ARYANE SAITO LIMA PARREIRA

DESENVOLVIMENTO DO APLICATIVO TREE LEARNING E ALGORITMO DE CLASSIFICAÇÃO PARA DETECÇÃO DE ÁRVORES COM FUNGOS

SANTO ANDRÉ - SP
2019

Aryane Saito Lima Parreira

DESENVOLVIMENTO DE APLICATIVO PARA DISPOSITIVOS MÓVEIS, ARMAZENAMENTO DE DADOS EM NUVEM E CLASSIFICAÇÃO DE DADOS APLICADOS AO PROBLEMA DE QUEDA DE ÁRVORES

Trabalho de Graduação apresentado ao curso de Engenharia da Informação da Universidade Federal do ABC como requisito parcial para obtenção do título de Bacharel em Engenharia da Informação

Professor Orientador: Prof. Dr. André Kazuo Takahata

SANTO ANDRÉ - SP
2019

SUMÁRIO

Resumo

Acompanhamos noticias sobre queda de árvores diariamente, e isso poderia ser evitado com mais informações sobre as árvores, já que algumas características podem indicar o risco de queda. Foi proposto então a criação de um aplicativo para recolher dados de árvores: localização, fotos, e algumas características do tronco e folhas. Esses dados são armazenados em tempo real em um banco de dados no Firebase, e ficam disponíveis em URL's públicas para serem utilizados para qualquer instituição de interesse. Para exemplificar o uso dessa base de dados criada, foi desenvolvido um algoritmo utilizando *K-Nearest Neighbor* (KNN) para classificar as fotos acerca da presença ou não de fungos. Foram coletados dados de cerca de 80 árvores e a esses dados se encontram disponíveis para qualquer futura análise. O algoritmo proposto para a classificação de árvores acerca da presença de fungos funcionou de acordo com o esperado e demonstrou uma das possibilidades de utilização dos dados colhidos para ajudar na predição do risco de queda de árvore, apesar do tipo de fungo mais frequente encontrado (líquen) não representar diretamente um risco de queda.

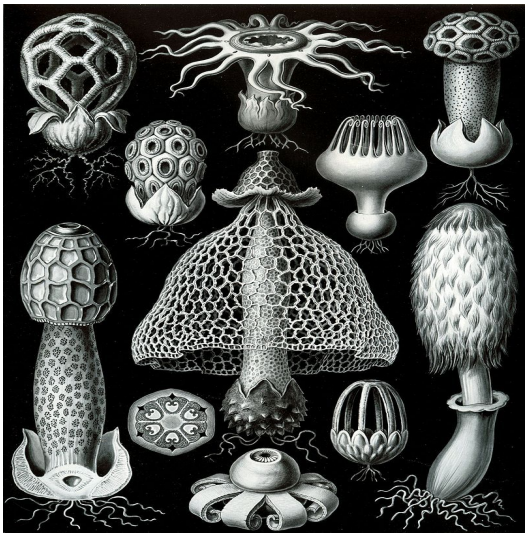
Palavras-chave: Queda de Árvores. Aplicativo. KNN. Base de dados. Fungos.

1. Introdução

A queda de árvores é um problema em zonas urbanas, e tem sido um assunto recorrente nos noticiários principalmente em épocas de alto índice pluviométrico. Nessas épocas podemos acompanhar diariamente muitas notícias sobre a queda de árvores e todas as consequências desse evento. Segundo a colunista Ananda Apple (2019), houve mais de 2 mil quedas de árvores em janeiro e fevereiro de 2019, só na cidade de São Paulo, eventos que afetam drasticamente a rotina urbana.

Dados os graves impactos desses eventos no ambiente urbano faz-se necessário analisar as características que culminam nesse evento extremo, a queda da árvore. Algumas características podem ser analisadas para predizer o risco de queda de uma árvore, por exemplo, as condições de crescimento das árvores, a biologia dos organismos xilófagos, que são os organismos que se alimentam de madeira, e as alterações nas propriedades anatômicas-físico-mecânicas do lenho sadio e deteriorado. Por exemplo, a presença das espécies de fungos das subdivisões Basidiomycotina e Ascomycotina estão associadas ao apodrecimento do cerne das árvores, sendo os basidiomicetos mais frequentes e que causam danos mais expressivos (BRAZOLIN, 2009).

Figura 1: Basidiomicetes



Fonte: Kunstformen der Naturde Ernst Haeckel, 1904

Grande parte dessas características que podem influenciar na queda são visuais e podem ser observadas por qualquer cidadão conforme afirma a SOCIEDADE INTERNACIONAL DE ARBORICULTURA, em cartilha (Figura 2)publicada em 2011. Algumas das características, defeitos ou sinais de possíveis defeitos nas árvores urbanas são os seguintes:

1. Rebrota oriunda de destopo, desobstrução de redes elétricas ou poda;

2. Rede elétrica adjacente à árvore;

3. Ramos partidos ou parcialmente presos à copa;

4. Cavidade aberta no tronco ou em galho;

5. Galhos mortos ou que estão morrendo;

6. Galhos que se originam a partir de um único ponto do tronco;

7. Lesões antigas apresentam fungos e apodrecimento;

8. Alteração recente no nível do solo ou na inclinação, ou outra alteração decorrente de construção;

Figura 2: Cartilha da Sociedade Internacional de Arboricultura

Fonte: Cartilha da Sociedade Internacional de Arboricultura, 2011.

Em posse dessas informações visuais sobre as árvores, técnicos podem julgar se árvore precisa de uma melhor análise ou se não apresenta risco. Hoje não existe uma base de dados unificada onde se encontram dados de árvores, o que dificulta o mapeamento do risco de queda.

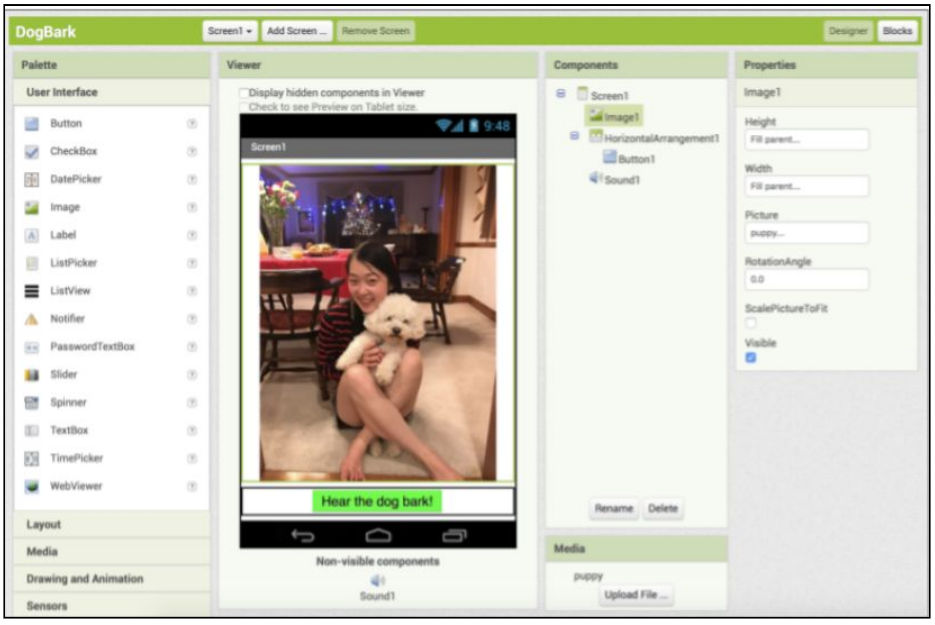
Com um aplicativo instalado no celular é fácil adquirir vários tipos de dados, inclusive acerca das árvores que fazem parte do caminho de uma pessoa.

1.1 App Inventor

O App Inventor é uma plataforma de software livre desenvolvida em 2009 pelo MIT(Massachusetts Institute of Technology), que permite o desenvolvimento de aplicativos para o sistema Android utilizando código baseado em blocos.(LAO, 2016)

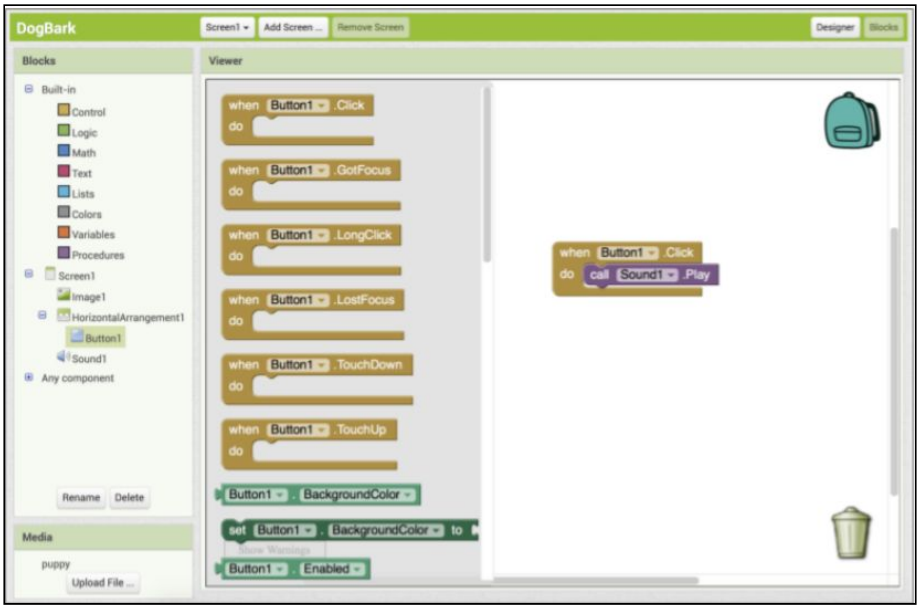
> O objetivo do MIT App Inventor tem sido democratizar a criação de aplicativos móveis, permitindo que pessoas com pouca ou nenhuma experiência em programação criem aplicativos funcionais.” (LAO, 2016)

Figura 3: app Inventor



Fonte: arquivo pessoal

Figura 4: app Inventor

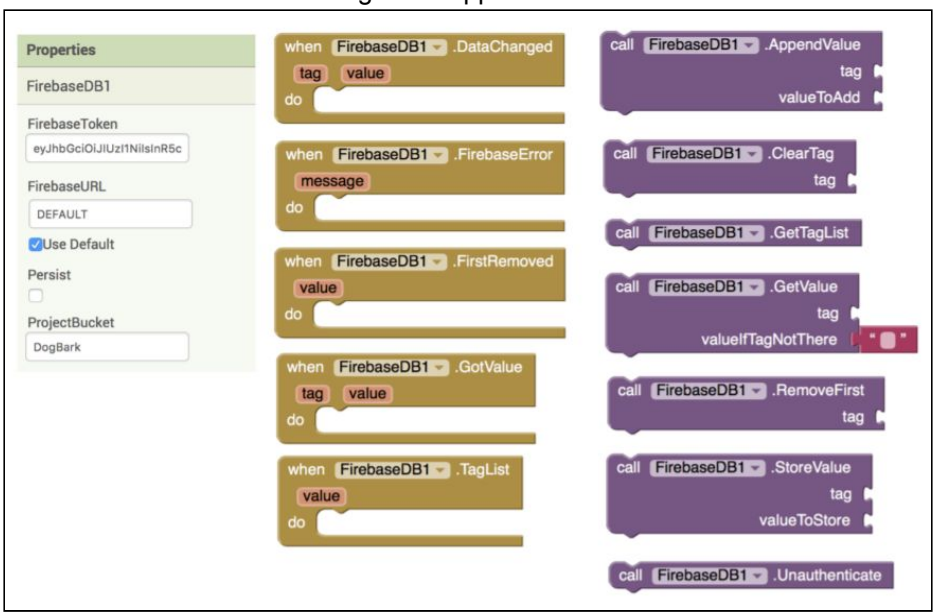


Fonte: arquivo pessoal

Um projeto do MIT App Inventor como mostrado na Figura 3 e 4 é construído usando uma interface de designer e uma interface de blocos. A interface do designer (Figura 3) permite aos usuários arrastar, soltar e personalizar componentes como botões em uma tela do Android. A interface (Figura 4) fornece blocos de código que podem ser organizados para formar programas básicos.Neste exemplo, um arquivo de som é tocado com o botão clicado.

Essa plataforma também permite a integração com banco de dados em nuvem (FirebaseDB), e demonstra que qualquer pessoa pode entender e usar a nuvem para construir algo de relevância para si mesmos e suas comunidades. (LAO, 2016)

Figura 5: app Inventor



Fonte: arquivo pessoal

O componente FirebaseDB atualmente disponível no MIT App Inventor tem quatro propriedades definidas (à esquerda) e 12 blocos para programação, vide Figura 4, onde podemos ver que 5 blocos da esquerda verificam um evento (um erro, um novo dado, etc) e os 7 blocos da direita sao blocos de que permitem executar alguma ação no Firebase DB.

Tendo isso em vista, o MIT App Inventor se torna uma alternativa para desenvolver um aplicativo para coleta de dados aplicado a um problema da comunidade, pela sua facilidade de desenvolvimento e integração com banco de dados na nuvem. Ou seja, um cidadão com conhecimento mínimo de lógica de programação e interessado em obter dados acerca de um problema da comunidade pode desenvolver um aplicativo para tal finalidade e ter todos os dados coletados armazenados em um banco de dados de nuvem.

Para o problema de queda de árvores, é possível então desenvolver o aplicativo pela plataforma do App Inventor, armazenar os dados colhidos na nuvem e deixá-los disponíveis para qualquer tipo de análise de dados.

1.2 Reconhecimento de padrões e Classificação

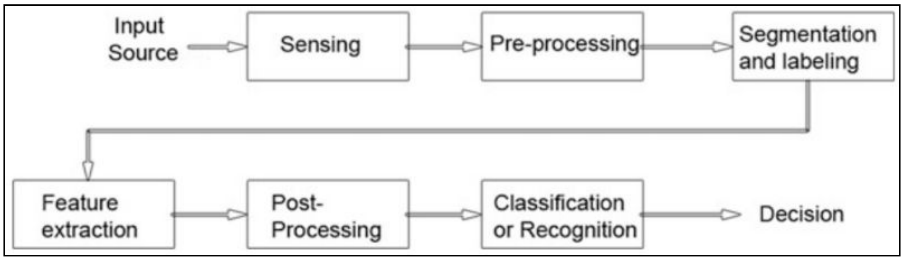
Nós humanos temos a capacidade de reconhecer coisas e padrões com certa facilidade, sem perceber quais são os passos por trás dessa atividade, ela é feita inconscientemente. Pode ser difícil imaginar como um computador poderia ter tal capacidade, mas isso é possível através de inúmeros algoritmos já desenvolvidos. (DOUGHERTY, 2013)

Por exemplo, é muito fácil conseguimos identificar uma maçã e distingui-la de uma laranja, mas podemos utilizar algoritmos de reconhecimento de padrões para que o computador também seja capaz de identificar uma maçã.

Um processo de classificação funciona com os seguintes passos:

- Aquisição: Coleta de dados, tirar uma foto, por exemplo.
- Pré-processamento: Preparar o dado (foto) para extrair somente as informações interessantes do objeto, então pode exemplo neste passo podem ser usadas técnicas para diminuir o ruido da foto.
- Segmentação e rotulagem: A imagem é separada para serem extraídas as características de interesse, por exemplo um objeto é destacado do background.
- Extração de características: extrair uma característica que pode ser comum entre os objetos da mesma classe.
- Pós-processamento: Pode ser usado para prepará-lo para a extração de recursos.
- Classificação e Reconhecimento: Aplicação do algoritmo escolhido.
- Decisão: Resultado

Figura 6: processo de classificação



Fonte. Processo de classificação (DOUGHERTY, 2013)

Podemos aplicar então algoritmos de Reconhecimento de padrões para tomar decisões sobre alguma característica relevante dos dados colhidos, e classificar as fotos considerando a presença ou a ausência de fungos.

2. Objetivo

O objeto geral deste trabalho é colher dados de árvores e criar uma base de dados pública com informações relevantes que poderiam ajudar na identificação do risco de queda. E fazer uma análise nos dados colhidos para demonstrar uma das possíveis aplicações.

2.1. Objetivos Específicos

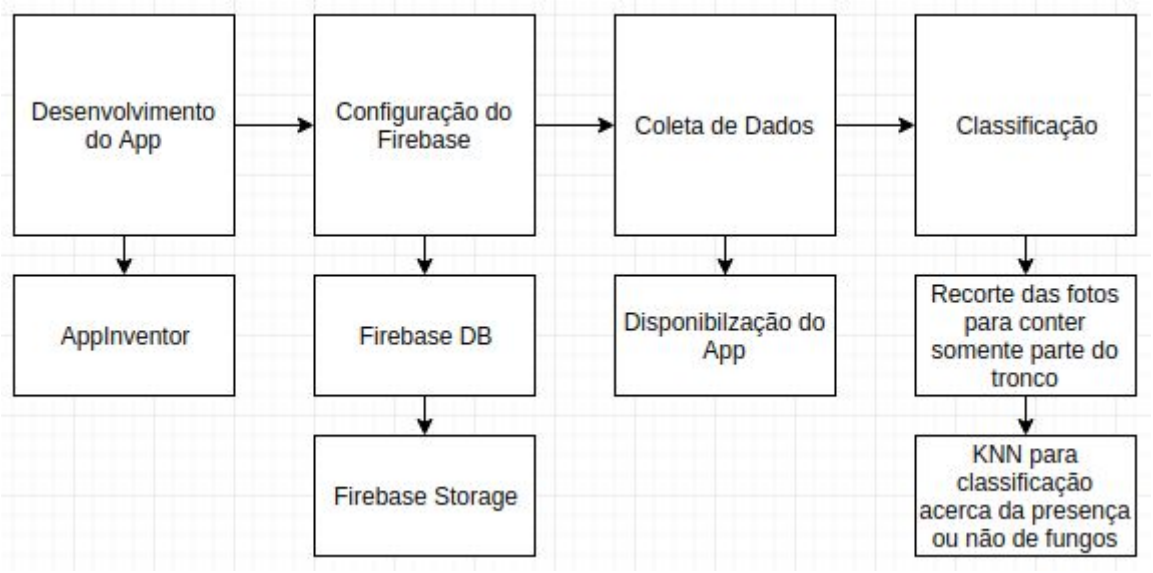
- Desenvolver aplicativo para dispositivos móveis
- Criar base de dados pública
- Colher dados de algumas árvores
- Desenvolver algoritmo para identificar árvores com fungos

3. Procedimento

Para atender os objetivos traçados foram escolhidas algumas tecnologias para desenvolver o aplicativo, criar o banco de dados e classificá-los. (Figura 7)

O procedimento pode ser exemplificado conforme Figura 7, e os passos serão detalhados a partir do item 3.1.

Figura 7: Fluxograma do Procedimento



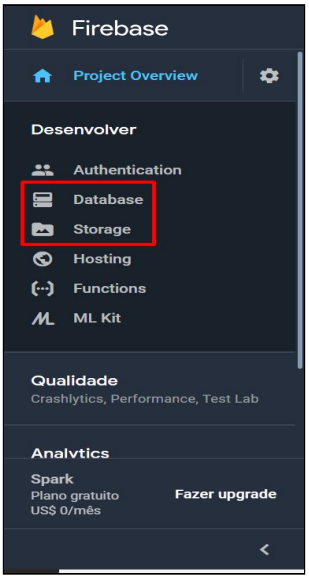
Fonte: arquivo pessoal

3.1 Desenvolvimento do App

3.1.1 Firebase

Foi criada uma conta gratuita no firebase, para utilizarmos as funções de Banco de Dados e Armazenamento (Storage), conforme figura 8.

Figura 8: console do projeto no Firebase



Fonte: arquivo pessoal

3.1.2 App Inventor

Criamos um aplicativo para funcionar como um formulário, onde temos uma lista de características da árvores a serem selecionadas, localização atual, possibilidade de enviar duas fotos e o botão de enviar, conforme mostrado na Figura 9.

Figura 9: interface de Usuário do App.

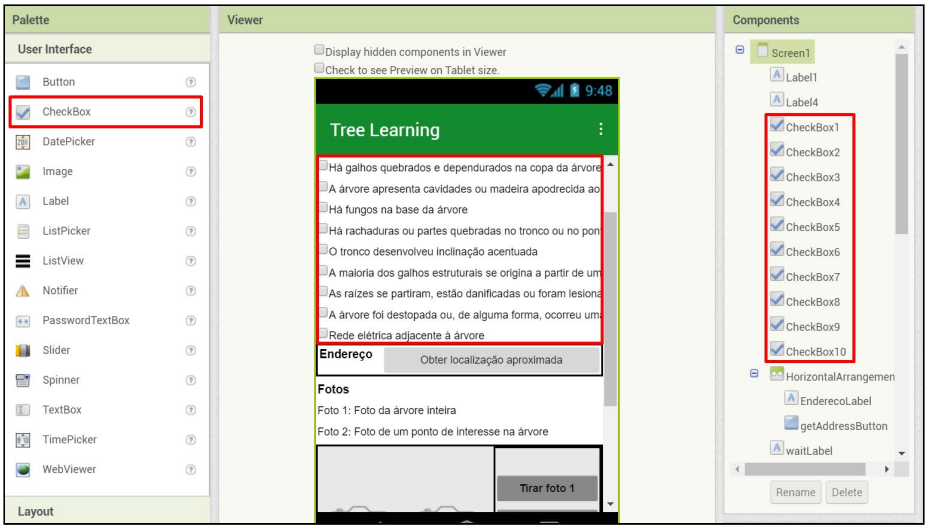
Fonte: arquivo pessoal

O usuário que observa a árvore deve marcar as características presentes, inserir a localização clicando no botão correspondente, tirar duas fotos conforme a descrição e após verificação dos dados deve-se clicar no botão Enviar.

As características das árvores foram baseadas na cartilha de Reconhecimento de riscos de árvores (SOCIEDADE INTERNACIONAL DE ARBORICULTURA, 2011), mostrado na Figura 2, possibilitando que qualquer cidadão consiga identificar essas características, observando a árvore por apenas alguns instantes.

Para a primeira parte do formulário foi utilizado o elemento checkbox, onde o usuário pode selecionar se a característica do texto ao lado está presente na árvore observada, como podemos ver na figura 10.

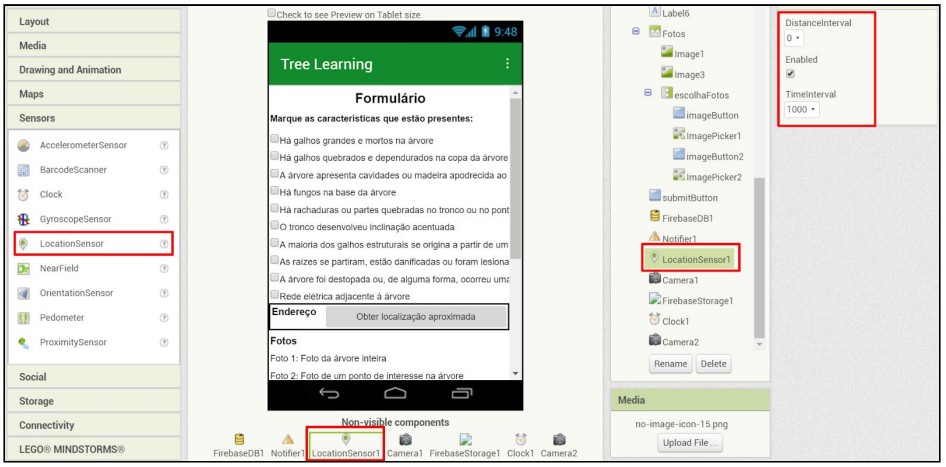
Figura 10: Primeira parte do formulário utilizando o elemento CheckBox.



Fonte: arquivo pessoal

Para obter o endereço adicionamos o elemento de Button, para acionar o GPS, mas para ter acesso ao sensor de GPS precisamos adicionar o elemento Location Sensor, um elemento não-visível, conforme figura 11.

Figura 11: Elemento LocationSensor para acessar o GPS e obter a localização.



Fonte: arquivo pessoal

Para as fotos, utilizamos os elementos Button para o botão de tirar foto e escolher foto da galeria, e o elemento Image para exibir as imagens.

Também adicionamos o elemento Camera, que é um elemento não-visível que permite o App acessar o sensor de Câmera e Galeria.

Como vamos armazenar os dados no Firebase precisamos adicionar o elemento não-visível FirebaseDB, para fazermos essa integração.

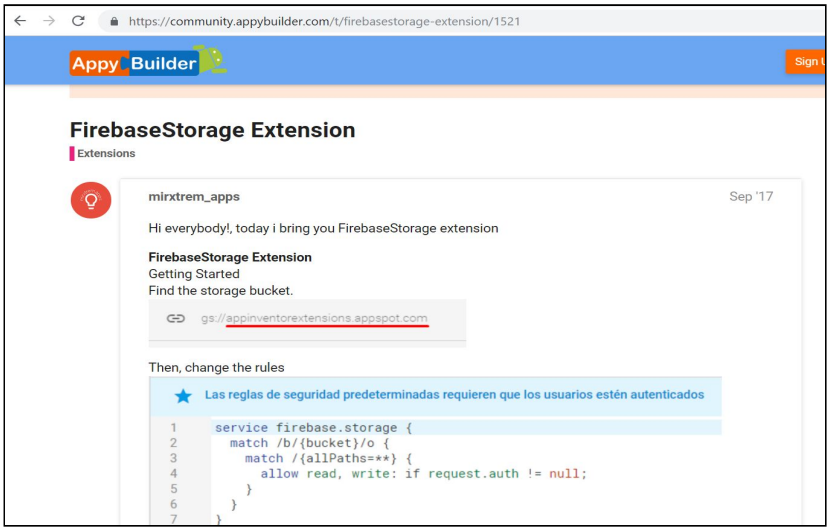
Figura 12. Propriedades do elemento FirebaseDB

Fonte: arquivo pessoal

Como mostrado na Figura 12, as informações de propriedades do elemento foram preenchidas conforme as informações do projeto criado no Firebase. FirebaseURL é a URL do banco de dados do Firebase, FirebaseToken é a chave de acesso ao banco para escrita e leitura de dados, e a propriedade ProjectBucket é o nome dado ao banco de dados no Firebase.

Não é possível armazenar imagens nesse banco de dados, então utilizamos outro recurso do Firebase, o Firebase Storage. Para isso precisamos adicionar uma extensão chamada Firebase Storage, que pode ser baixada no site da comunidade da plataforma AppyBuilder, que é uma plataforma baseada no App Inventor, onde foram desenvolvidas algumas funções além do que o App Inventor suporta. (APPYBUILDER, 2018)

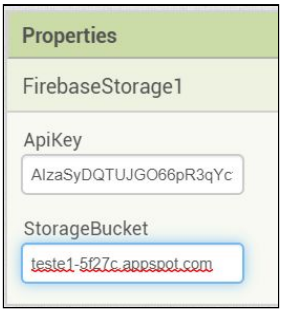
Figura 13: Site da comunidade AppyBuilder onde está disponível a extensão Firebase Storage.



Fonte: (APPYBUILDER,2018)

As propriedades desse elemento também foram preenchidas conforme as informações do projeto criado no Firebase (Figura 14). Onde ApiKey é a chave de acesso da Interface de Programação de Aplicativos, onde é possível a comunicação com o Firebase storage. E a propriedade StorageBucket é a URL da Interface de Programação de Aplicativos.

Figura 14. Propriedades do elemento FirebaseStorage

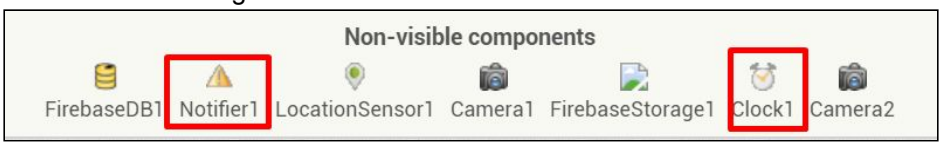


Fonte: arquivo pessoal

Além dos elementos já citados também foram adicionados os elementos Clock para obter informações do relógio do dispositivo, assim é possível determinar a hora exata que o usuário submete o formulário, e também utilizamos essa informação para nomear os arquivos de fotos e conseguir relacioná-los com o banco de dados da resposta. E o elemento Notifier (Figura 15) que permite acesso às

notificações para dar avisos ao usuário quando o formulário é enviado ou quando ocorre algum erro.

Figura 15. Elementos não-visíveis utilizados

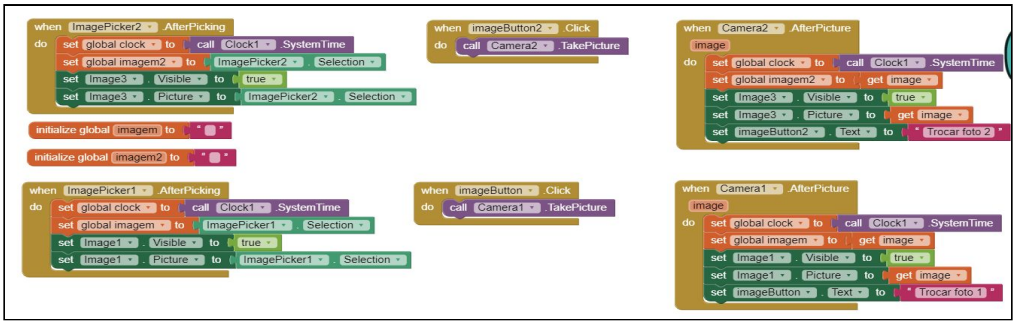


Fonte: arquivo pessoal

Com todos os elementos adicionados, utilizando a interface de blocos fizemos toda a parte lógica do app.

Para as fotos, quando um dos botões é clicado (Tirar/Escolher foto), é iniciada a câmera ou a galeria, respectivamente. Após a seleção da foto, a imagem que aparece na interface do app é trocada para a foto escolhida. Durante esse passo, também é armazenado o timestamp (Clock1.SystemTime) que será o nome do arquivo de imagem a ser enviado ao Firebase, e a tag do banco de dados, dessa forma conseguimos relacionar a foto com as informações enviadas no formulário (Figura 16).

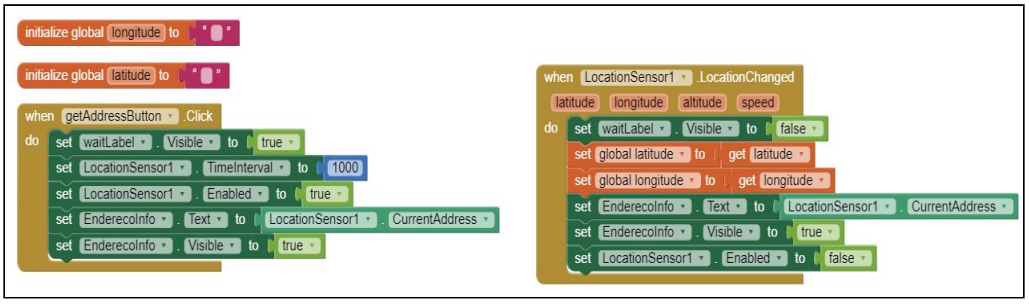
Figura 16: lógica para selecionar foto da galeria ou tirar uma foto.



Fonte: arquivo pessoal

Para obter a localização, o GPS é ativado, e quando se obtém a primeira leitura, são armazenadas as informações de latitude e longitude e é exibido na tela o endereço (Figura 17).

Figura 17. Programação dos blocos para obter localização

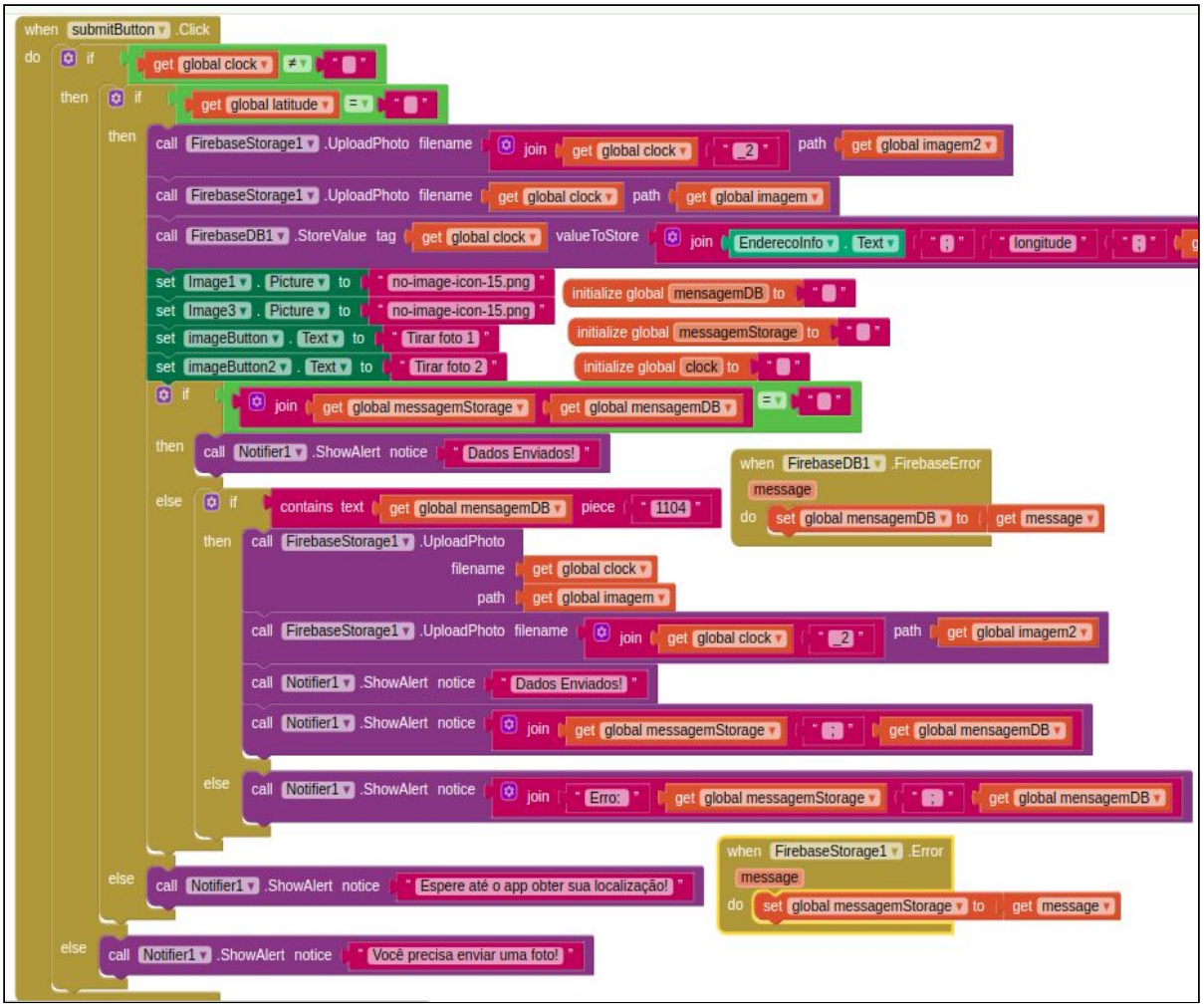


Fonte: arquivo pessoal

Conforme Figura 18, para envio do formulário, primeiramente é feita a validação das variáveis clock e latitude, pois se elas são diferentes de “ ”(vazio) significa que pelo menos uma foto foi tirada e que foi identificada a localização, respectivamente. Caso contrário, é exibida uma notificação ao usuário informando qual campo está inválido.

É feita então uma tentativa de enviar os dados do formulário e as fotos, caso falhe, o app tenta novamente enviar as informações, se houver sucesso em alguma das tentativas uma mensagem de “Dados Enviados” é exibida na tela, senão uma mensagem de erro é exibida. Após o envio do formulário com sucesso, as fotos escolhidas são removidas da tela, para evitar que o usuário envie o mesmo formulário duas vezes.

Figura 18. Programação dos blocos para envio das informações para o Firebase.



Fonte: arquivo pessoal

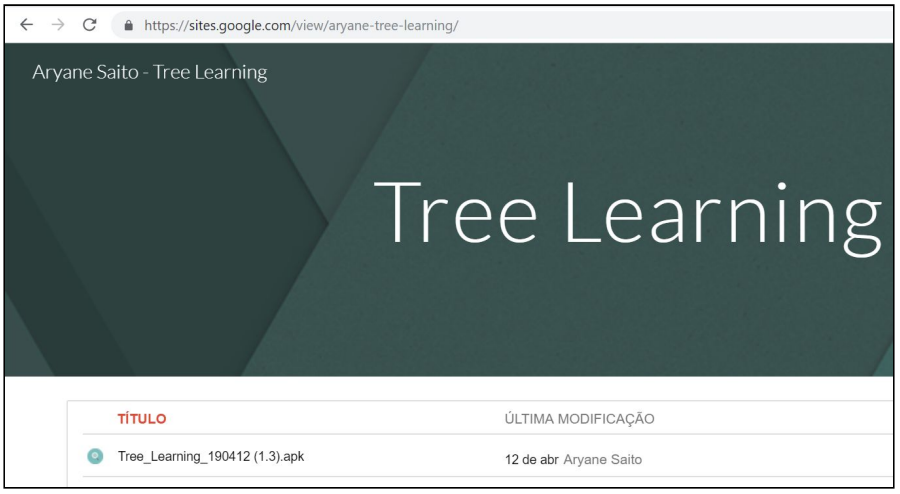
Os dados dos elementos de *checkbox* são enviados como texto, utilizando ";" como separador, e o *timestamp* como *tag* do banco de dados, onde será possível identificar a foto que será nomeada com essa *tag*.

3.1.3 Disponibilização do App

Uma versão estável do App foi disponibilizada em um site na internet: https://sites.google.com/view/aryane-tree-learning/ (Figura 19)

Assim os usuários puderam baixar e instalar o app no seu smartphone.

Figura 19. Site para disponibilização do App



Fonte: arquivo pessoal

3.2 Coleta de Dados

O app foi instalado por 6 pessoas, que foram instruídas a observar as árvores que fazem parte do seu cotidiano ou de lugares de passeio, principalmente aquelas que apresentavam presença de algum tipo de fungo. A coleta de dados foi realizada do dia 12 de abril de 2019 ao dia 19 de abril de 2019.

O formulário continha as seguintes informações, onde o usuário deveria selecionar as que estavam presentes na árvore observada:

- 10 frases do tipo Verdadeiro ou Falso, se o usuário marcou a característica é enviada a informação verdadeiro (true), e não marcou é enviado falso (false), retiradas da cartilha da SOCIEDADE INTERNACIONAL DE ARBORICULTURA:
    - Há galhos grandes e mortos na árvore;
    - Há galhos quebrados e dependurados na copa da árvore;
    - A árvore apresenta cavidades ou madeira apodrecida ao longo do tronco;
    - Há fungos na base da árvore nos galhos maiores;
    - Há rachaduras ou partes quebradas no tronco ou no ponto de inserção de galhos;
    - O tronco desenvolveu inclinação acentuada;

    - A maioria dos galhos estruturais se origina a partir de um único ponto do tronco;
    - As raízes se partiram, estão danificadas ou foram lesionadas por alteração do nível do solo, por pavimentação, por reparos nas calçadas ou pela escavação de valas;
    - A árvore foi destopada ou, de alguma forma, ocorreu uma poda intensa;
    - Rede elétrica adjacente à árvore;
- Depois o usuário deveria informar a localização, clicando no botão correspondente, e seriam extraídos:
    - Endereço;
    - Latitude;
    - Longitude;
- E por último o usuário deveria tirar 2 fotos:
    - Foto 1: uma foto da árvore inteira
    - Foto 2: foto de um ponto de interesse na árvore (por exemplo, parte do tronco com fungo)

3.3 Algoritmo de Classificação

Foi decidido analisar essas fotos acerca da presença ou não de fungos, que podem influenciar o risco de queda.

Para o pré-processamento e rotulagem das fotos, as imagens foram recortadas para que contivesse apenas um pedaço do tronco onde poderia ser verificado a presença ou não de fungos (Figura 20). A segmentação da imagem para o reconhecimento da posição da árvore na foto ficou fora do escopo deste estudo.

Figura 20. Imagem Original → Imagem processada/rotulada

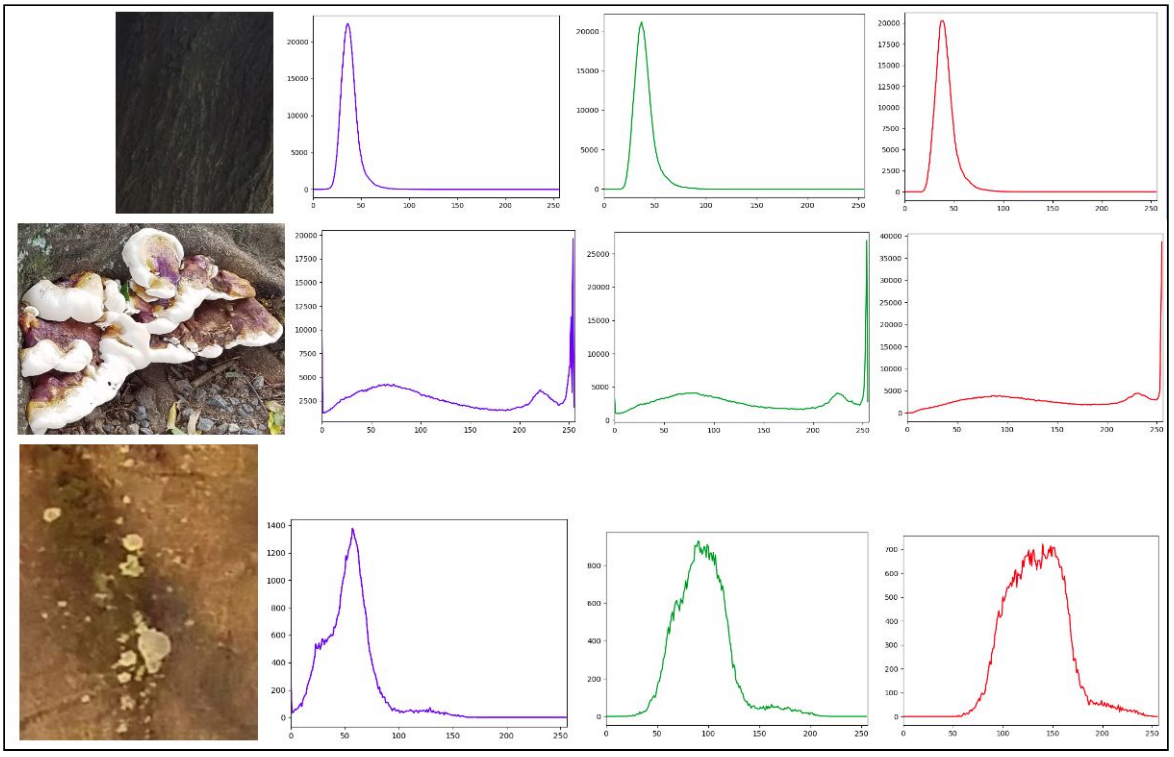
Fonte: arquivo pessoal

Para o processamento de imagens e aplicação do algoritmo de classificação, foi utilizada a OpenCV (Open Source Computer Vision Library) que é uma biblioteca de software de visão computacional e aprendizado de máquina de código aberto, (OPENCV TEAM, 2019).

Para analisar as imagens a característica escolhida foi o histograma, que é determinado pela função discreta $h(r_k)=n_k$ onde r_k é a k-ésimo valor de intensidade e n_k é o número de pixels na imagem com a intensidade $r_{k,}$, onde $r_{k,}$ é inteiro de 0 a 255. (GONZALES e WOODS, 2006). Utilizamos a imagem em um espectro RGB, onde cada pixel é um vetor com 3 valores (vermelho, verde e azul), extraímos o histograma de cada canal e concatenamos em um vetor os 3 histogramas, dessa forma obtemos 256x3 *features*, ou 768 *features*.

Podemos observar alguns exemplos de histogramas obtidos na Figura 21, onde o eixo Y do histograma apresenta a frequência dos *pixels* mostrados do eixo X (de 0 a 255).

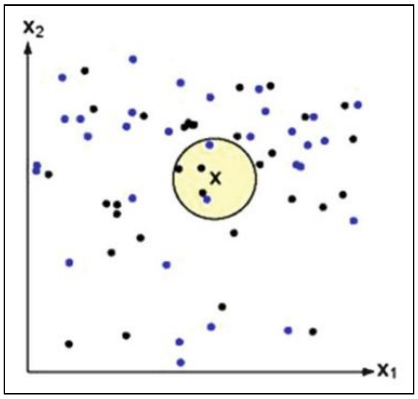
Figura 21: Exemplo de histograma dos canais RGB em uma árvore sem fungo e duas com fungo



Fonte: arquivo pessoal

O classificador escolhido foi o *K-Nearest Neighbor* (KNN), esse classificador exemplifica o seguinte ditado: “Se anda como um pato, grasna como um pato, e parece um pato, então provavelmente é um pato” (DOUGHERTY, 2013), isto é, a classe de uma instância de teste é determinada pela classe de seus vizinhos mais próximos, como exemplificado na Figura 22.

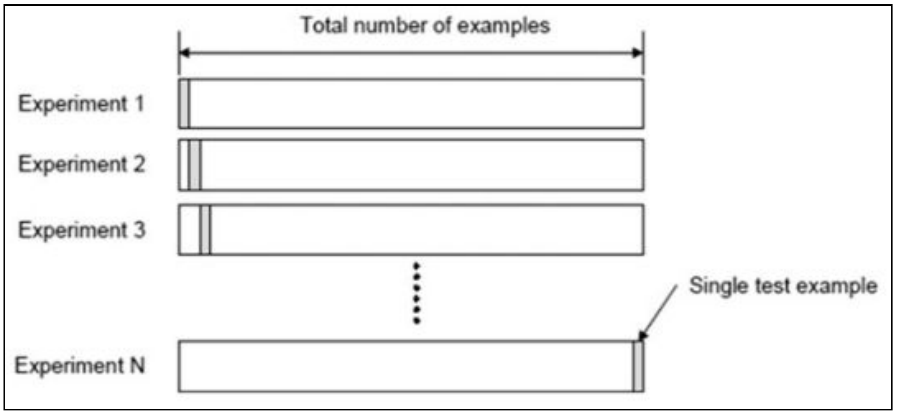
Figura 22. Caso *k*=5, a classe de x será determinada pela classe dos pontos pretos



Fonte: Dougherty, 2013.

O tipo de distância escolhida foi a distância euclidiana e para treinar e validar o algoritmo foi utilizada a técnica de validação cruzada em *n* compartimentos, que é a técnica mais simples de validação cruzada, onde os dados são separados em dois grupos, um para treinamento e outro para validação. No caso, utilizamos ainda um caso particular chamado *leave-one-out,* onde utilizamos o maior número de dados possíveis para treinamento (*N*-1), utilizamos apenas uma amostra para a validação (Figura 23).

Figura 23: validação cruzada: Leave-one-out

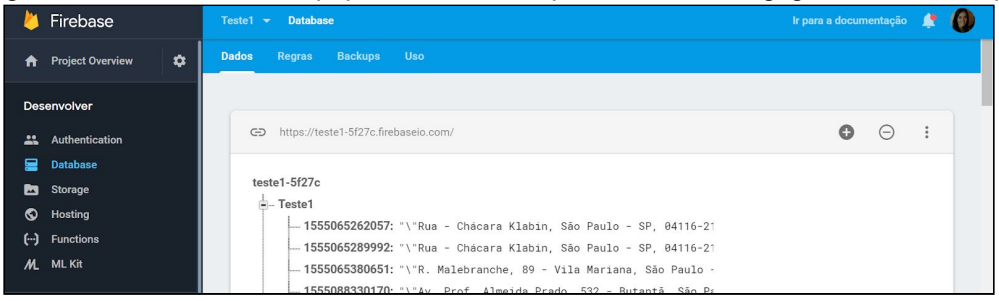


Fonte: Dougherty, 2013

4. Resultados

Como mencionado na Subseção 3.2, seis usuários instalaram o aplicativo e foram instruídos a coletar dados de árvores que faziam parte de seus caminhos diários ou passeios de modo a verificar todos os pontos da lista do formulário e obter mais duas fotografias, uma da árvore inteira e outra de um ponto de interesse. Em cerca de uma semana foram coletados dados de cerca de 80 árvores.

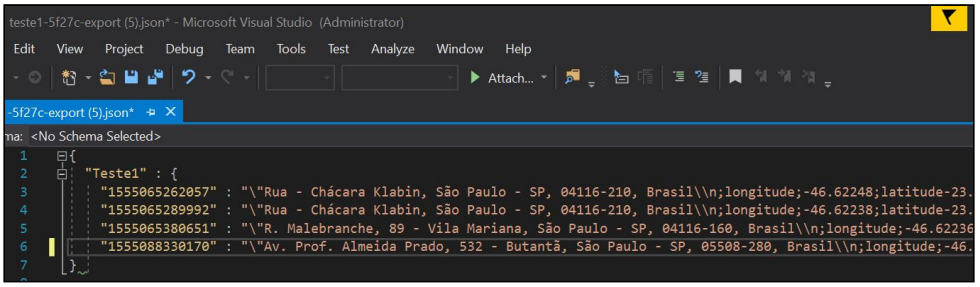
Figura 24. Banco de Dados populado conforme previsto, com a tag igual ao timestamp.



Fonte: arquivo pessoal.

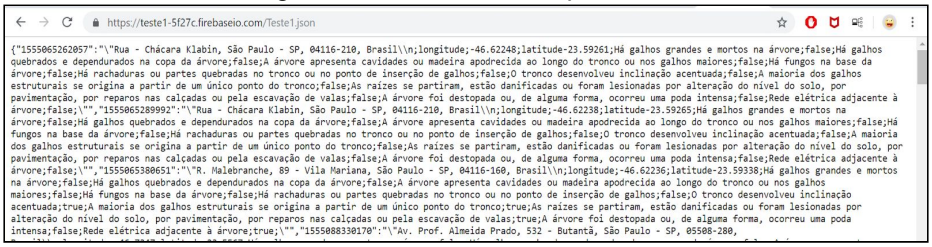
Podemos observar os dados coletados já disponiveis no Firebase, conforme Figura 24. Os dados do banco do Firebase podem ser exportados em formato JSON (JavaScript Object Notation), que é um formato leve de intercâmbio de dados, fácil para humanos lerem e escreverem e fácil para as máquinas analisarem e gerarem (CROCKFORD, 2018). Os dado podem ser acessados também neste formato através de uma URL pública do firebase: https://teste1-5f27c.firebaseio.com/Teste1.json, possibilitando qualquer pessoa ter acesso a essas informações. Podemos obervar a informação de duas formas: exportando o arquivo, conforme Figura 25, ou somente pela URL, conforme Figura 26.

Figura 25. Json exportado pelo Firebase



Fonte: arquivo pessoal

Figura 26. Json acessado pela URL

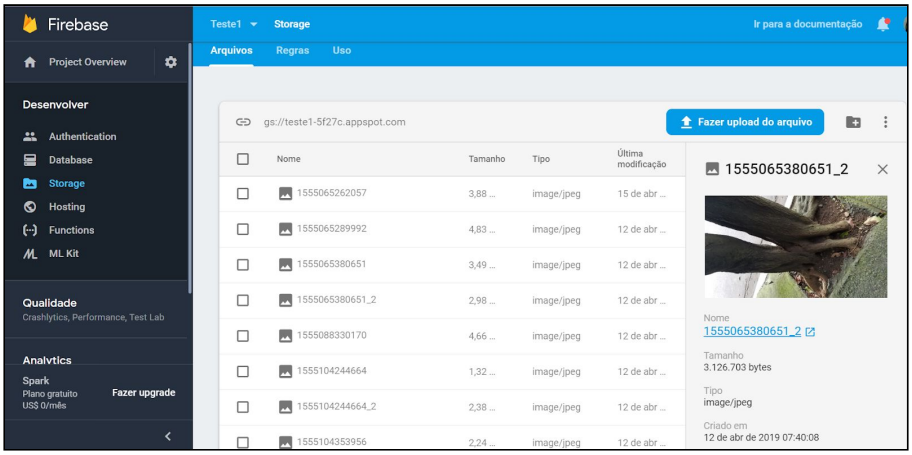


Fonte: arquivo pessoal

Exemplo de dados de uma árvore: "1555065262057": "\"Rua - Chácara Klabin, São Paulo - SP, 04116-210, Brasil\\n;longitude;-46.62248;latitude-23.59261;Há galhos grandes e mortos na árvore;false;Há galhos quebrados e dependurados na copa da árvore;false;A árvore apresenta cavidades ou madeira apodrecida ao longo do tronco ou nos galhos maiores;false;Há fungos na base da árvore;false;Há rachaduras ou partes quebradas no tronco ou no ponto de inserção de galhos;false;O tronco desenvolveu inclinação acentuada;false;A maioria dos galhos estruturais se origina a partir de um único ponto do tronco;false;As raízes se partiram, estão danificadas ou foram lesionadas por alteração do nível do solo, por pavimentação, por reparos nas calçadas ou pela escavação de valas;false;A árvore foi destopada ou, de alguma forma, ocorreu uma poda intensa;false;Rede elétrica adjacente à árvore;false;\""

As fotografias ficam disponíveis no Firebase Storage, com o nome do arquivo igual a tag do banco de dados (timestamp) para a foto 1, e o nome da foto 2 tem o sufixo “_2” no nome do arquivo. (Figura 27)

Figura 27. Tela do Firebase storage, onde as fotos estão armazenadas.

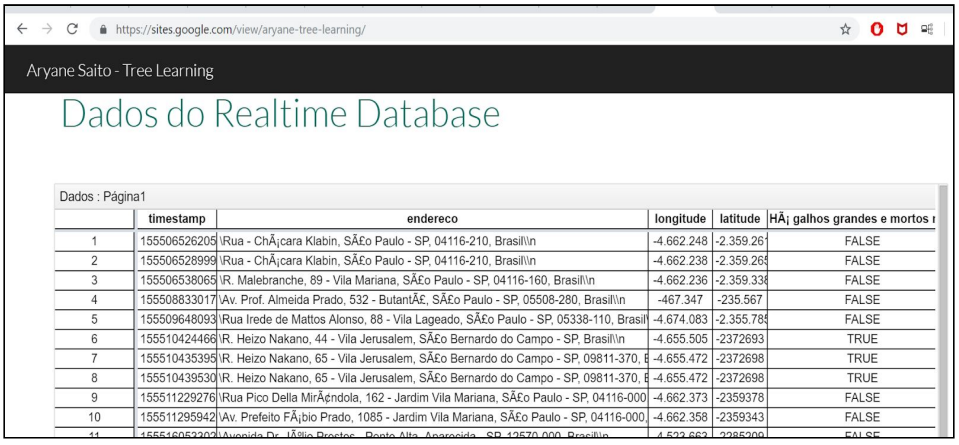


Fonte: arquivo pessoal

As fotografias também podem ser acessadas por uma url pública, no formato https://firebasestorage.googleapis.com/v0/b/teste1-5f27c.appspot.com/o/1555065262057?alt=media&token=a030f9c2-b08b-417e-9fbe-c41c6f699a06, onde em vermelho é o nome do arquivo a ser acessado, que pode ser encontrado no campo timestamp das respostas do formulário, que estão disponiveis conforme Figura 26.

Foi feita então uma planilha no Microsoft Office Excel online, estruturando todos os dados, inclusive a url das fotos, e foi disponibilizada no site do projeto onde também está hospedado o arquivo .apk para instalação do App. (Figura 28)

Figura 28. Planilha com todos os dados, disponivel no site do projeto.

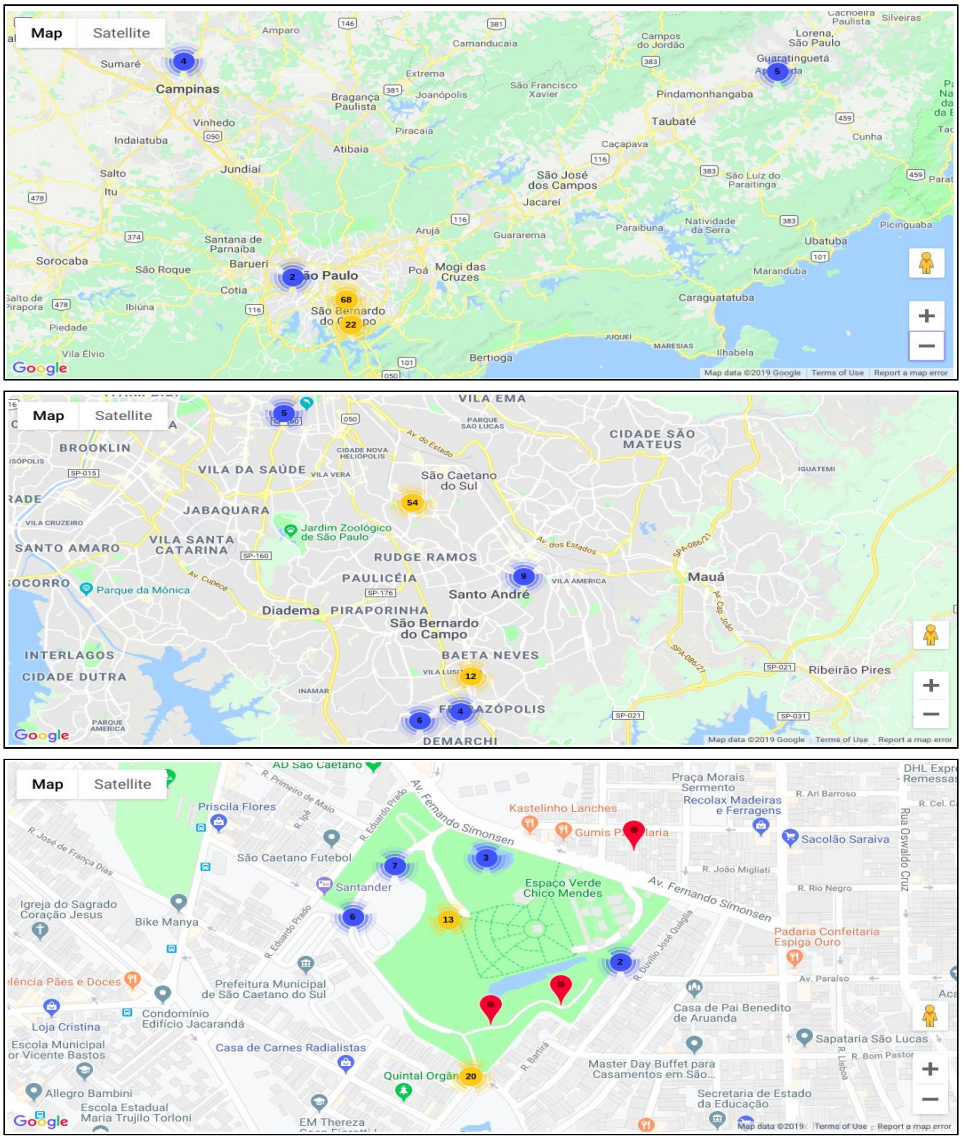


	timestamp	endereco	longitude	latitude	HÃ¡ galhos grandes e mortos
1	1555065262057	\Rua - ChÃ¡cara Klabin, SÃ£o Paulo - SP, 04116-210, Brasil\\n	-4.662.248	-2.359.26	FALSE
2	1555065289992	\Rua - ChÃ¡cara Klabin, SÃ£o Paulo - SP, 04116-210, Brasil\\n	-4.662.238	-2.359.265	FALSE
3	1555065380651	\R. Malebranche, 89 - Vila Mariana, SÃ£o Paulo - SP, 04116-160, Brasil\\n	-4.662.236	-2.359.336	FALSE
4	1555088331017	\Av. Prof. Almeida Prado, 532 - ButantÃ£, SÃ£o Paulo - SP, 05508-280, Brasil\\n	-467.347	-235.567	FALSE
5	1555096480931	\Rua Irede de Mattos Alonso, 88 - Vila Lageado, SÃ£o Paulo - SP, 05338-110, Brasil\	-4.674.083	-2.355.785	FALSE
6	1555104244664	\R. Heizo Nakano, 44 - Vila Jerusalem, SÃ£o Bernardo do Campo - SP, Brasil\\n	-4.655.505	-2372693	TRUE
7	1555104353956	\R. Heizo Nakano, 65 - Vila Jerusalem, SÃ£o Bernardo do Campo - SP, 09811-370, B	-4.655.472	-2372698	TRUE
8	1555104395304	\R. Heizo Nakano, 65 - Vila Jerusalem, SÃ£o Bernardo do Campo - SP, 09811-370, B	-4.655.472	-2372698	TRUE
9	1555112292764	\Rua Pico Della MirÃ¢ndola, 162 - Jardim Vila Mariana, SÃ£o Paulo - SP, 04116-000	-4.662.373	-2359378	FALSE
10	1555112959424	\Av. Prefeito FÃ¡bio Prado, 1085 - Jardim Vila Mariana, SÃ£o Paulo - SP, 04116-000,	-4.662.358	-2359343	FALSE
11	1555160532024	\Avenida Dr. JÃºlio Prestes - Ponte Alta, Aparecida - SP, 12570-000, Brasil\\n	-4.523.663	-2385209	FALSE

Fonte: arquivo pessoal

Foram obtidas 73 respostas distribuídas nos locais mostrados na Figura 29:

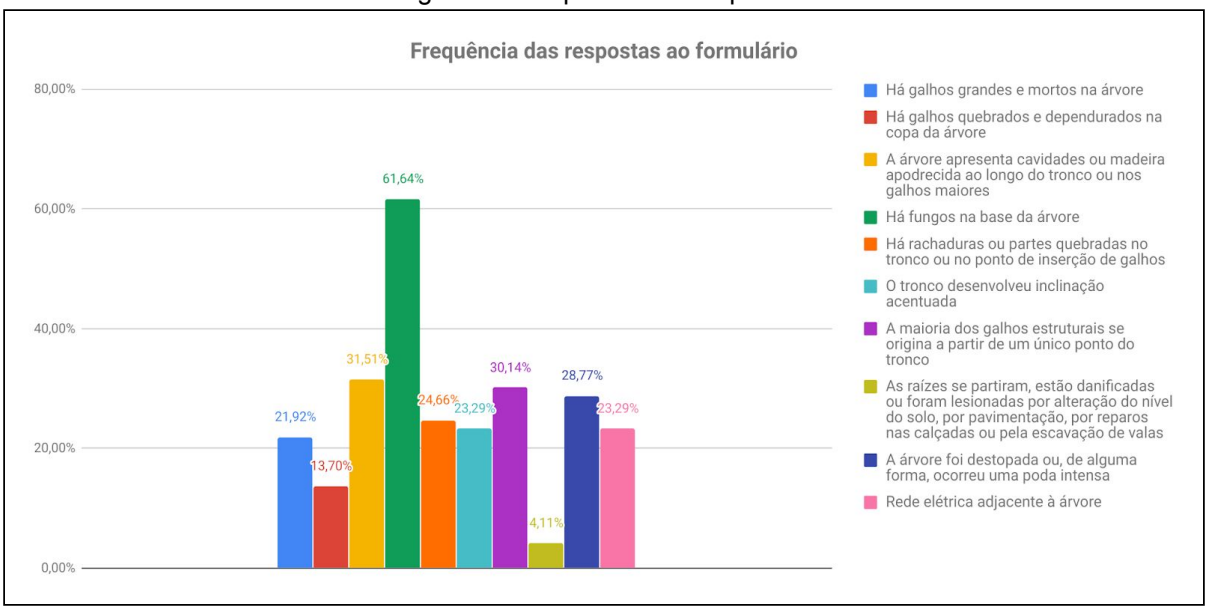
Figura 29: mapas de distribuição de coleta de dados



Fonte: arquivo pessoal

A frequência de respostas está distribuída conforme o gráfico mostrado na Figura 30.

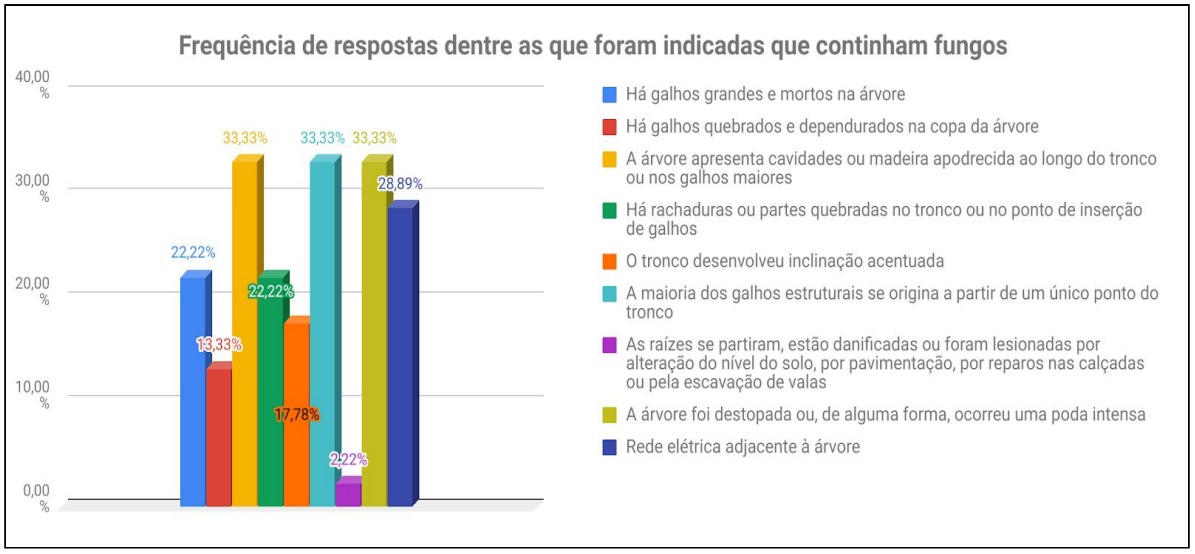
Figura 30: frequência de respostas



Fonte: arquivo pessoal

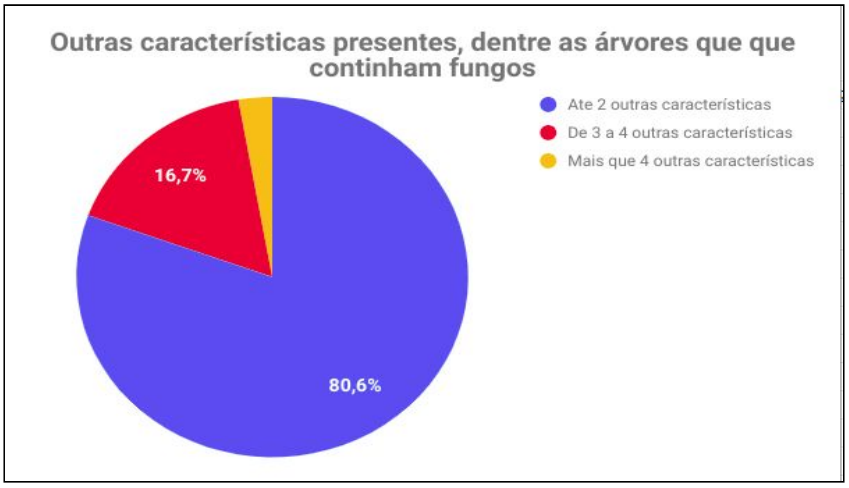
Dentre as que foram respondidas que continham fungos, podemos observar a relação com outras características na Figura 31, e também quais são essas características relacionadas na Figura 32.

Figura 31: frequência de respostas



Fonte: arquivo pessoal

Figura 32: distribuição de características presentes entre as árvores que continham fungos



Fonte: arquivo pessoal

Para a aplicação do algoritmo, as imagens foram reclassificadas quanto a presença ou não de fungos, independente da resposta ao formulário, dessa forma observamos 48 árvores com fungos.

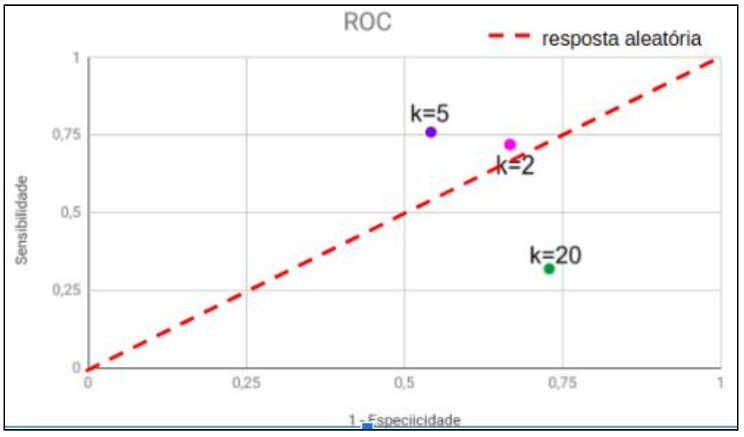
O algoritmo foi testado para alguns valores de *k*, onde *k* é o valor de vizinhos mais próximos utilizados no KNN e obtivemos os seguintes resultados:

Tabela 1. Resultados das medidas de performance

Nome	Significado	Resultado *k*=5	Resultado *k*=2	Resultado *k*=20
N	Total de amostras	73	73	73
p	Total de positivos reais	48	48	48
n	Total de negativos reais	25	25	25
p’	Total de positivos preditos	43	50	45
n’	Total de negativos preditos	30	23	28
TP (verdadeiro positivo)	se real positivo e foi predito positivo	35	32	26
FP (falso positivo)	se real negativo e foi predito positivo	8	18	19
TN(verdadeiro negativo)	se real negativo e foi predito negativo	17	7	6
FN (falso negativo)	se real positivo e foi predito negativo	13	16	22
Erro tipo I	FP/N	0,109	0,247	0,260
Erro tipo II	FN/N	0,178	0,219	0,301
Total Error (Erro Total)	Erro Tipo I + Erro Tipo II	0,288	0,466	0,562
Accuracy(Acurácia)	1 - Total Error	0,712	0,534	0,438
False Positive Fraction (Fração de Falso Positivo)	FP/N	0,320	0,720	0,760
True Positive Fraction (Fração de verdadeiro positivo)	TP/N	0,729	1,280	1,040
Precision (Precisão)	Porcentagem de acerto daqueles que foi predito positivo	0,814	0,667	0,542
Recall (Revocação)	Porcentagem de positivos verdadeiros dentre os elementos da classe positiva	0,729	0,667	0,542
Sensitivity(Sensibilidade)	Porcentagem de positivos verdadeiros dentre os elementos da classe positiva	0,729	0,667	0,542
Specificity(Especificidade)	porcentagem de negativos verdadeiros dentre os elementos da classe negativa	0,680	0,280	0,240

Para comparar os resultados, foi elaborada uma análise no espaço de características operacionais do receptor (ROC) que são úteis para organizar os classificadores e visualizar seu desempenho, (FAWCETT, 2006). (Figura 33)

Figura 33: características operacionais do receptor



Fonte: arquivo pessoal

Foi obtida então a seguinte matriz de confusão para k=5, onde ocorreu a melhor performance:

Tabela 2. Matriz de confusão.

		Valor Previsto	
		Positivos	**Negativos**
Valor real	**Positivos**	35	13
	Negativos	8	17

Fonte:arquivo pessoal

5. Conclusão

Com facilidade, foi criado o app para a coleta de dados, e em apenas uma semana foram observadas 73 árvores.

Ao recolher os dados foi identificado que a característica de interesse a ser analisada com algum algoritmo de classificação era o tipo de fungo, pois foram observados fungos em várias árvores, mas nem todos eles representam algum risco.

Foi observado que a maioria dos fungos encontrados era do tipo líquen, e eles não representam um risco de queda (BRAZOLIN, 2009). Talvez uma próxima

análise seria diferenciar o tipo de fungo, para poder relacioná-lo com algum fator de risco de queda.

Podemos observar que apenas 19,4% das árvores possuem mais de 2 sintomas associados, e mais de 60% possui algum tipo de fungo, dessa forma não podemos associar o risco de queda somente com a presença ou não de fungo.

De acordo com as medidas de performance verificamos que o algoritmo realmente está funcionando de acordo com o esperado, obtendo uma precisão de 81,4%, e um baixo índice de falso positivos, podemos observar também a sensibilidade(ou recall) que é a razão entre o número de positivos verdadeiros e o número de elementos da classe positiva é cerca de 73%.. Em um estudo futuro poderiam-se rever estratégias para aumentar a sensibilidade, pois no caso estamos negligenciando cerca de 25% das árvores com fungo, apesar de termos uma precisão razoável.

Esses resultados poderiam ser superiores se as features fossem selecionadas e filtradas, para diminuir a dimensionalidade do problema, já que estamos trabalhando com 768 features, utilizando os três canais do RGB.

Foi verificado também que a probabilidade de se encontrar uma árvore com fungo é maior do que sem fungo, de acordo com a distribuição dos nossos dados já que 65,7% das árvores verificadas tem fungos.

Esses dados colhidos estão disponíveis online e, podem ser de grande utilidade para órgãos públicos que desejam obter informações sobre as árvores em região urbana e podem ser utilizados para diminuir o incidência de quedas.

6. Referências

APPLE, Ananda. **Cidade de SP tem mais de 2 mil quedas de árvores em janeiro e fevereiro**. G1, São Paulo, 05 de mar. de 2019. Disponível em: <https://g1.globo.com/sp/sao-paulo/noticia/2019/03/05/cidade-de-sp-tem-mais-de-2-mil-quedas-de-arvores-em-janeiro-e-fevereiro.ghtml>. Acesso em: 29 de mar. de 2019.

BRAZOLIN, Sérgio. **Biodeterioração, anatomia do lenho e análise de risco de queda de árvores de tipuana, Tipuana tipu (Benth.) O. Kuntze, nos passeios públicos da cidade de São Paulo, SP**. Escola Superior de Agricultura “Luiz de Queiroz”, Piracicaba, 2009.

CROCKFORD D. **Apresentando como JSON (JavaScript Object Notation) funciona**. [S.l.] [2018.?]Disponivel em <http://www.json.org/>. Acesso em 28 out. 2019.

APPYBUILDER TEAM. **About AppyBuilder**. [S.l.] [2018.?]Disponivel em <https://appybuilder.com//>. Acesso em 28 out. 2019.

DOUGHERTY, Geoff. **Pattern Recognition and Classification. An Introduction**. New York: Springer, 2013.

FAWCETT. T. **An Introduction to ROC Analysis**. Pattern Recognition Letters. Palo Alto - CA, 19 de dezembro de 2006. n. 27, p. 861–874. Disponível em: <https://www.researchgate.net/publication/222511520_Introduction_to_ROC_analysis>. Acesso em: 20/09/2019

LAO, Natalie. **Developing Cloud and Shared Data Capabilities to Support Primary School Students in Creating Mobile Applications that Affect Their Communities.** Massachusetts Institute of Technology, 2016.

OPENCV TEAM. **Sobre OpenCV (Open Source Computer Vision Library)**. [S.l.]. [2019 ?] Disponível em: <https://opencv.org/about/> Acesso em: 12 out. 2019.

Rafael C. Gonzalez and Richard E. Woods. **Digital Image Processing (2nd ed.).** Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

Sociedade Internacional de Arboricultura. **Reconhecimento de Riscos de Árvores**. 2011. Disponível em: <http://www.treesaregood.org/portals/0/docs/treecare/TreeRisk.pdf>. Acesso em: 29 de mar. de 2019.

7. Apêndice

Código do classificador em Python, também disponível em: https://github.com/saitoaryane/treelearning.git

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import numpy as np
import os
import glob
from sklearn.neighbors import KNeighborsClassifier

# Funcao para extrair o histograma de cada imagem
def extract_features(image):
    color = ('b','g','r')

    for i,col in enumerate(color):
        histr = cv2.calcHist([image],[i],None,[256],[0,256])
        hist= np.concatenate(histr)
    return hist

# carregando as imagens de treinamento
train_path  = "./Fotos/Fotos/Dataset2/train/"
train_names = os.listdir(train_path)
print train_names
train_features = []
train_labels   = []

# iterando sobre os dados de treinamento
print "[STATUS] Started extracting features.."

for train_name in train_names:
        cur_path = train_path + "/" + train_name
        cur_label = train_name
        i = 1
        for file in glob.glob(cur_path + "/*.jpeg"):
                # lendo a imagem de treinamento
                image = cv2.imread(file)
                # extraindo o histograma da imagem
                features = extract_features(image)
                # adicionando ao vetor de features e labels
```

```
            train_features.append(features)
            train_labels.append(cur_label)
            # show loop update
            i += 1

# Tamanho dos vetores
print "Training features: {}".format(np.array(train_features).shape)
print "Training labels: {}".format(np.array(train_labels).shape)

# criando o classificador k=5 (valor padrao)
print "[STATUS] Creating the classifier.."
modelN= KNeighborsClassifier()

# fit os dados de treinamento e labels
print "[STATUS] Fitting data/label to model.."
modelN.fit(train_features, train_labels)
predict= []

# iterando sobre os dados de teste
for file in glob.glob("./Fotos/Fotos/Dataset2/*.jpeg"):
    # lendo a imagem
    image = cv2.imread(file)
    features = extract_features(image)
    #carregando o modelo e fazendo a predicao
    prediction = modelN.predict(features.reshape(1, -1))[0]
    #Adicionando a predicao no vetor
    predict.append(prediction)

print predict
```