.: s.n.], 2015. p. 1–8.
- BULTEN, W.; ROSSUM, A. C. V.; HASELAGER, W. F. G. Human slam, indoor localisation of devices and users. In: *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*. [S.I.: s.n.], 2016. p. 211–222.
- FREUND, Y.; SCHAPIRE, R. E. *A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting*. 1996.
- JAMES, G. et al. *An Introduction to Statistical Learning: With Applications in R*. [S.I.]: Springer Publishing Company, Incorporated, 2014. ISBN 1461471370, 9781461471370.
- MACLIN, R.; OPITZ, D. W. Popular ensemble methods: An empirical study. *CoRR*, abs/1106.0257, 2011. Disponível em: <<http://arxiv.org/abs/1106.0257>>.
- NAGI, S.; BHATTACHARYYA, D. K. Classification of microarray cancer data using ensemble approach. *Network Modeling Analysis in Health Informatics and Bioinformatics*, v. 2, n. 3, p. 159–173, 2013. ISSN 2192-6670. Disponível em: <<http://dx.doi.org/10.1007/s13721-013-0034-x>>.
- NG, A. Stanford cs229 - machine learning - ng. 2008.
- PIMENTA, R. D. da H. Pokémon go: imersão, publicidade e ludicidade em um novo modelo de compra e inserção de mídia. 2016.
- PROVOS, N.; MAZIERES, D. Bcrypt algorithm. In: USENIX. [S.I.], 1999.
- QUINLAN, R. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.
- SCHAPIRE, R. E. *Explaining AdaBoost*. 2013.
- TORRES-SOSPEDRA, J. et al. Ujiindoorloc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems. In: *Indoor Positioning and Indoor Navigation (IPIN), 2014 International Conference on*. [S.I.: s.n.], 2014. p. 261–270.

ANEXO A – TELAS DO APLICATIVO APSCANNER

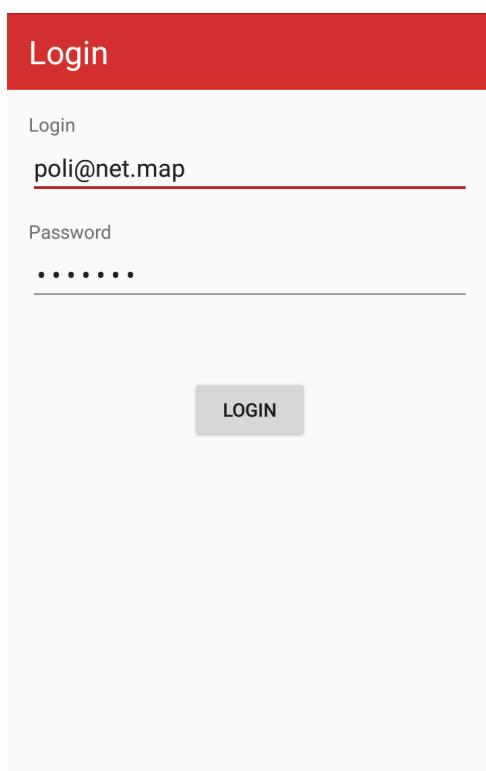


Figura 21: Tela para o usuário efetuar o login

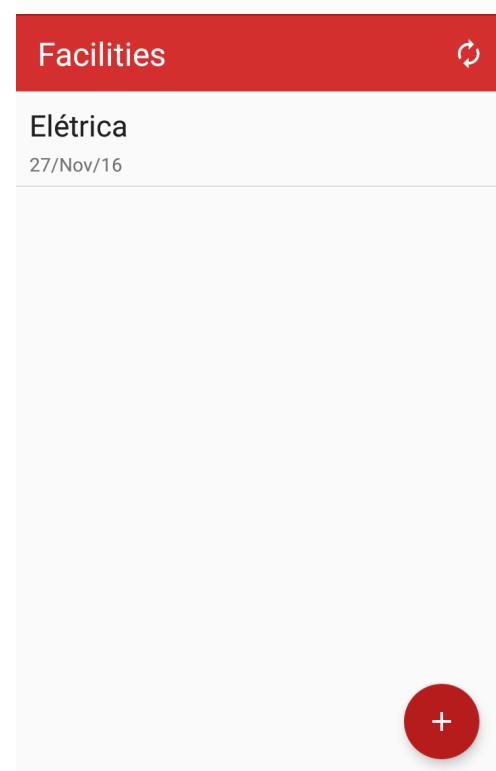


Figura 22: Tela informando as instalações do usuário

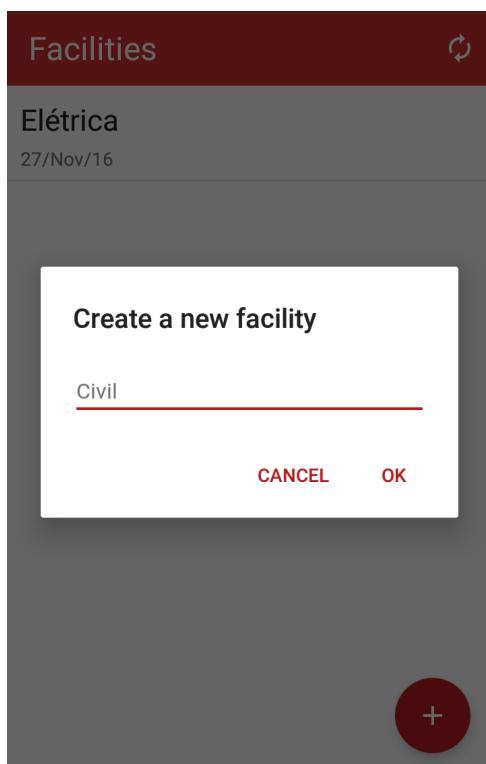


Figura 23: Tela para criar nova instalação

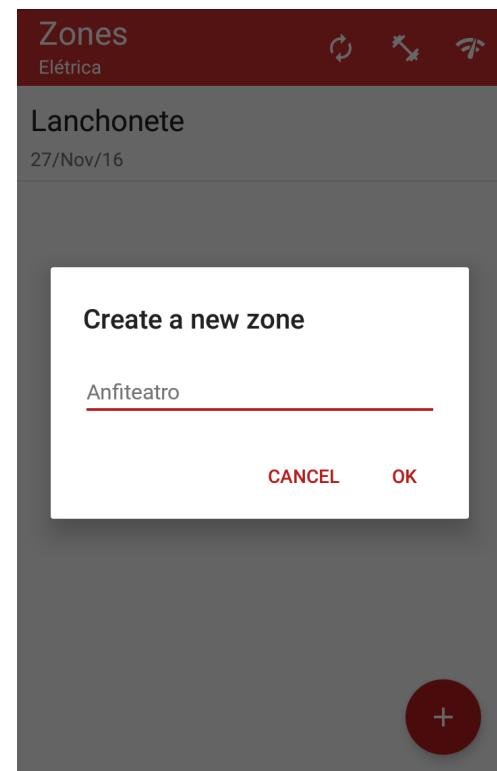


Figura 24: Tela para criar nova zona no sistema

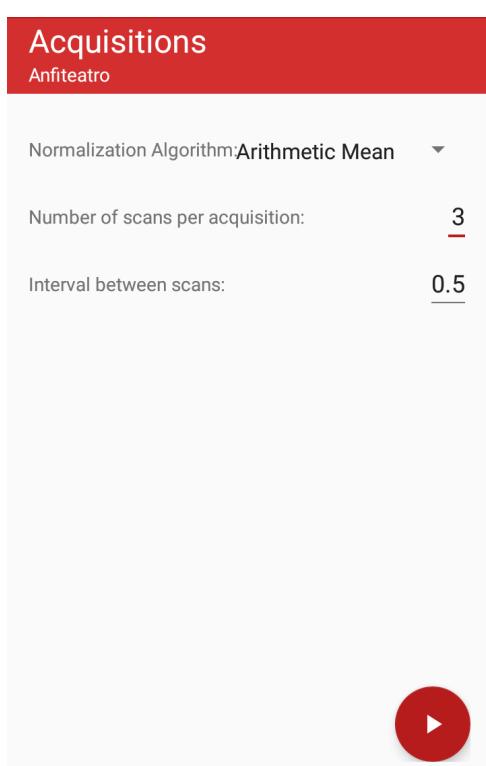


Figura 25: Tela para iniciar aquisição de sinais

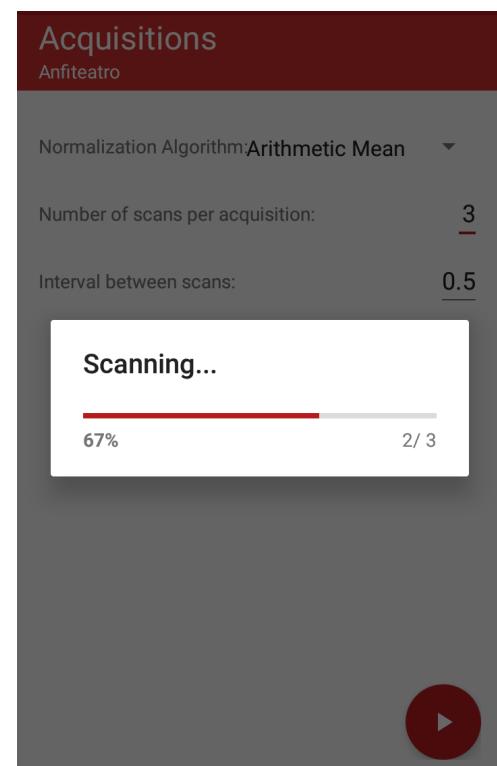


Figura 26: Tela do aplicativo adquirindo os sinais

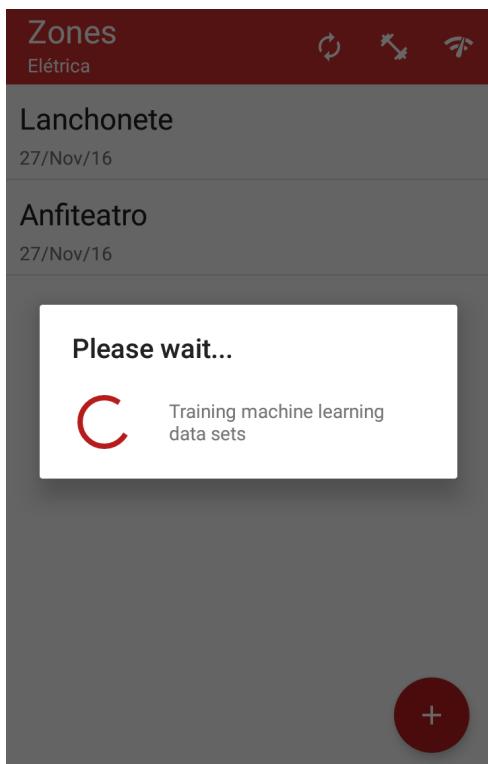


Figura 27: Tela informando que os dados estão sendo treinados

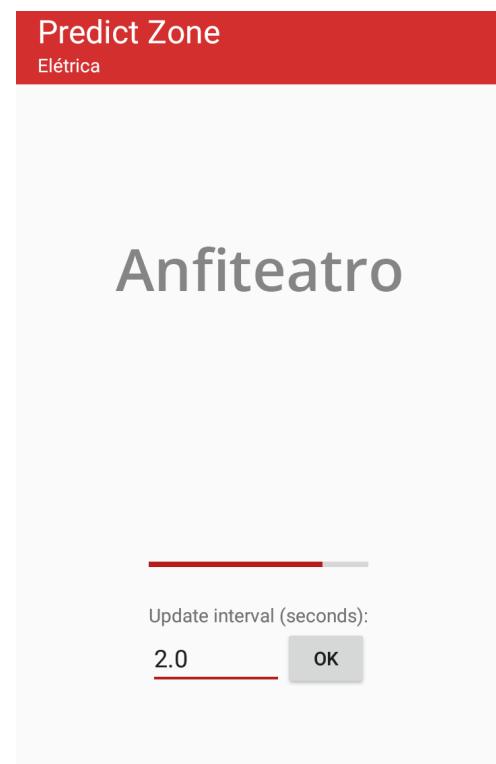


Figura 28: Tela do aplicativo informando a zona atual

ANEXO B – PLANTAS DO PRÉDIO DE ENGENHARIA ELÉTRICA DA POLI-USP COM MEDIÇÕES

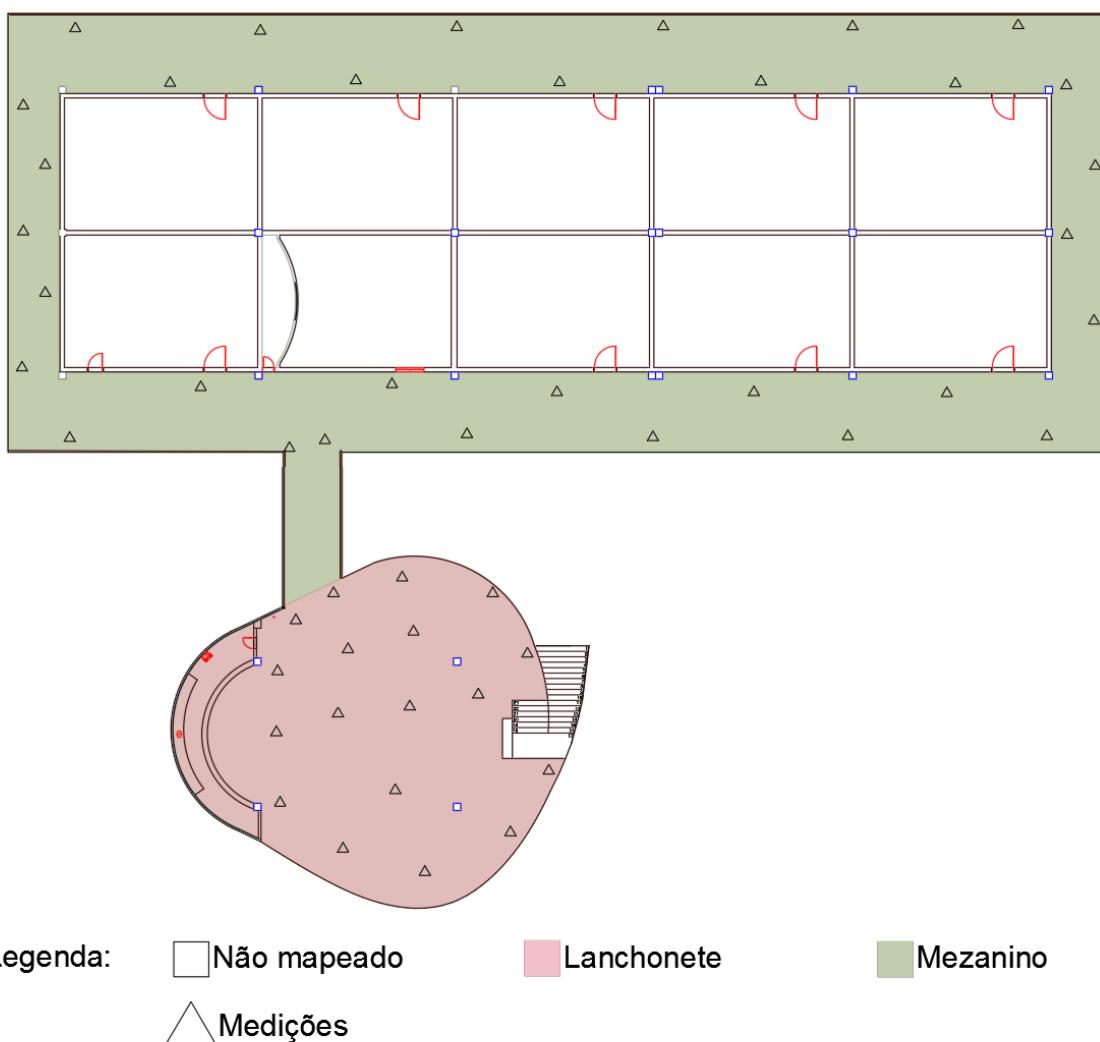


Figura 29: Representação do 1º andar do bloco B

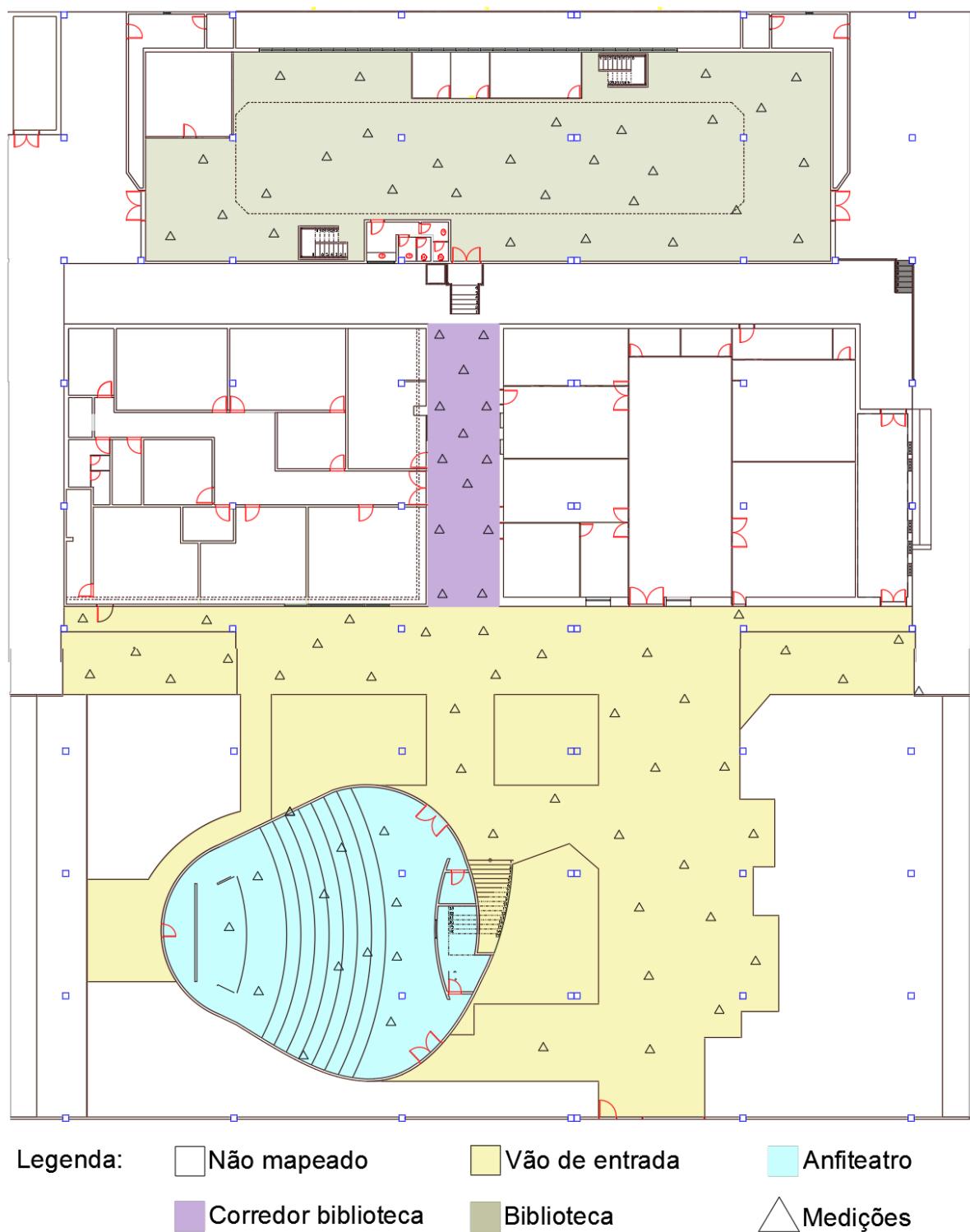


Figura 30: Representação do andar térreo do bloco B