Universidade Federal de Pernambuco
Centro de Informática

Graduação em Ciência da Computação

VALIDAÇÃO DE ASSINATURAS UTILIZANDO UM APP MOBILE

Carlos Henrique Gonçalves e Silva

Trabalho de Graduação

Recife
2017

Universidade Federal de Pernambuco
Centro de Informática

Carlos Henrique Gonçalves e Silva

VALIDAÇÃO DE ASSINATURAS UTILIZANDO UM APP MOBILE

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da computação.

Orientador: *Cleber Zanchettin*

Recife
2017

Dedico a todos que contribuíram para o bem da ciência.

Agradecimentos

Primeiramente, gostaria de agradecer a minha família por ser sempre meu porto seguro para todas as ocasiões. Em especial aos meu pais que me mostraram a importância da educação e sempre me ofereceram o suporte necessário para que eu continuasse buscando meus objetivos.

Agradeço também a todos os meus amigos, em especial aos que fiz durante o curso, que tornaram todos os momentos mais prazerosos. Um abraço em especial a todo o grupo BdR, eu levarei vocês sempre comigo.

Agradeço a Alex, Ana, Carlos Alberto, Carlos Júnior, Fabiana, Gustavo, Maria, Maria Eduarda, Newton, Rhuan, Roberto, Sasa, Karla, Kerol, Vera e Vitor que ajudaram a validar esse projeto.

Por fim, agradeço a todos os professores que eu tive na minha vida. Em especial, ao meu orientador Cleber Zanchetin por estar sempre disponível e por ter me auxiliado na escrita e no desenvolvimento deste trabalho.

Don't panic.
—DOUGLAS ADAMS (1981)

Resumo

Nesse trabalho foi proposta a criação de um sistema com o intuito de verificar a autenticidade de assinaturas em um dispositivo móvel. Esse sistema captura as assinaturas através de um aplicativo que funciona em dispositivos com o sistema operacional *Android*. As informações são então enviadas para o servidor, que por sua vez vai aplicar algoritmos de verificação de assinatura para validar se a assinatura é válida ou não. O algoritmo de verificação é baseado na composição dos algoritmos de *Dynamic Time Warping* (DTW) e do *One Class Support Vector Machine* (OC-SVM). A eficiência do sistema se aproximou de 96% na base de dados montada para esse trabalho. O algoritmo que usa *DTW* quando validado sozinho, obteve 90% de eficiência ao ser testado no *dataset* da *SigComp* (2011), uma das principais competições da área.

Palavras-chave: Reconhecimento de assinaturas, Inteligência artificial, assinaturas online, biometria, dtw, oc-svm

Abstract

In this article, we porpose a system that aims to verify the signature's authenticity in a mobile device. This system will get the signature through a screen's mobile device running Android Operation System. The signature's data are then sent to the server that will be the input for the verification algorithms to check if the signature is genuine or forgery. The verification's algorithm was developed based on the composition of the following algorithms, the *Dynamic Time Warping* (DTW) and *One Class Support Vector Machine* (OC-SVM). The general accuracy of the verification approximates 96% when tested with the created database for this project. The DTW's algorithm was validated stand-alone and the accuracy of the system was 90% using the SigComp database, one of the main competitions in the area.

Keywords: artificial intelligence, signature recognition, online signatures, biometric, dtw, oc-svm

Sumário

Lista de Figuras

Lista de Tabelas

Lista de acrônimos

API *Application Programming Interface*

DTW *Dynamic Time Warping*

OC-SVM *One Class Support Vector Machine*

CAPÍTULO 1

Introdução

Identificação pessoal diz respeito ao conjunto de medidas que precisam ser tomadas para identificar um indivíduo em uma população [1]. Há várias maneiras de identificar uma pessoa, pois cada indivíduo carrega consigo características que são suficiente para distingui-los dos demais. Portanto, podemos citar como exemplo três maneiras básicas para identificar uma pessoa:

- por meio de algo que ela sabe, algum tipo de informação secreta, como senhas e números de identificação;
- por meio de algo que ela possui, exemplos desse método são chaves físicas, passaportes, crachás e documentos de registro geral;
- por meio de características físicas ou comportamentais inerentes a pessoa. Isso é conhecido como a ciência da biometria [DHR15].

A biometria pode se basear em singularidades fisiológicas ou comportamentais para identificar um indivíduo. As características fisiológicas são as que usam atributos e formas do corpo que são únicas ao indivíduo, como por exemplo impressões digitais, formato da íris, formato facial, geometria das mãos, entre outros [DHR15]. Por outro lado, o reconhecimento comportamental diz respeito ao comportamento dos indivíduos e como podemos identifica-los a partir disso. Embora estes comportamentos, como a fala, a maneira de digitar ou a forma de assinar, possam parecer genéricos, cada indivíduo os executam de forma única possibilitando assim a sua identificação [HO04].

A identificação pessoal feita a partir da assinatura manuscrita ainda é uma das maneiras mais usadas quando falamos de identificação pessoal, mesmo quando comparados a identificadores mais precisos e práticos, como por exemplo o reconhecimento da impressão digital ou íris. Isso se deve, dentre outros fatores, ao fato da assinatura ser uma maneira eficaz e simples de ser executada. Além de ser uma maneira legalmente aceita para identificação pessoal, sendo amplamente utilizada em documentos oficiais por bancos e outros órgãos oficiais, no mundo todo, para indicar a autenticidade e consentimento do responsável [FBFa15].

No Brasil, os casos de falsificação de assinatura resultam em prejuízos milionários a pessoas e instituições [FBFa15]. Isso requer então uma sistema de verificação cada vez mais avançado para diminuir a ocorrência desses casos. O sistema eleitoral é outro cenário onde esse crime já foi aplicado, antes do voto ser realizado com a biometria, o voto era computado apenas mostrando um documento oficial com foto, e então a assinatura do eleitor. O mesmo crime ocorre em Vestibulares e Concursos públicos no país.

[1] https://www.gestaodesegurancaprivada.com.br/identificacao-pessoal/

Esses e outros tipos de fraudes relacionados a falsificação de assinaturas podem ser evitados pelo uso de um sistema que automatize a validação das mesmas. Uma opção que resolveria ou diminuiria grande parte desses problemas seria a utilização de um sistema de verificação e validação de assinaturas em dispositivos móveis. Isso tornaria a validação mais acessível, tendo em vista que atualmente esses dispositivos estão bem difundidos na nossa sociedade, além da sua mobilidade e relativo alto grau de poder computacional, facilitando assim a utilização desse sistema em diferentes tipos de ambientes.

Antigamente, os grafoscopistas, especialistas responsáveis pela validação da autenticidade de assinaturas, utilizavam técnicas manuais para executar o seu trabalho, o que tornava todo o processo demorado. Com o avanço da computação, técnicas automáticas foram surgindo e a medida que os resultados dessas técnicas se tornaram mais precisos, ganhava-se cada vez mais credibilidade entre esses profissionais.

A utilização das técnicas manuais apresentavam resultados satisfatórios, mas, apesar disso, um dos problemas enfrentados por esses profissionais é a falta de um padrão ao avaliar uma dada assinatura. Possibilitando assim a obtenção de resultados diferentes uma vez que a validação de um documento fosse feita por profissionais diferentes. Isso impulsionou e impulsiona os estudos no âmbito computacional para a resolução desse problema, principalmente na área de extração de características [NFM+12].

Os validadores, ou reconhecedores, de assinatura são divididos em dois grandes grupos, os que utilizam sistemas *on-line* e os que utilizam sistemas *off-line* [SBM14].

Nos sistemas *off-line* apenas a imagem final da assinatura é levada em consideração na hora do seu reconhecimento. Essa imagem pode ser obtida através de *scanners* ou através de câmeras, uma vez que essa captura só é realizada depois que o usuário termina de realizar toda a assinatura.

Por outro lado, nos sistemas *online* são utilizadas informações capturadas enquanto o usuário assina. Isso só é possível pois esse processo é feito utilizando telas digitalizadoras. Como as informações são extraídas durante a assinatura, algumas características não estão disponíveis e por isso não são levadas em consideração na versão *off-line*, como, por exemplo, dados relacionados ao tempo de assinatura, pressão na tela, troca de sentido da caneta e a quantidade de vezes em que a caneta foi levantada da tela, entre outras [HO04].

No geral, sistemas *off-line* precisam de um design mais sofisticado em comparação aos *online*, isso se deve ao fato da quantidade de características ser reduzida quando comparado ao *on-line* [GCH15].

Normalmente, em um sistema de reconhecimento biométrico, o dado a ser testado é comparado com os dados obtidos no momento de cadastro, ou seja, na etapa de treinamento do sistema. Esse processo pode ser utilizado nos processos de verificação ou reconhecimento. O processo de verificação, ocorre em casos em que deseja-se saber se os dados capturados pertencem a um indivíduo em específico. Já no caso de reconhecimento, o propósito é identificar qual dentre os usuários cadastrados, pertence aquela informação [DHR15].

Existem vários tipos de assinaturas falsas. A complexidade para reconhecer uma assinatura falsa varia com o tipo de assinatura que estamos validando [DHR15]. As assinaturas falsas são divididas nessas categorias:

1. Quando o falsificador não conhece a assinatura da vítima;

2. Quando o falsificador conhece o nome do dono da assinatura;

3. Quando o falsificador já viu a assinatura;

4. Quando o falsificador tem acesso a assinatura original durante a assinatura forjada;

5. O falsificador tem acesso a informações capturadas de uma assinatura *online* [DHR15].

A classe 1 representa o conjunto de assinaturas em que o falsificador não tem nenhum conhecimento do que ele está falsificando. Esse caso pode ser representado caso a assinatura de um usuário seja validada utilizando um identificador diferente do identificador do autor da assinatura.

A classe 2 representa a classe onde tudo que o falsificador sabe é o nome, o que facilita algumas escolhas, como a ordem que as letras aparecem, por exemplo. A classe número 3 representa uma cópia quando o forjador tem acesso ao desenho da assinatura mas não tem informações de como ela é escrita.

A classe 4 representa uma falsificação em que seria possível fazer uma espécie de decalque da assinatura original. A classe 5 é composta quando o falsificador tem algum conhecimento sobre a forma final da assinatura, assim como informações capturadas de uma assinatura *online*. A classe 5 pode facilmente ser divididas em classes menores, dependendo da quantidade e qualidade de recursos que o falsificador possuí disponível [DHR15].

Em grande parte dos trabalhos realizados utilizando classificadores *online*, são utilizados mesas digitalizadoras específicas e em ambientes controlados, em vez de dispositivos móveis, que por sua vez são mais utilizados em ambientes mais dinâmicos e em diferentes contextos [MMDG14].

Como exemplos disso podemos citar casos em que a assinatura poderia ser obtida com o usuário sentado e apoiado em alguma superfície, em outros casos ela seria obtida com o usuário em pé ou até mesmo andando, levando o sistema a ser usado com diferentes inclinações e ambientes. Além disso, os dispositivos móveis variam no que diz a respeito de hardware e recursos computacionais, o que dificulta na criação de um classificador único que seja escalável [SBM14].

1.1 Objetivo

Este trabalho tem como objetivo o estudo e a implementação de um sistema de verificação de assinaturas a ser utilizado em dispositivos móveis. O sistema permitirá a captura e validação da assinatura no dispositivo móvel. Para a solução se tornar escalável, será necessário a criação de um servidor de processamento, o qual auxilia o dispositivo móvel na verificação das assinaturas, centralizando informações dos usuários.

Como objetivos específicos deste trabalho, podemos listar:

- Construção de um aplicativo para dispositivos móveis que capture e extraia informações de uma assinatura;

- A construção de um verificador de assinaturas cujo processo de treinamento seja realizado somente com assinaturas válidas. Uma vez que no âmbito prático, como em cadastro em bancos, só estará disponível assinaturas válidas;
- Criação de um banco de dados para armazenar as informações dos usuários e das assinaturas, que serão utilizadas na verificação futura do usuário;
- Criação de um servidor para que hospede o banco de dados e que receba as características de uma assinatura, e utilize esses dados como entrada do verificador para classificar a assinatura como original ou cópia;
- Validação e testes usando a solução.

1.2 Organização do trabalho

Este trabalho está dividido em cinco capítulos. No segundo capítulo, será detalhado a revisão bibliográfica. No terceiro capítulo, serão abordados os detalhes da implementação e da arquitetura do sistema. No quarto capítulo iremos descrever os resultados obtidos. Por fim, o quinto capítulo foi reservado para as conclusões e trabalhos futuros.

CAPÍTULO 2

Revisão da Literatura

Um sistema de reconhecimento biométrico opera comparando informações biométricas do item a ser validado, que pode ser uma assinatura manuscrita ou uma impressão digital, por exemplo, com as informações adquiridas no momento do cadastro [DHR15]. Existem várias abordagens possíveis para verificar a veracidade de um item biométrico, como é o caso da assinatura. Uma vez que existem muitas formas de extrair as características necessárias, assim como várias técnicas de inteligência artificial para identificação da mesma. Nesse capítulo, discutiremos algumas dessas abordagens.

Guerbai et al (2015) [GCH15] propôs um sistema de verificação de assinaturas que usa uma *One Class Vector Support Machine* (OC-SVM)[MMR$^+$01] para validar a assinatura. As características utilizadas em seu trabalho, que são obtidas através da transformada de *Curvelet*, são independentes do autor da assinatura. Ou seja, o verificador utiliza atributos globais em vez de específicos para cada usuário.

Geralmente o número de assinaturas é bem restrito na etapa do treinamento de sistemas como esse, uma vez que não é muito prático conseguir diversas assinaturas do mesmo usuário no momento do cadastro. O autor então propõe a implementação de um limiar de aceitação que é baseado na média e no desvio padrão dentre as assinaturas válidas, com o intuito de evitar o possível sobre-treinamento que o algoritmo possa sofrer.

As características das assinaturas *online* capturadas por Heinen et al.(2004) são variadas, incluem tempo da assinatura, trocas de sentido do eixo, quantidade de vezes que a caneta foi levantada, interseções da assinatura, distribuição da densidade dos pontos assim como as velocidades médias e a máxima. Em seu artigo, Heinen et al. (2014) utiliza uma Rede Neural Artificial conhecida como *Multi Layer Perceptron* (MLP) [PM92] para a verificação da assinatura. O sistema obteve média total de acerto de 99.86% [HO04].

Chen et al. (2016) [CXL16] propôs um sistema de verificação de assinaturas *online* que utiliza a distância calculada utilizando o algoritmo *Dynamic time warping* (DTW) [KR05] com um fator limiar de aceitação baseado em características que podem ser calculadas primariamente a partir do tempo e da posição da assinatura em um plano cartesiano, como velocidades, lista de acelerações e ângulos.

Esse artigo faz um comparativo com limiares de aceitação dependente do autor com um limiar independente do autor e comum para todos os autores. Os resultados mostraram que um limiar calibrado para cada autor é mais eficiente do que um geral, e o artigo então utiliza esse método para gerar seus resultados.

Hafemann et al. propôs um estudo sobre o aprendizado de *features* que seja independente dos usuários, ou seja, ele almeja encontrar *features* que sejam comuns em assinaturas falsas, por exemplo, utilizando redes neurais convolucionais para tal. Como resultado, eles conseguiram

melhorar o estado da arte de algumas *datasets* importantes da área [HSO17].

Arora et al.(2015)[ASK15] propôs um verificador que utiliza a distância entre duas assinaturas para verificar assinaturas. A tomada de decisão se dá a partir da comparação da distância entre as características comparadas com um limiar. Um ponto curioso desse artigo é que o limiar é escolhido testando vários valores dentro de um conjunto de valores e é utilizado o que teve maior taxa de acerto. Três algoritmos de distância são validados, distância *Manhattan*, distância *Chebyshev* e a distância euclidiana. A distância euclidiana foi a que teve melhor desempenho.

Fayyaz et al.(2015) propôs uma abordagem utilizando *deep learning* que utiliza algoritmo com auto-aprendizado para selecionar as características que vão ser utilizadas para verificar a assinatura. Depois da extração de características, elas são aplicadas em um *OC-SVM*. O resultado do artigo mostra performance superior do que outros algoritmos, o que evidencia a importância de uma boa escolha das características [FSSF15].

Nas próximas sub-seções estão descritas as fundamentações teóricas de dois algoritmos. Os algoritmos são o *Dynamic Time Warping* e o *One Class Support Vector Machine*.

2.1 *Dynamic Time Warping*

DTW é um algoritmo muito utilizado para comparar sequências temporais. Por ser um classificador bem diverso, ele é usado como ferramenta para resolver diversos problemas, como por exemplo, reconhecimento de fala, reconhecimento de linguagens de sinais, visão computacional, reconhecimento de sinais e muitos outros [Sen09].

A ideia principal do *DTW* é permitir o ajuste de dados no tempo e no uso de programação dinâmica para otimizar o ajuste de séries temporais. O algoritmo começa criando uma matriz de distância $C \in \mathbb{R}^{NxM}$ que representa todos os pares entre X e Y, sendo X e Y duas entradas que se deseja comparar a distância entre elas, N e M são o tamanho dessas entradas. Essa matriz de distâncias é chamada de matriz de custo local para o alinhamento de X e Y e é definida por:

$$C_l \in \mathbb{R}^{NxM} : c_{i,j} = ||x_i - y_j||, i \in [1:N], j \in [1:M]$$

Uma vez que a matriz de custo local é construída com base na distância entre as entradas, o algoritmo encontra o caminho de alinhamento que tem o menor custo na matriz. Esse alinhamento define a correspondência de um elemento $x_i \in X$ to $y_i \in Y$ seguindo a condição que atribui o primeiro ao último elemento de X e Y.

A definição do alinhamento encontrado pelo *DTW* é uma sequência de pontos $p = (p_1, p_2, \ldots, p_k)$ no qual cada ponto desse caminho representa uma par de elementos, um que representa um elemento de uma entrada e o outro da segunda entrada. Isso é, $p_l = X(i)Y(j) \in [1:N]x[1:M]$ para $l \in [1:K]$ que satisfaz os seguintes critérios:

1. Critério da fronteira $p_1 = X(1), Y(1)$ e $p_k = X(N)Y(M)$. O primeiro e último ponto deve ser o primeiro e último ponto das sequências;

2. Critério da monotonicidade Essa condição garante que o alinhamento não "volte no tempo". Ou seja, o item anterior não pode vir depois do seu sucessor $n_1 \leq n_2 \ldots \leq n_k$ e $m_1 \leq m_2 \leq$

$\ldots \leq m_k$;

3. Critério da continuidade Esse critério garante que o algoritmo não pulou muitos pontos de uma vez. Ou seja, o próximo elemento é no máximo 1 elemento a mais em ambos as entradas $p_{l+1} - p_l \in \{(1,1),(1,0),(0,1)\}$.

O custo associado ao caminho é computado com base na matriz de peso local definida anteriormente:

$$c_p(X,Y) = \sum_{l=1}^{L} c(x_{n_l}, y_{n_l})$$

A caminho com o menor custo associado é chamado de caminho de custo ótimo. A distância euclidiana normalmente é utilizada para mensurar a similaridade.

2.2 *Support Vector Machine*

One Class Support Vector Machines são um caso especial de *Support Vector Machines* que permitem classificar objetos de apenas um tipo, e distinguir objetos dessa classe de todas os outros objetos possíveis [GCH15]. Essa máquina recebe um conjunto de treinamento que contém somente objetos pertencentes a classe desejada, e então o algoritmo mapeia todas as características desses dados em um espaço de maior dimensão através de uma função que chama *kernel*. O algoritmo então delimita uma região que agrupa, da melhor maneira possível, grande parte dos objetos da classe.

OC-SVM pode ser descrita como uma função $f(x)$ que define se um dado pertence a classe genuína ou não. A função $f(x)$ pode ser definida como:

$$f(x) = sgn\{\sum_{i=1}^{m} \alpha_i K(x,x_i) - \rho\}$$

no qual m é a quantidade de dados no conjunto de treinamento, ρ é a distância da esfera até o início, α_i são os multiplicadores de Lagrange que são obtidos através da otimização da seguinte fórmula:

$$\min_{\alpha} = \{\frac{\alpha_i \alpha_j K(x_i,x_j)}{2}\}$$

as seguintes regras são aplicadas no valor de α:

$$0 \leq \alpha_i \leq \frac{1}{vm}$$

$$\sum_{i}^{m} \alpha_i = 1$$

no qual v é a porcentagem de amostras que foram selecionadas para serem *outliers*, K é a função *kernel*. A função *kernel* é responsável pela projeção do valor em seu campo original

para o campo de *features*. Essa projeção permite a separação das amostras por meio de uma função de decisão [GCH15]. A amostra será aceita se o sinal da função $f(x)$ for positivo, será rejeitada caso contrário.

Várias funções *kernel* podem ser utilizadas, a mais comum é a *Radial Basis Function* (RBF), que permite determinar o raio da esfera de aceitação com base na variável γ. A definição de *RBF* é:

$$K(x, x_i) = exp(-\gamma \times d(x, x_i))$$

no qual $d(x, x_i)$ é a distância da amostra x para a amostra de treinamento x_i, e γ é o parâmetro do *kernel*. Podemos ver a representação na figura 2.1.

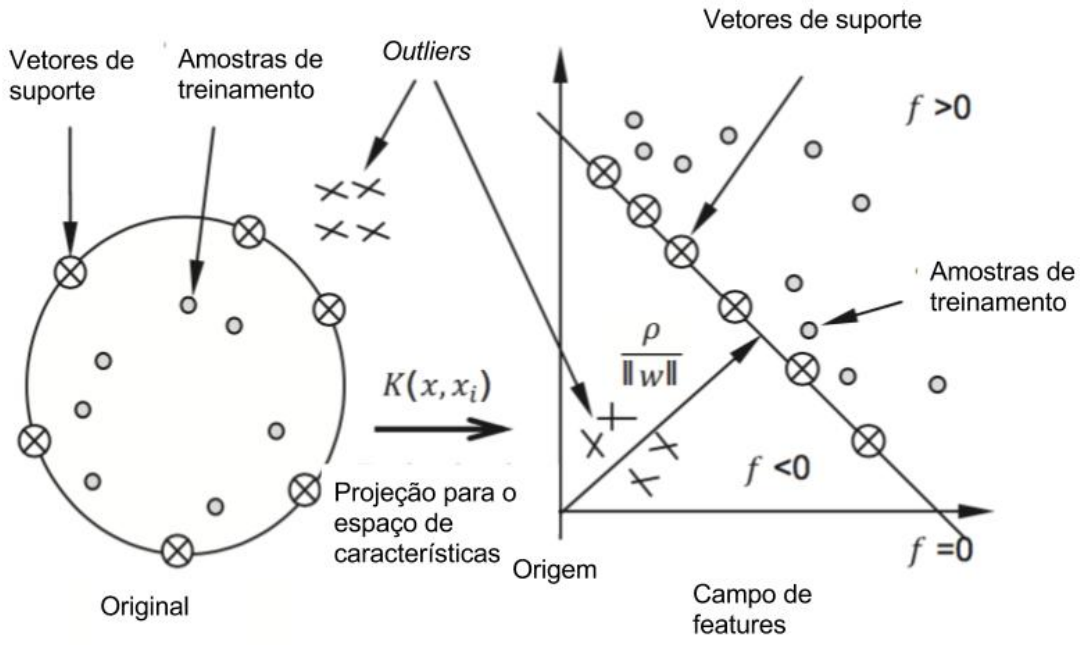


Figura 2.1 Classificação de amostras utilizando o *OC-SVM*. Adaptado de [GCH15]

CAPÍTULO 3

Descrição do sistema proposto

Neste capítulo serão descritas as especificações e o detalhamento do sistema proposto neste trabalho. Na primeira seção, encontra-se o levantamento dos requisitos, então são descritos os detalhes e processos que foram utilizados na criação da interface gráfica, em seguida temos uma descrição detalhada da arquitetura do sistema, e por fim, detalhes da implementação propriamente dita.

3.1 Requisitos

Em Engenharia de Software, requisitos podem ser definidos como o conjunto das restrições, especificações matemáticas e descrições abstratas de alto nível de um sistema [Som06]. O conjunto de requisitos é uma das formas de descrever uma especificação de um sistema, no qual todos os objetivos daquele software devem estar descritos. Existem dois tipos de requisitos, requisitos funcionais e requisitos não-funcionais. Os funcionais descrevem as funcionalidades que os usuários precisam ou querem que o sistema ofereça. Por outro lado, os requisitos não-funcionais são atributos globais de um software, como usabilidade, desempenho, especificações de hardware, entre outros.

3.1.1 Requisitos Funcionais

Como descrito anteriormente, os requisitos funcionais são requisitos que definem funções do sistema, o que o sistema deve fazer em determinada situação ou como o sistema deve reagir a partir de certo *input* ou comando [Som06].

Segue a lista de requisitos funcionais levantadas inicialmente para esse sistema:

- O sistema deve ser capaz de capturar a assinatura do usuário;
- O software deve ser capaz de extrair as características da assinatura captura;
- O software deve ser capaz de mandar requisições ao servidor com o intuito de cadastrar o usuário;
- O software deve ser capaz de receber uma resposta do servidor que contém a resposta contendo o status do cadastro;
- O cadastro deverá conter ao menos cinco assinaturas verdadeiras e o CPF, o qual é único para cada usuário;

- O software deve mandar requisição para o servidor, contendo o CPF do usuário e as características da assinatura capturada para validação dessa assinatura;
- O software deve ser capaz de receber a resposta do servidor contendo a análise da assinatura, se ela é equivalente as capturadas para aquele usuário ou não;
- O servidor deve ter um mecanismo de classificação para classificar as assinaturas como originais ou cópias;
- O classificador utiliza em seu treinamento as assinaturas provindas do cadastro do usuário;
- O servidor deve ser capaz de receber uma requisição de cadastro e responder com o status do cadastro, se foi cadastrado com sucesso ou não;
- O servidor deve ser capaz de receber uma requisição contendo uma assinatura a ser avaliada e o CPF do usuário, e então responder com uma boa precisão se a assinatura foi feita pelo usuário especificado.

3.1.2 Requisitos Não-Funcionais

Os requisitos não-funcionais são atributos sobre o sistema, tais como restrições de tempo, restrições sobre o processo de desenvolvimento, e linguagens de programação, assim como outras restrições que possam impactar no desenvolvimento do software [Som06].

Segue a lista de requisitos não-funcionais levantadas para esse sistema:

- A parte mobile do sistema deve rodar no sistema operacional *Android* 6.0 ou superior;
- O sistema não deve demorar mais que 1 minuto para realizar uma validação;
- O tamanho da tela do aparelho móvel deve conter no mínimo 4.5 polegadas;
- O software deve ser de fácil utilização;
- O software deve ter um bom desempenho, tanto com assinaturas obtidas com uma caneta *Stylus* ou com as assinaturas obtidas utilizando o dedo para a captura.

3.2 Interface Gráfica do Usuário

O software foi desenvolvido e pensado para a plataforma *Android*. A interface padrão de aplicativos *Android* é renderizada a partir de um arquivo *xml* adaptado que dá suporte ao desenvolvimento de diversos componentes da Interface de usuário de usuário (GUI) nativos. Para os itens mais comuns que compõem a GUI, o *layout* e controles já estão totalmente implementados no kit de desenvolvimento básico, sendo necessário apenas invocar esses elementos, fazendo referência no código fonte. Exemplos desses objetos de *GUI* são caixas de texto, elementos de seleção, elementos para criação de lista, botões, caixas de texto, entre outros. Cada um desses

elementos são chamados de *View*, no *Android*. Esses e outros componentes são suficientes para criar a boa parte da interface gráfica básica da grande maioria dos aplicativos disponíveis para plataforma. [1]

"Todos os elementos da interface do usuário em um aplicativo para *Android* são criados usando objetos *View* e *ViewGroup*. Uma *View* é um objeto que desenha algo na tela com o qual o usuário pode interagir. Um *ViewGroup* é um objeto que contém outros objetos *View*(e *ViewGroup*) para definir o layout da interface."[2]

Porém, nem todas as partes que foram necessárias para desenvolver esse aplicativo estavam disponíveis na framework nativa da interface padrão do *Android* [3]. Não foi encontrado nenhum componente nativo que oferecesse um espaço para a captura da assinatura, por exemplo. Então, foi utilizado a biblioteca *android-signature* [4], com essa biblioteca, torna-se possível adicionar uma *Android View* customizada para desenhos de assinaturas.

Essa biblioteca encontra-se disponível no site *Github* [5], que é um site que hospeda projetos, em sua maioria de software. Essa plataforma ainda oferece uma série de recursos para o desenvolvimento de software colaborativamente. Alguns exemplos são rastreio de *bugs*, pedido de alteração de código, organização de tarefas, entre outros. Esses recursos estão disponíveis para comunidades *open-source* ou para códigos privados de empresas.

A maioria das bibliotecas disponibilizadas no *Github* possuem algum tipo de licença, que determina quais são as regras de distribuição e regras de alteração que aquele software ou fragmento de software pode sofrer. A biblioteca em questão é distribuída seguindo os modelos da licença do tipo Apache 2.0. Essa licença é conhecida por ser uma licença permissiva, isso significa que essa biblioteca pode sofrer alterações e ser distribuídas em ambiente fechado sem nenhum tipo de custo ou permissão adicional por parte dos desenvolvedores originais do software. Essa licença, porém não assegura nenhum tipo de garantia ou responsabilidade por parte dos distribuidores do código.

Como temos permissão para alterar a biblioteca *signature-pad*, algumas funções foram adicionadas na biblioteca para melhor se adequar com as necessidades do projeto. As alterações que foram necessárias foram:

1. A cada *onMove()*, que é chamado toda vez que o usuário move o dedo ou caneta, de um ponto para outro no campo de assinatura, foi feita uma modificação para retornar o ponto anterior e o atual;

2. Alguns métodos auxiliares de comparação entre dois pontos, para facilitar o cálculo de algumas características, como Velocidade ou Aceleração também foram adicionados;

3. Foi adicionada a funcionalidade de capturar a pressão que a caneta ou dedo está fazendo sobre o *tablet*. Essa característica só funciona em alguns *tablets*, pois é necessário um hardware compatível.

[1] https://developer.android.com/guide/topics/ui/index.html?hl=pt-br
[2] https://developer.android.com/guide/topics/ui/overview.html
[3] https://developer.android.com/guide/practices/ui_guidelines/index.html
[4] https://github.com/gcacace/android-signaturepad
[5] https://github.com

Baseado nos requisitos levantado inicialmente, constatou-se que o aplicativo precisaria de quatro telas para suprir todas as funcionalidades corretamente. A descrição de cada tela poderá ser conferida na lista abaixo:

1. **Tela de validação**: Essa tela será responsável pela validação de uma assinatura. Seus elementos principais serão: id do usuário a ser validado, campo para entrada da assinatura, elemento visual que indicará a progresso da requisição, o campo para exibição do resultado da validação e links para as outras telas;

2. **Tela de Cadastro**: Essa tela será responsável pelo cadastro de novos usuários. Os elementos principais serão o campo do id do novo usuário, campo para a entrada das assinaturas, um elemento visual que indicará a progresso da requisição e o campo para exibição do status do cadastro;

3. **Tela de Configuração**: Essa tela será responsável pela configuração do aplicativo;

4. **Tela de Sobre**: Essa tela será responsável por exibir uma breve descrição do projeto, o propósito do mesmo assim como as menções para as bibliotecas utilizadas no processo de desenvolvimento.

Uma vez feita a prototipação inicial das telas e suas funcionalidades, foram desenvolvidas as telas finais. O design da maioria dos componentes foi feito de acordo com o *material design* oferecido pelo próprio *Android* [6]. Essa decisão foi tomada uma vez que os usuários já estão acostumados com esse padrão por causa dos outros aplicativos do sistema. As imagens das telas estão mostradas na Figura 3.1.

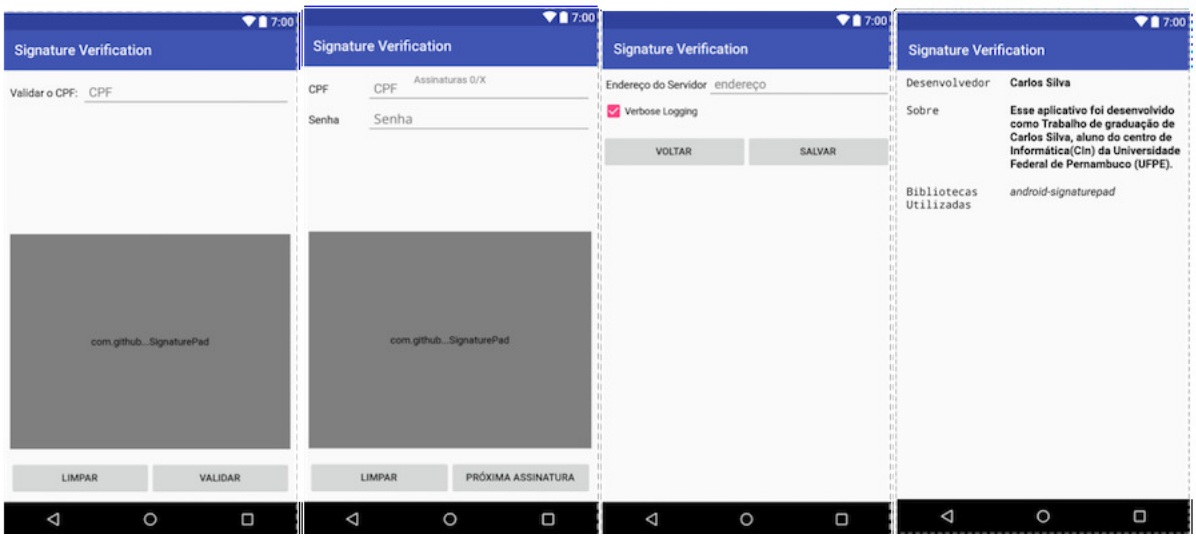


Figura 3.1 Imagem das telas do aplicativo desenvolvidos no Android Studio

[6]https://developer.android.com/training/material/index.html

3.3 Detalhes Técnicos

Uma vez definidos detalhes da interface e dos requisitos, o próximo passo é a definição dos detalhes técnicos. Nesta seção, será descrita uma visão geral da arquitetura do software e então será apresentado os detalhes da implementação da ferramenta.

3.3.1 Arquitetura

Uma vez que a abrangência, quantidade e complexidade de softwares em geral estão aumentando, a complexidade também cresce na mesma proporção, e com ela a dificuldade de organizar e estruturar os componentes. São essas razões que tornam o estudo da Arquitetura de software cada vez mais importante [SLF10]. A definição clássica de arquitetura de software dada por Shawn e Garlan (1994) é que a arquitetura de um software define o que é o sistema em termos de seus componente e o relacionamento dentre os componentes a ser utilizada na construção de um software [GS94]. Isso é, representa a documentação formal, de todos os componentes que serão necessários durante a etapa de desenvolvimento.

Uma boa documentação permite uma análise prévia das dificuldades serão encontradas, e como devem ser superadas. Normalmente, essa especificação se dá sem levar em conta as tecnologias específicas que devem ser utilizadas, o que geralmente é uma boa prática pois foge de como resolver um problema de uma maneira focal e possibilita ter uma visão mais ampla das soluções. Em outras palavras, funciona como uma espécie de "esqueleto"do produto final.

Devido o tamanho desse projeto e sua finalidade prioritariamente acadêmica, foi decidido usar uma documentação de software simplificada e informal. Uma estrutura mais simplificada para acelerar o desenvolvimento e a organização das partes deste código sem a preocupação de documentar todas as partes do sistema formalmente. Foi utilizado um princípio de metodologia ágil para o desenvolvimento.

Partindo da especificação de requisitos funcionais e não-funcionais, percebeu-se que o projeto precisaria ter duas grandes engrenagens que agiriam juntas, uma seria a parte do servidor, e outra seria a parte mobile. As duas engrenagens trabalham em conjunto para que o sistema possa entregar o resultado esperado. Essa descrição se encaixa no padrão de arquitetura Cliente-Servidor.

Cliente-Servidor é uma das estruturas mais utilizadas em aplicações distribuídas. Essa arquitetura se baseia em dividir as tarefas e funções do sistema em dois agentes, os servidores e os clientes, que se comunicam geralmente a partir de uma rede de computadores. Uma parte dessas funções vai ser destinada a um agente chamado de servidor, que normalmente irá oferecer serviços para que, um ou mais clientes, complete alguma função ou uma determinada tarefa. Normalmente, os clientes precisam de acesso a informações que não estão disponíveis localmente, mas que podem ser requisitadas ao servidor. Os clientes que estão impossibilitados de acessar o servidor tem suas funcionalidades diminuídas ou completamente invalidadas [Hel00].

Os clientes tem por função primária requisitar funções do servidor, e uma vez requisitadas, os mesmos ficam livres para executarem alguma outra função localmente, enquanto esperam a resposta do servidor para a ação inicial. Os servidores, por sua vez, esperam requisições dos clientes, e então processam e respondem com os dados e computações requisitados pelo

clientes. Alguns exemplos clássicos que utilizam esse modelo cliente servidor são o protocolos e clientes de *e-mail* e *web-sites* da *World Wide Web* (WWW). Na figura 3.2 temos uma representação do esquema cliente-servidor que será utilizado neste projeto.

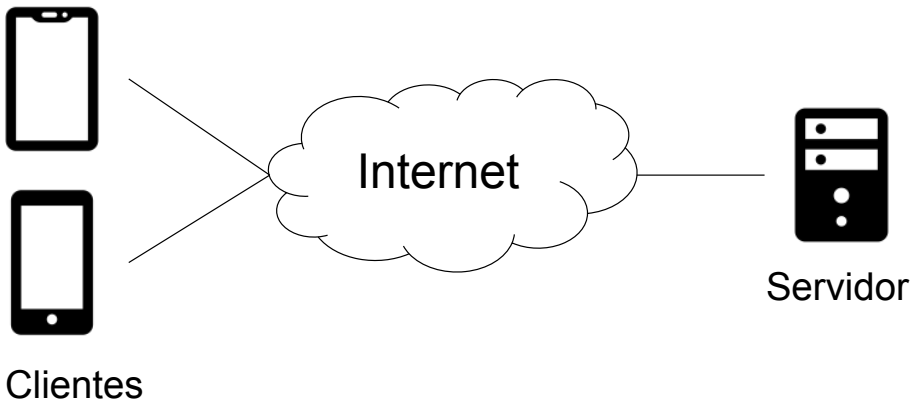


Figura 3.2 Esquemático do modelo Cliente-Servidor

3.4 Implementação

Implementação é definida como a realização da especificação técnica de um programa por meio da linguagens de programação. Ou seja, é a concretização das necessidades e requisitos esperados por um sistema. Os detalhes da implementação deste trabalho serão descritas nas próximas sub-seções. Para facilitar o entendimento, a descrição foi separada em quatro subtópicos, são eles: Classificador, Servidor, Banco de Dados e Aplicativo.

3.4.1 Classificador

Após a análise do estado da arte, foi decidido o uso das tecnologias *DTW* e *OC-SVM* para o desenvolvimento do classificador. Os motivos da escolha do *DTW* foram que esse algoritmo não depende diretamente do tamanho das entrada, o que é uma boa qualidade quando se trata de comparações entre assinaturas, uma vez que as "deformações"entre as assinaturas vão se repetir, porém, nem sempre no mesmo instante. Para a implementação de um classificador que utilize *DTW*, verificou-se que o artigo de Chen et al. (2016) obteve uma performance superior quando comparado a artigos similares, e por isso essa parte do algoritmo foi inspirada nesse trabalho [CXL16].

Outro algoritmo que se mostrou interessante para esse tipo de problema foi o *OC-SVM*, devido a sua versatilidade de ser usada para tratar diferentes unidades e só precisar de um tipo classe para classificar. Ou seja, esse classificador só necessita de informações verdadeiras para classificar. Para concretizar o uso do *OC-SVM*, o algoritmo utilizado foi inspirado no artigo de

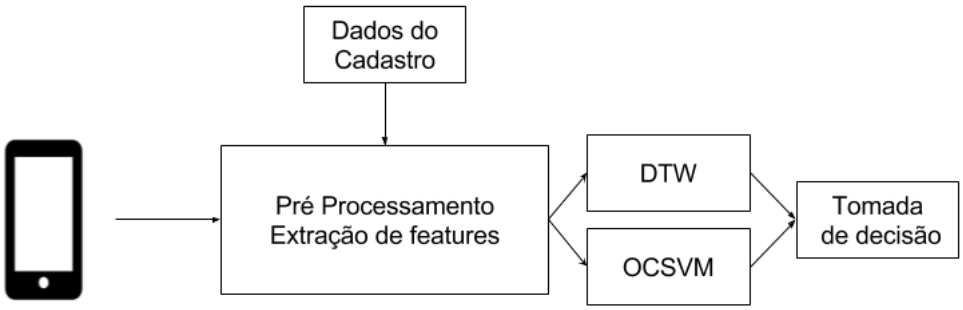


Figura 3.3 Imagem do fluxo dos verificadores

Guerbai et al. (2015) [GCH15].

Esses artigos vão ser utilizados da seguinte maneira:

- O classificador final proposto neste trabalho é baseado em dois classificadores intermediários. Um deles será baseado no algoritmo de *Dynamic Time Warping* (*DTW*) e o outro no algoritmo de *One Class Support Vector Machine* (*OC-SVM*)
- O verificador que é baseado em *DTW*, usa as características e limiar descritos no artigo de Chen et al. [CXL16]
- O verificador baseado em *OC-SVM* usa as características descritas no artigo de Heinen [HO04], porém com o limiar e algoritmo do artigo de Guerbai [GCH15]

Existem duas maneiras principais de se construir um sistema de verificação de assinaturas, pode ser independente do usuário ou dependente do usuário. A principal diferença entre eles é que no dependente do usuário, o sistema possui um verificador exclusivo para cada um dos usuários. No independente do usuário, por outro lado, é utilizado um classificador padrão para todos os usuários. Normalmente os verificadores dependentes do usuário possuem uma performance superior quando comparados aos independentes [HSO17]. O verificador baseado em *DTW* é independente do usuário enquanto o *OC-SVM* é dependente do usuário.

Em linhas gerais o verificador deste trabalho funciona da seguinte forma:

1. Um conjunto de assinaturas válidas é obtido na etapa de cadastro. Esse conjunto então passa pelas etapas de pré-processamento e extração das características. Uma vez que os dados das características vão ser utilizados pelo algoritmo de verificação;
2. Para verificar uma nova assinatura, primeiro ela passa pelas etapas de pré-processamento e extração de características. E então as características da assinatura a ser verificada são utilizados em conjunto com os dados das assinaturas do cadastro para validação com *Dynamic Time Warping* (DTW) e *One Class Support Vector Machine* (OC-SVM);

Tabela 3.1 Tabela de Características utilizadas pelo DTW

#	Descrição da característica	Definição
1	Coordenada x	x(n)
2	Coordenada y	y(n)
3	Vetor deslocamento	$S(n) = \sqrt{x+y}$
4	Velocidade no eixo X	$V_x(n) = [x(n+1) - x(n-1)]/2$
5	Velocidade no eixo Y	$V_y(n) = [y(n+1) - y(n-1)]/2$
6	Velocidade Total	$V(n) = \sqrt{{V_x}^2(n) + {V_y}^2(n)}$
7	Aceleração no eixo X	$a_x(n) = [V_x(n+1) - V_x(n-1)]/2$
8	Aceleração no eixo Y	$a_y(n) = [V_y(n+1) - V_y(n-1)]/2$
9	Aceleração Total	$a(n) = \sqrt{a_x^2(n) + a_x^2(n)}$
10	Coseno do ângulo entre o eixo x e a curva da assinatura	$cos\alpha = \frac{x(n+1)-x(n)}{\sqrt{[x(n+1)-x(n)]^2+[y(n+1)-y(n)]^2}}$
11	Coseno do ângulo entre o eixo x e a velocidade total	$cos\beta = V_x(n)/V_y(n)$
12	Ângulo entre o eixo x e a curva da assinatura	$\theta(n) = arctan\frac{y(n+1)-y(n)}{x(n+1)-x(n)}$
13	Velocidade angular	$V_\theta(n) = [\theta(n+1) - \theta(n-1)]/2$
14	Aceleração centrípeta	$a_c(n) = [V_x(n) \times a_y(n) - V_y(n) \times a_x(n)]/V(n)$

3. Então, o algoritmo toma a decisão final, como será mostrado no item 3.4.1.3, isso é, aceitar como uma assinatura válida ou recusar, com base nos resultados dos algoritmos executados no passo anterior.

O diagrama do processo usado na verificação está descrito na Figura 3.3.

Os detalhes de cada um desses algoritmos é descrito nos próximos tópicos.

3.4.1.1 DTW

As características utilizadas por esse algoritmo são baseadas e calculadas em termos dos eixos x e y que são obtidos através do dispositivo móvel. No classificador descrito por Chen(2016)[CXL16], além das características descritas na tabela 3.1, também é utilizado a pressão. No entanto, por não ser uma característica disponível para todos dispositivos *android*, ela não foi utilizada.

A lista de características está descrita na tabela 3.1. Na tabela, n é a representação da discretização do tempo.

A verificação de uma assinatura é avaliada com base no cruzamento das características extraídas da mesma com as extraídas das genuínas que estão no banco de dados. Esse cruzamento de informações é feito pelo algoritmo de *DTW*, que calcula a distância entre os dados. DTW é uma boa opção para esse tipo de problema pois ele serve para comparar entradas de tamanhos diferentes. Uma vez que ele compara as deformações entre as duas entradas.

Uma descrição do algoritmo pode ser visto no seguinte pseudo código:

1. Compute a distância de todas as características dentre as assinaturas de referência (as

obtidas no processo de cadastro) para gerar o *threshold*.

$$threshold_{\text{característica}_k} = \min_{R_i \in R, R_j \in R_j, i \neq j} D_{carac_k}(R_i, R_j)$$

no qual $threshold_{\text{característica}_k}$ é a distância para a característica k, k = 1, 2, 3..., $N_{\text{características}}$ ($N_{\text{características}}$ = 14 nesse trabalho, descritas em 3.1), R_i e R_j são duas assinaturas do conjunto de assinaturas válidas R armazenada no banco de dados. i, j = 1, 2, 3 ... N_{Ref} (N_{Ref} é a quantidade de assinaturas do conjunto de treinamento)

2. Compute a distância de todas as característica da assinatura que está sendo validada com as assinaturas de referência para obter a dissimilaridade para cada uma das características. O valor da dissimilaridade daquela característica será a menor distância para alguma das assinaturas de referência.

$$Dissimilaridade_{\text{característica}_k} = \min_{R_i \in R} D_{carac_k}(R_i, T)$$

no qual $Dissimilaridade_{\text{característica}_k}$ representa o valor da dissimilaridade da característica k entre a assinatura que está sendo testada T e as assinaturas de referência, k = 1, 2, 3, ..., $N_{\text{características}}$, R_i é uma assinatura do conjunto de assinaturas válidas R armazenada no banco de dados. i = 1, 2, 3, ... N_{Ref}

3. Compare o valor de dissimilaridade com o valor da distância média multiplicado por um fator de tolerância (F_{tol}), se o valor for menor, essa característica entra para as características aprovadas.

Algoritmo 1 Calcular a Quantidade de características aprovadas

```
1: função QUANTIDADE DE CARACTERISTICAS VALIDAS
2:     N_validas = 0
3:     para k = 1 até N_características faça
4:         se Dissimilaridade_característica_k < F_tol × threshold_característica_k então
5:             N_validas = N_validas + 1
6:         fim se
7:     fim para
8:     retorna N_validas
9: fim função
```

4. A decisão desse algoritmo, se utilizada sozinha, aceita caso mais da metade de todas as características for aprovada, rejeita caso contrário. Caso esse algoritmo seja utilizada junto da OCSVM, o mesmo critério é utilizado, porém, com o acréscimo no número de características.

Algoritmo 2 Decisão final do algoritmo

```
1: função DECISÃO FINAL(N_validas)
2:     se N_válidas > N_características/2 então
3:         retorna "Válida"
4:     senão
5:         retorna "Não Válida"
6:     fim se
7: fim função
```

O valor do fator de tolerância (F_{tol}) referenciado no terceiro passo do Algoritmo 1, é algo determinante para o resultado desse algoritmo. Uma vez que com o acréscimo dele, aumenta a área de aceitação das assinaturas, abrindo espaço para um maior número de aceitação de assinaturas que deveriam ser rejeitadas. E caso o valor dele seja muito baixo, o número de assinaturas válidas rejeitada tende a aumentar.

Para o cálculo desse valor, duas abordagens podem ser utilizadas, ele pode ser dependente do usuário, ou seja, um valor diferente para cada um dos usuários, ou independente, um valor comum para todas as validações.

Porém, devido a natureza do problema, não vamos ter um conjunto de treinamento muito grande na maioria dos casos, pois a princípio somente a assinatura capturada no momento do cadastro estará disponível. Então foi decidido utilizar uma abordagem independente do usuário, em que se utiliza um único fator de tolerância para todos os usuários do sistema. Para o cálculo desse valor, podemos utilizar as assinaturas de um banco de dados com o propósito do refinamento do valor do fator de tolerância, uma vez que ele contém um bom número de assinaturas.

Então, o fator de tolerância foi calculado a partir do banco de dados do SigComp do ICDAR 2011[LMvdH$^+$11]. Esse banco de dados contém assinaturas de 10 usuários, com assinaturas originais e falsificadas de cada um deles.

Para avaliar o desempenho do fator de tolerância, é utilizado o *Equal Rejection Rate* (EER). O EER é o valor em que a *False Rejection Rate*(FRR) é igual ao *False Acceptance Error*(FAR). FRR é a porcentagem de assinaturas válidas que são incorretamente classificadas. FAR é a porcentagem de assinaturas falsas que são incorretamente classificadas. Inspirado nos valores descritos no artigo Chen et al. (2016), o valor do fator de tolerância foi calibrado entre o intervalo de valores de 0.8 e 2.0 com taxa de acréscimo de 0.01 [CXL16].Esse cálculo foi executado com vários tamanhos do conjunto de treinamento. Os tamanhos do conjunto de treinamento foram 5, 6, 8 e 10. Podemos verificar na figura 3.4 como os valores das taxas *FAR* e *FRR* variam de acordo com a mudança do fatores de tolerância.

Levando em conta que o tamanho do conjunto de treinamento está diretamente ligado a taxa de acerto/rejeição desse sistema, foi decidido manter o tamanho do conjunto de treinamento desse trabalho como 8, uma vez que não foi observado uma redução significativa quando comparados 8 e 10 assinaturas. Consequentemente, temos que o fator de tolerância escolhido foi 1.501.

O valor do fator de tolerância escolhido por Chen et al.(2016) foi o de 1.32 [CXL16]. A diferença entre os valores se dá devido principalmente a alguns motivos, um deles foi que

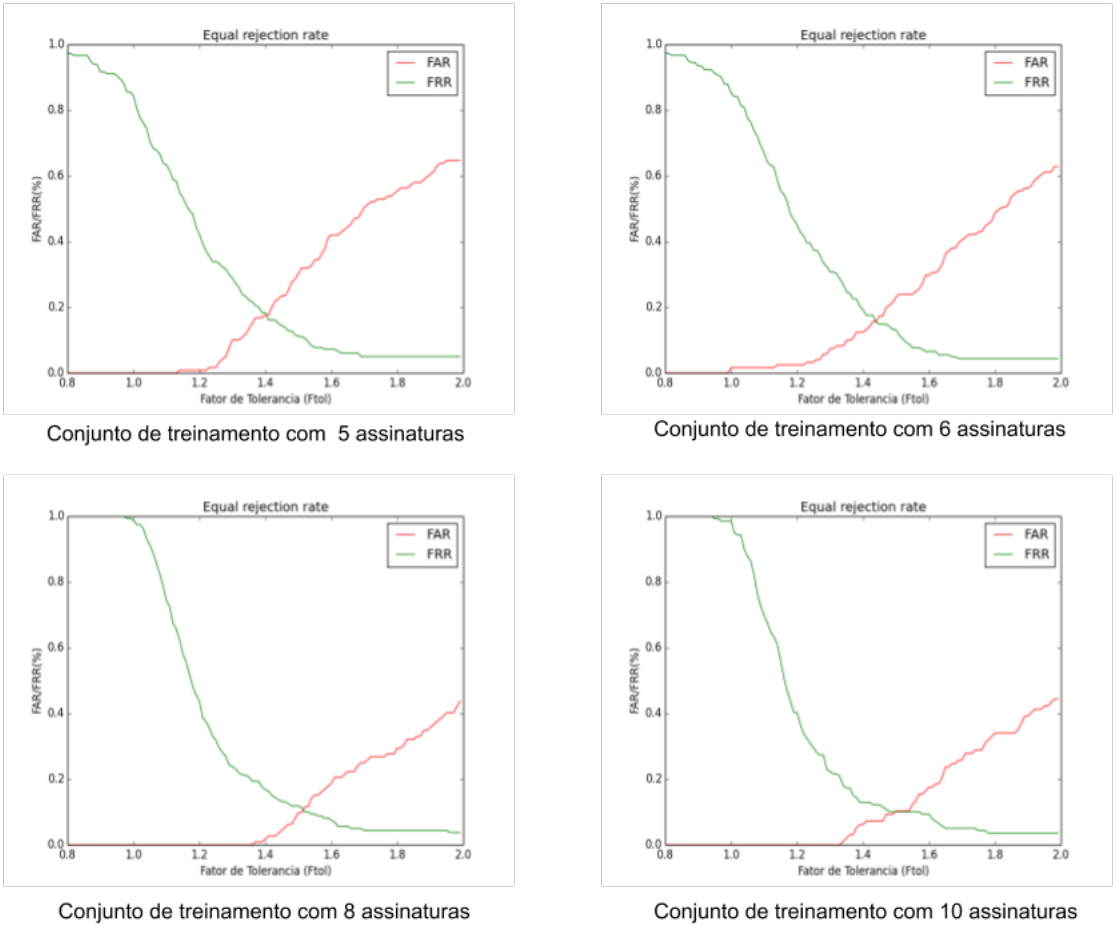


Figura 3.4 Gráfico do resultado da taxa de rejeição utilizando o banco de dados SigComp

o banco de dados escolhido para a geração desse valor foi diferente nos dois trabalhos, e a segunda razão é que existe uma diferença entre as características utilizadas, esse trabalho não utiliza a pressão no validador por limitação de hardware. Outra diferença é justamente no tamanho do conjunto de treinamento, Chen et al.(2014). utiliza o conjunto de treinamento de tamanho 5.

3.4.1.2 OC-SVM

Esse algoritmo foi inspirado no artigo de Guerbai et al. (2015), com algumas alterações. Uma das operações realizadas foram na escolha das características utilizados, uma vez que o artigo original utilizava reconhecimento de assinatura *off-line*. As características então, foram inspiradas no trabalho de Heinen et al. (2003), e estão descritas na lista abaixo:

- **Tempo total de assinatura** Essa característica se refere ao tempo total que o usuário levou para assinar. Essa variável não é tempo corrido, ela se refere a tempo útil assinado.

Ou seja, se o usuário levantar a caneta do *tablet*, esse tempo é descartado até ele encostar a caneta novamente no tablete.

- **Número de vezes que a caneta foi levantada do *tablet*** Essa característica se refere a quantidade de vezes que o usuário levanta a caneta durante uma assinatura
- **Velocidade média** Essa características representa a velocidade média em pixels/segundo que o usuário levou para desenhar toda a assinatura. Esse valor é calculado a partir de todas as velocidades capturadas. Entende-se por velocidade nesse contexto a distância que o usuário leva do momento que repousa a caneta até o momento que ele levanta. Ou seja, esse valor é uma média aritmética de todos os valores de velocidade obtidos.
- **Velocidade máxima** Essa características representa a maior velocidade em pixels/segundo que o usuário levou para desenhar toda a assinatura.
- **Trocas de sentido nos eixos x/y** Esse dado representa a troca de sentido em uma assinatura. Entende-se por troca de sentido quando um eixo, seja x ou y, ele está indo em uma direção do quadrante, e ele passa a ir em outra direção do quadrante.
- **Densidade da assinatura** A densidade da assinatura é feita separando toda a área útil que pode ser assinada em um retângulo, e então é convertida em um vetor de 10x5 proporcionalmente, ou seja, o primeiro quadrante representa a toda a área de 0x0 até (largura/10)/(altura/5). A cada ponto da assinatura que esteja dentro da área determinada quando convertido, a densidade aumenta nessa parte do vetor de densidade. Uma vez feito isso com todos os pontos, podemos então calcular a densidade em relação a quantidade de pontos totais.
- **Interseção** São marcados alguns traços na área útil, e então verifica-se o momento em que as assinaturas cruzam essas linhas imaginárias [GCH15].

Esse algoritmo, que só depende de informações verdadeiras, não precisando de falsificações para seu treinamento, será dependente do usuário, ou seja, cada usuário possuirá seu próprio *OC-SVM*. As características das assinaturas genuínas são armazenadas no banco de dados, e no momento do teste essas informações são utilizadas para o treinamento de uma *OC-SVM* que será utilizada para calcular se a assinatura que está sendo testada é válida para aquele usuário.

O algoritmo de decisão utilizado é semelhante com o escolhido para a *DTW*, ou seja, é considerada válida caso a quantidade de características aceitas seja maior que a metade do total de características.

3.4.1.3 Combinação entre os algoritmos

O algoritmo de decisão final, que é responsável por unir o resultado do *OC-SVM* e do *DTW* funciona mesclando as características que foram aprovadas em cada um dos algoritmos separadamente. Após isso, é verificado se a quantidade de características válidas é maior que 50% das características totais. O pseudo-código da validação pode ser encontrado em 3.4.1.3.

Algoritmo 3 Decisão final da validação

```
1: função DECISÃO FINAL(N_validas_DTW, N_validas_OCSVM)
2:     se N_validas_DTW + N_validas_OCSVM > N_características totais/2 então
3:         retorna "Válida"
4:     senão
5:         retorna "Não Válida"
6:     fim se
7: fim função
```

Em que, $N_{validas_{DTW}}$ é o número de características da assinatura em validação que o algoritmo de *DTW* classificou como válida, $N_{validas_{OCSVM}}$ é o número de características em validação que o *OC-SVM* classificou como válidas e $N_{\text{características totais}}$ é a quantidade de características presentes nos algoritmos de DTW e OC-SVM.

3.4.2 Servidor

Nessa sub-sessão, serão descritos os detalhes de implementação do Servidor.

A escolha da tecnologia utilizada do lado do servidor foi *Django*. *Django* é um *framework* de *Python Web*, que foca em um desenvolvimento rápido e limpo. Um dos motivos da escolha desse *framework* para atuar no servidor, foi a linguagem que ele é baseada, que é a linguagem *Python*[7]. *Python* é uma linguagem de alto nível, com tipagem dinâmica, totalmente escalável e é orientada a objetos.

Além de possuir um ótimo acervo de bibliotecas e *frameworks* prontos para o uso, ela ainda mostra-se relativamente rápida levando em conta toda a sua abstração. *Python* passou a ser a segunda linguagem mais utilizada no *Github* no ano de 2017[8]. O lado do servidor ainda utiliza um programa a parte, também desenvolvido em *Python*, que são os algoritmos que utilizados no verificador.

No contexto *Django*, um *app* é uma aplicação web que tem por obrigação exercer um determinado serviço, por exemplo, um *app* pode fazer o *log* de um sistema, uma aplicação de uma votação, sistema de login, sistema de mensagens ou uma aplicação de acesso a uma base de dados. Ou seja, são módulos específicos que facilitam o reuso em outros projetos *Django*. Ao criar um projeto em Django, os arquivos, estrutura e diretório base são criados automaticamente pelo *kit* de ferramentas da linguagem.

Neste projeto, foi criado um *app* lida com as requisições, busca no banco de dados, validação e respostas para o cliente. Um projeto é composto por arquivos de configurações juntamente com um conjunto de *apps*. Isso torna o desenvolvimento mais modular e ágil. Assim a comunidade pode criar módulos, o que torna a ferramenta cada vez mais rica, uma vez que não é preciso ficar reinventando a roda para módulos genéricos.

Django também detém uma abstração no que diz a respeito do banco de dados. Para facilitar a integração com os banco de dados, o próprio *framework* já dá suporte nativamente

[7] https://www.python.org/

[8] https://octoverse.github.com/

aos principais tipos de bancos de dados comerciais, como por exemplo *Postgres*, *MySql*, *SQ-Lite*, entre outros. A abstração do banco de dados se dá pois existe uma opção de definir uma espécie de Classe em *Python*, que descreve todas as Entidades, os Atributos e os respectivos relacionamentos entre eles.

O próprio *Django* então transforma essa classe em um banco de dados de fato, criando todas as tabelas, relacionamentos e regras necessárias. Nesse projeto, optou-se pelo uso da abstração do banco de dados oferecida pelo *Django*, o banco escolhido foi o *SQLite* pela sua simplicidade e cobertura de todos os requisitos necessários por esse projeto.

A representação de uma assinatura no banco de dados é dada unicamente a partir da descrição textual de todas as características. Em outras palavras, uma assinatura é composta por campos contendo o identificador do usuário juntamente com cada uma das características necessárias (definidos em 3.1 e em 3.4.1.2) para o projeto. A imagem final da assinatura não é armazenada no banco de dados.

Um projeto *Django* pode ser descrito em poucas palavras como uma abstração de um Servidor Web, que por sua vez é uma aplicação que recebe e responde requisições do tipo HTTP. Seu uso mais comum é em páginas web, que programas chamados navegadores, exemplos de navegadores são o *Google Chrome* e *Mozilla Firefox*, fazem requisições *web* a servidores. Os navegadores ainda são responsáveis por receber a resposta do servidor e exibir o resultado da requisição, que na maioria dos casos, contém textos, imagens, links e ainda pode conter outras mídias digitais como vídeo. Dado a natureza desse projeto, o protocolo foi utilizado unicamente para mandar e receber informações em forma de texto através da web entre o servidor e o aplicativo.

Como o validador também foi desenvolvido em *Python*, a integração entre o *Django* e os arquivos que contém a implementação do validador foi feita de maneira nativa.

Como requisito adicional, o servidor e os clientes terão que estar conectados a uma rede local, para diminuir a complexidade, visto que não alterará o propósito nem os resultados.

Ao receber uma mensagem do tipo *POST* no endereço de cadastro, o programa primeiramente checa se o número do Cadastro de pessoa Física(*CPF*) já está cadastrado, e em caso negativo, responde utilizando uma exceção descrevendo o erro que ocorreu. Se for um usuário novo, o servidor então extrai todas as informações das assinaturas que já estão descritas na requisição e registra elas juntamente com o usuário no banco de dados. Já no caso do servidor receber uma mensagem no endereço de validação com os dados de uma assinatura e um usuário, as outras assinaturas são carregadas do banco de dados para o treinamento do validador e então a assinatura é validada e a resposta é encaminhada para o solicitante.

3.4.3 Aplicativo

Nessa sub-seção, serão descritos os detalhes de implementação do Aplicativo.

O *Android* é um sistema operacional mobile *open source* de desenvolvimento fechado, que teve seu primeiro celular lançado em 2008 e que hoje já conta com mais de 2 milhões de aparelhos ativos por mês, segundo a *Google*, sua principal desenvolvedora.[9] Esse é um dos

[9]http://odia.ig.com.br/mundoeciencia/2017-05-17/android-ultrapassa-marca-de-2-bilhoes-de-usuarios-ativos-por-mes.html

motivos para a escolha desse sistema operacional e ambiente de desenvolvimento.

O aplicativo é a parte desse software mais próxima ao usuário, isso é, é a parte em que as informações de cadastro e validação são coletadas e enviadas para o servidor para serem avaliados. A aplicação, como mencionado anteriormente, foi desenvolvida para a plataforma *Android*, tendo como requisito mínimo a versão da API 23 (equivalente ao *Android* versão 6.0), atualmente o *Android* se encontra na Versão 8.0 (*API level 27*).

O sistema foi desenvolvido na plataforma oficial de desenvolvimento oficial do *Android*, o *Android Studio* [10]. O *Android Studio* reúne todos os elementos e softwares que são necessários para começar a desenvolver uma aplicação que rode em seu sistema operacional.

O aplicativo é responsável por todas as etapas de extração, envio dessas informações para o servidor. Também será responsabilidade do aplicativo mostrar as respostas do servidor para o usuário final. O aplicativo foi desenvolvido e testado utilizando um *Tablet Samsung Galaxy Tab A* rodando o *Android 7.1.1 (Nougat)*.

O processo de cadastro funciona com uma tela para o usuário entrar com o id, senha e um espaço para a assinatura. Uma vez terminada a assinatura atual, o usuário deve então clicar para capturar a nova assinatura, antes do campo de assinatura ser limpo, as características da assinatura atual são extraídas e salvas localmente e então o campo é limpo para que uma nova assinatura seja capturada. Esse processo é repetido até capturar todas as assinaturas de referência necessárias, cada uma referenciando uma assinatura. Como último passo, os dados são enviadas para ao servidor para que o cadastro seja realizado. A imagem da tela de cadastro pode ser conferida na figura 3.5.

O procedimento do envio de requisição de validação é semelhante ao de cadastro, com a diferença que somente uma assinatura é coletada, que é a assinatura teste. O aplicativo também é responsável por notificar os resultados da validação. A imagem da tela principal do aplicativo contendo uma assinatura e a notificação contendo o resultado da assinatura pode ser vista na figura 3.6

A conexão com o servidor é feita através da biblioteca *Volley*.[11] Essa biblioteca tem algumas características que que são interessantes para o nosso projeto, e que acabaram levando a sua escolha para tratar da parte de transmissão de dados entre os dispositivos utilizados no projeto. *Volley* é uma biblioteca de requisições HTTP para *Android* que torna acessos à internet mais fácil e rápido. Utilizamos a classe *StringRequest* do pacote *Volley*, temos a garantia de alguns benefícios em relação a outras bibliotecas de redes, uma delas é que todas as requisições web são automaticamente organizadas, isso quer dizer que existe um esforço do ponto de vista do sistema operacional para organizar e estruturar todas as requisições que utilizam a rede, outra melhoria é a facilidade na customização das requisições, sendo possível adicionar funções de retorno facilmente para que a resposta do servidor seja tratada corretamente.

[10] https://developer.android.com/studio/index.html

[11] https://developer.android.com/training/volley/index.html

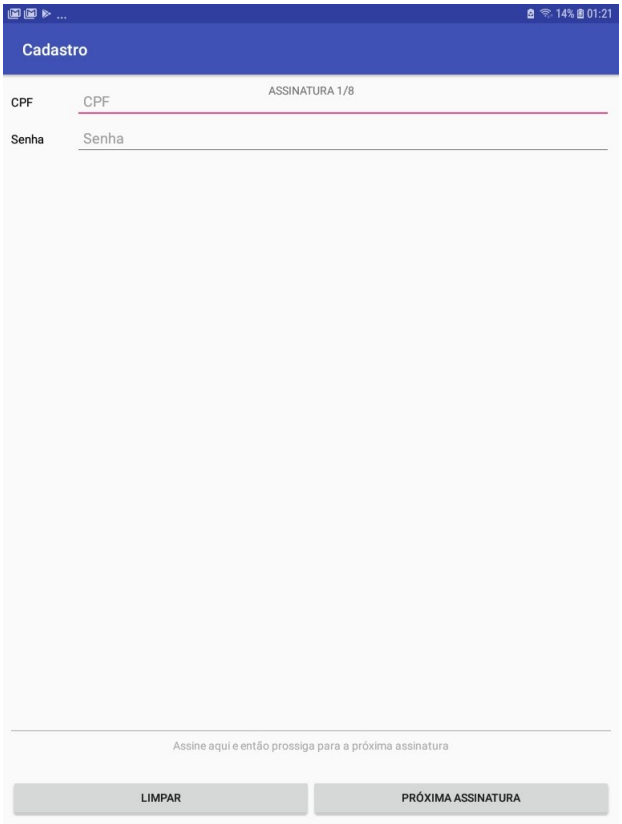
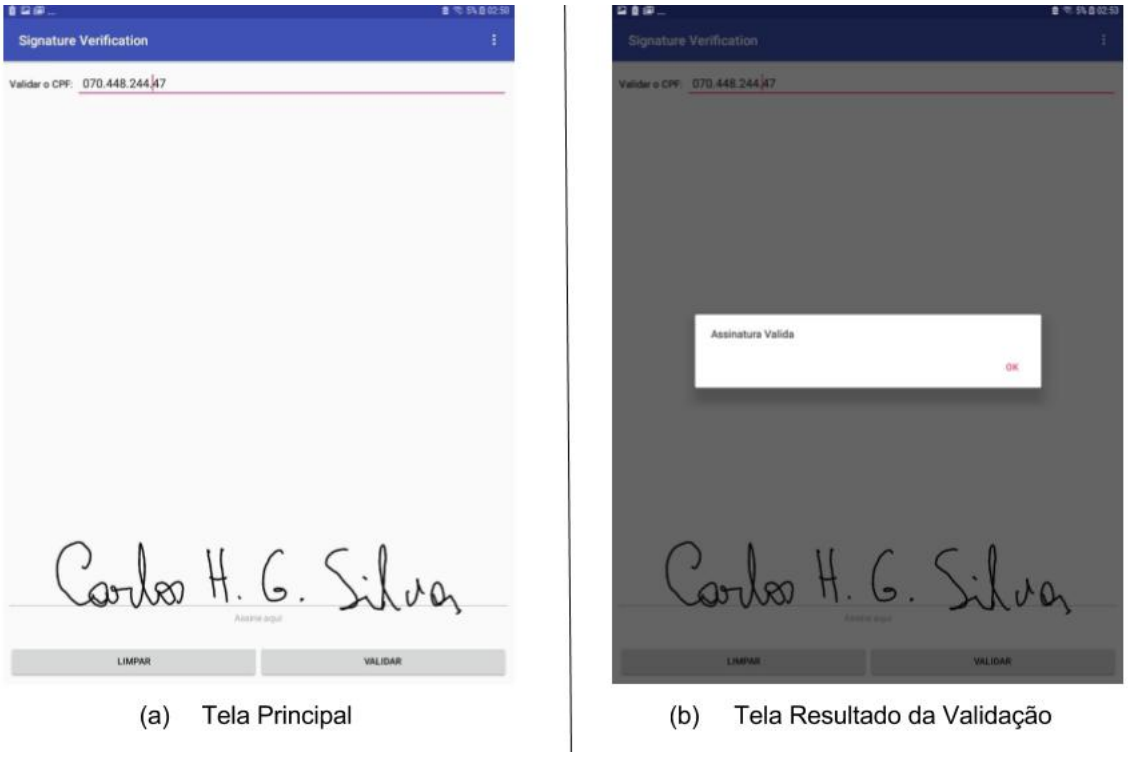


Figura 3.5 Tela do cadastro do sistema



(a) Tela Principal

(b) Tela Resultado da Validação

Figura 3.6 Tela principal e de validação

CAPÍTULO 4

Experimentos

Nesse capítulo serão descritos os resultados obtidos por esse trabalho. O projeto como um todo foi validado utilizando os bancos de dados da *SigComp* e o banco de dados criado para essa finalidade.

4.1 Taxa de acerto variando o tamanho do conjunto de treinamento

Como visto no capítulo 3, o tamanho do conjunto de treinamento é um fator importante para a eficiência geral desse algoritmo. O conjunto de treinamento é obtido inicialmente no momento do cadastro do usuário no sistema, e então é utilizado como referência para validar as futuras consultas.

O valor do tamanho do conjunto do treinamento utilizado nesse trabalho foi obtido após a validação com parte da base de dados da *Sigcomp* [LMvdH$^+$11]. Esse banco de dados conta com assinaturas Holandesas e Chinesas, capturadas utilizando métodos *offline* e *online*. Apenas as assinaturas Holandesas que foram capturadas utilizando o método *online* foram utilizadas na validação.

A validação ocorreu analisando a eficiência do algoritmo utilizando o mesmo *dataset*, porém com diferentes tamanhos de conjunto de treinamento. Os resultados dessa validação se encontram na figura 4.1.

Constatou-se que a taxa de acerto total do sistema tende a aumentar uma vez que o tamanho do conjunto de treinamento aumente. Isso se deve a maior generalização que ocorre quando se aumenta o número de assinaturas. Com 10 assinaturas no conjunto de treinamento, o nosso algoritmo teve precisão de 90.4%.

Analisando o resultado obtido pelo nosso algoritmo com os resultados dos algoritmos que participaram da competição cujo banco de dados é originalizado, pode-se constatar que a eficiência está dentre a média do esperado para o estado da arte. Na competição, os algoritmos de classificação *online* utilizando a base de dados Holandesa obtiveram uma taxa de precisão entre 88% e 96%. Na competição, era permitido que eles utilizassem um conjunto de treinamento de até 12 assinaturas [LMvdH$^+$11].

Entretanto, o nosso resultado não pode ser comparado diretamente porque o resultado obtido na competição foi obtido a partir de duas bases de dados, uma delas exclusiva para o treinamento e outra exclusiva para testes. A nossa validação dividiu o conjunto de treinamento utilizado na competição entre testes e treinamento. Porém, é um bom indicativo da qualidade geral do algoritmo.

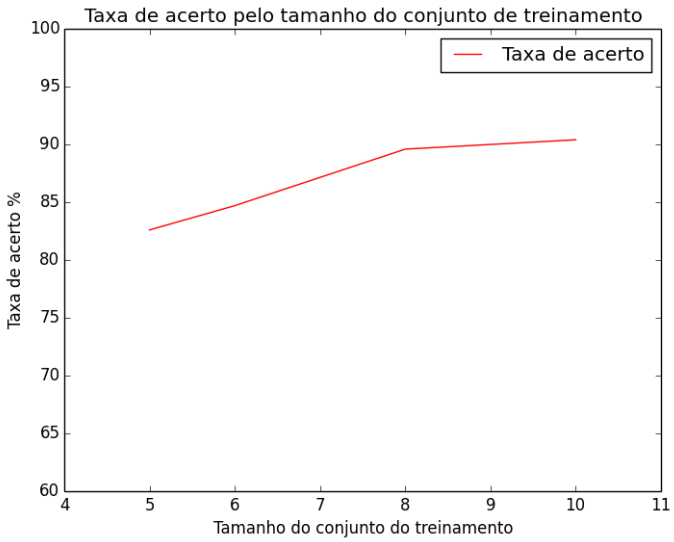


Figura 4.1 Relação do tamanho do conjunto de treinamento por taxa de acerto

4.2 Eficiência geral do sistema

Uma vez que não foi encontrado nenhum banco de dados que contemple todas as características utilizadas pelo verificador utilizado por esse trabalho, um banco de dados foi montado. Esse banco de dados tem o propósito de verificar a extração de característica e o algoritmo de verificação.

Esse banco de dados foi obtido utilizando o próprio software de captura de assinaturas desenvolvido nesse projeto. De cada assinatura, foram capturados todas as *features* que são utilizadas no algoritmo de verificação. Essa base de dados foi capturada em dois experimentos isolados, totalizando a participação de 17 participantes.

Na figura 4.2 podemos ver assinaturas válidas (a) e assinaturas inválidas (b) capturadas nesse experimento.

4.2.1 Coleta de dados

Foram realizados dois experimentos para a coleta de dados para a base de dados. Na primeira amostra, tivemos 6 participantes e 11 na segunda.

Em ambos os experimentos, os participantes se cadastraram no sistema, o que requer um total de 8 assinaturas para o treinamento do sistema. Em seguida, cada um deles assinou mais algumas vezes, sendo na primeiro experimento 6 vezes e 15 no segundo, para validar o algoritmo com assinaturas verdadeiras.

A assinatura de cada participante foi exposta aos demais e a estes foram solicitados que tentassem imitar as assinaturas alheias, a fim de validar a precisão do sistema ao distinguir as assinaturas verdadeiras das falsas.

(a) Assinaturas Verdadeiras (b) Assinaturas Falsas

Figura 4.2 Imagem com assinaturas válidas e falsificadas obtidas durante a construção do banco de dados

Os valores estão descritos na tabela 4.1.

Do total de assinaturas obtidas no levantamento de dados, temos que o sistema classificou corretamente 383 amostras. A taxa de acerto, baseado nessa base de dados é de 96%. Dos 15 classificados incorretamente, 10 são falsos negativos, ou seja, o sistema classificou como cópias mas deveria ter classificado como originais. Enquanto que 5 delas são falsos positivos, que são o caso em que o sistema classificou como originais mas são cópias.

Essa taxa de falsos positivo/falso negativo pode ser modificada ajustando o número mínimo de *features* para a assinatura ser considerada verdadeira. O valor padrão que está no algoritmo é metade das *features*, ou seja, se metade ou mais das características forem reconhecidas, o sistema aceita a assinatura como válida. Para sistemas mais críticos, esse valor pode ser alterado. Uma vez que é mais custoso deixar um usuário autenticado incorretamente entrar em um sistema do que pedir que o usuário credenciado assine duas vezes.

4.3 Análise visual das falhas

Para melhor entendimento sobre as falhas do sistema, iremos analisar algumas imagens finais de assinaturas incorretamente classificadas pelo sistema. Primeiro iremos analisar dois falsos negativos e então iremos analisar dois falsos positivos. Para proteger a privacidade dos usuários, algumas caixas em azul foram adicionadas para esconder alguns detalhes das assinaturas dos participantes.

Tabela 4.1 Base de dados de Teste

	Amostra #1	Amostra #2	Total
Número de participantes	6	11	17
Número de Homens	6	3	9
Número de Mulheres	0	8	8
Participantes com faixa etária entre 10-20	0	3	3
Participantes com faixa etária entre 20-30	5	2	7
Participantes com faixa etária entre 30-40	1	2	3
Participantes com faixa etária entre 40-50	0	4	4
Assinaturas válidas	30	165	195
Assinaturas copiadas para teste	38	165	203
Número de assinaturas coletadas	68	330	398

4.3.1 Falsos negativos

As imagens de duas assinaturas que são falsos negativos estão dispostas ao lado de algumas assinaturas de referência na figura 4.3 e 4.4. Iremos agora analisar para tentar perceber quais foram as características que levaram a falsa classificação.

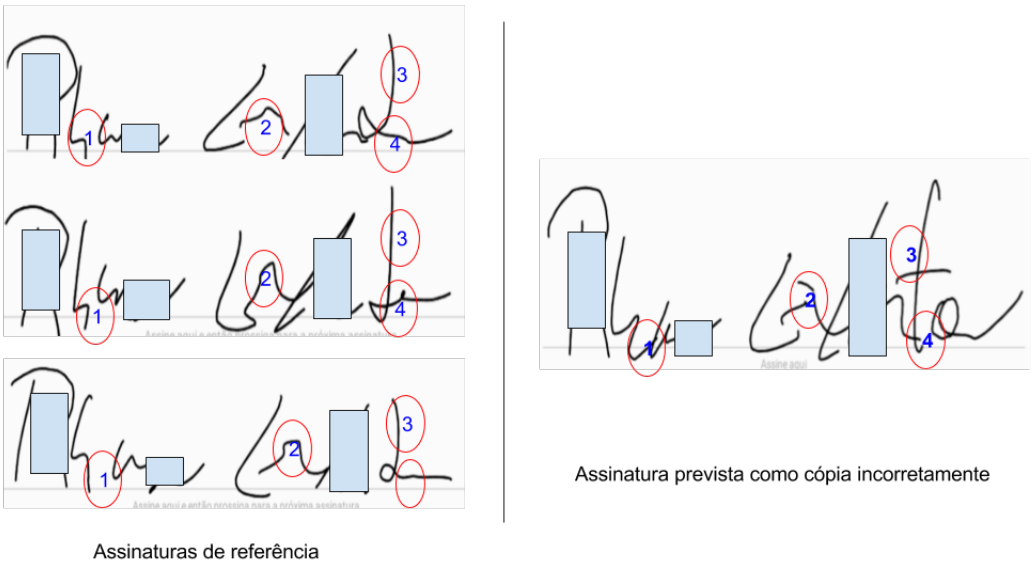


Figura 4.3 Falsos negativo ao lado de assinaturas de referência #1

Na figura 4.3, podemos destacar quatro detalhes que podem ter levado a classificação incorreta por parte da nossa solução. No detalhe realçado com 1 na imagem, nas assinaturas de referência há uma descontinuidade, ou seja, houve um levantamento do *tablet* e então a continuação da assinatura, já na classificada incorretamente, esse traço é contínuo. O contrário

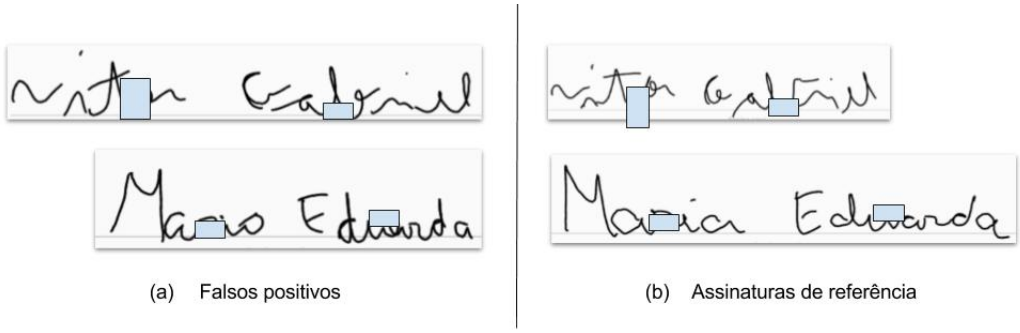
ocorre no detalhe 2, onde é contínuo nas referências e discreto na que está sendo testada. O item 3, temos que o traço do t não existe nas assinaturas de referência. Já o item 4, a assinatura teve acréscimo de um arredondamento.

Figura 4.4 Falsos negativo ao lado de assinaturas de referência #2

Na imagem 4.4, podemos destacar dois detalhes que podem ter levado a classificação incorreta. No detalhe realçado com 1 na imagem, temos que a curva teve uma deformação, o que pode ter sido por conta da má captura do *tablet*. No detalhe marcado com 2, temos que na assinatura incorretamente classificada a assinatura começa com uma 'entrada' a mais, nas assinaturas de referência, ela começa na parte da curva.

4.3.2 Falso positivo

Iremos analisar a versão final de dois falsos positivos. Podemos observar na figura 4.5 podemos ver que as cópias são bem parecidas. Podemos ver na imagem que os detalhes principais das assinaturas originais estão representados nas cópias. Podemos perceber que ambas as assinaturas não contemplam o nome completo dos integrantes, o que pode ter ajudado na generalização da assinatura.



(a) Falsos positivos (b) Assinaturas de referência

Figura 4.5 Falsos positivos ao lado de assinaturas de referência

CAPÍTULO 5

Conclusão

Nesse capítulo, são mostrados os resultados e conclusões obtidos por esse trabalho. Primeiro é mostrado as contribuições e conclusões do sistema, então é mostrado as dificuldades encontradas no decorrer desse projeto e por fim os trabalhos futuros.

5.1 Contribuições

Nessa seção estão descritas as contribuições desse trabalho, fazendo um paralelo com a seção de objetivos do primeiro capítulo.

1. **Construção de um aplicativo para dispositivos móveis que capture e extraia informações de uma assinatura**

   Esse objetivo foi concluído utilizando a biblioteca *android-signaturepad* e que funciona tanto com assinaturas feitas com o dedo como assinaturas feitas com a caneta *stylus*. A partir da biblioteca do *android-signaturepad*, podemos extrair as características necessárias para a validação.

2. **Construção de um aplicativo para dispositivos móveis que capture e extraia informações de uma assinatura**

   Foi implementado um verificador de assinaturas baseado dois verificadores independentes, o DTW e o OC-SVM. Por ser baseado em dois classificadores independentes, eles trabalham em conjunto para gerar um algoritmo mais robusto. A combinação entre eles, é feita ao fim da execução de cada um deles separadamente. As características que são aceitas em cada um dos dois são somadas e então é comparado com a quantidade de características totais.

3. **Implementação de um servidor para que receba as características de uma assinatura e utilize esses dados como entrada para o verificador classificar a assinatura como original ou cópia**

   Foi desenvolvido um servidor utilizando *Django* que, através dele, é possível cadastrar novos usuários no sistema e validar as assinaturas dos usuários já cadastrados. A validação ocorre através do Verificador que foi desenvolvido nesse projeto e o cadastro é feito através das assinaturas verdadeiras capturadas pelo aplicativo. Esse mesmo servidor é escalável para funcionar com vários *devices* diferentes, devido a escolha das tecnologias envolvidas.

4. **Validação e testes usando a solução**

   O sistema como um todo obteve uma taxa de acerto média de 96% no banco de dados criado a partir do próprio sistema, em que foi testado tanto a etapa de pré-processamento e extração de características como o algoritmo de verificação em si. O sistema final ainda conta com um servidor no qual serve para centralizar informações para que seja acessado de vários dispositivos.

   Outra validação realizada foi utilizando o banco de dados da *SigComp*, umas das principais competições da área, a taxa de acerto máxima foi de 90.6%, o que mostrou um resultado coerente quando comparado com os resultados dos algoritmos que participaram da competição.

5.2 Conclusão

A adoção de assinaturas para garantir consentimento e concordância é uma prática que é executada no mundo todo e por órgãos oficiais. Esse trabalho tinha como objetivo demonstrar a possibilidade de validar assinaturas a partir de um dispositivo móvel. Para atingir o nosso objetivo, foi desenvolvido, como mostrado na subseção anterior, um sistema que permita a captura e a validação da assinatura. Os testes realizados nesse projeto mostram que em 96% dos casos o algoritmo conseguiu verificar de maneira correta a autenticidade da assinatura.

O algoritmo utilizado neste trabalho é baseado na união de dois algoritmos, o DTW e o OC-SVM. Eles se complementam haja vista que o DTW é mais utilizado para séries contínuas. Por outro lado, o OC-SVM é mais utilizado para valores discretos. A união dos dois algoritmos é obtida por meio da soma das características aprovadas por cada algoritmo executado de forma independente, e então, esse valor é comparado com a quantidade total de características analisadas pelos dois algoritmos. Se o número de características aprovadas for maior do que metade de características totais, o algoritmo aceita a assinatura como válida.

Todas as partes do sistema, o aplicativo, o servidor e o verificador foram desenvolvidas de maneira modular. O que facilita o desenvolvimento de novas partes para ampliação do projeto, como um *app* para um sistema operacional diferente, por exemplo.

5.3 Dificuldades

Várias dificuldades surgiram durante a escrita e desenvolvimento desse Trabalho de Graduação. Existiram dificuldades tanto da gestão e organização do tempo, como dificuldades de hardware e de software.

Como parte das dificuldades sofridas em relação ao hardware, podemos citar que o *tablet* principal usado no desenvolvimento, o (*Galaxy Tab A*), não suporta a captura de pressão, o que impossibilitou o uso desse dado como uma características nos verificadores utilizados. E pelo grande número de artigos que utilizam essa informação, podemos assumir que essa adição traria acréscimo a taxa de acerto.

Como dito anteriormente, não foi encontrado na literatura um banco de dados que tivesse

todas as características utilizadas pelo validador do sistema, o que impossibilitou uma comparação mais embasada da performance do nosso algoritmo.

A falta de familiaridade com alguns aspectos das tecnologias utilizadas foram responsáveis por alguns atrasos com problemas simples, como sintaxe ou alguma configuração inicial.

A dificuldade principal em relação ao tempo ocorreu principalmente pela dificuldade de medir a quantidade de tempo que uma determinada tarefa necessita para ficar pronta. O que acarretou de mal planejamento do cronograma das atividades.

5.4 Trabalhos futuros

Como trabalhos futuros, seria interessante a implementação do projeto em outros sistemas operacionais, como o *IOs* [1] e o *Windows Phone* [2].

Nesse trabalho, o servidor utilizado foi um servidor local, devido a simplicidade de implementação. Esse servidor deve ser colocado em algum servidor externo para ser acessado através da internet, aumentando assim a escalabilidade da solução.

Pode ser estudado o impacto causado pela qualidade do hardware utilizado na captura das assinaturas. Como exemplo concreto, poderia ser validado o impacto que a característica da pressão iria causar na performance do validador. E se melhorasse a performance consideravelmente, alterar o escopo de dispositivos suportados para que o software só funcione em hardwares compatíveis.

Outro ponto de melhoria, seria criar um banco de dados maior, tanto no número de assinaturas quanto no de participantes. Assim contemplando uma validação mais abrangente do sistema.

Outra característica que seria interessante de ser validada, seria o uso de outras versões dos verificadores para comparar os resultados com os atuais. Uma sugestão seria os algoritmos que utilizam *deep learning*, como o de [FSSF15] que utiliza *deep learning* para selecionar as características.

[1] https://www.apple.com/br/ios/ios-11/

[2] https://www.microsoft.com/pt-br/store/apps/windows-phone

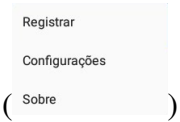
APÊNDICE A

Guia de usuário

O guia de usuário do sistema desenvolvido nesse projeto de graduação está descrito nesse apêndice. Cada uma das principais funções estão descritas abaixo.

A.1 Cadastro

Para a realização do cadastro no sistema, primeiro temos que abrir a tela de cadastro. Podemos fazer isso seguindo esses passos:

1. Clicar no menu representado pelos "três pontos"() no canto superior direito
2. Clicar no item do menu que tenha escrito "Registrar"no menu que foi criado após o passo anterior (

)

Após isso, o usuário será levado a uma tela em que o usuário precisa entrar com o CPF e a senha que será cadastrado no sistema.

Feito isso, o usuário deve assinar na linha indicada e clicar no botão "Próxima Assinatura", para que a assinatura que acabou de ser adicionada seja salva e o campo de assinatura seja limpo para que o usuário entre com a próxima assinatura. O usuário deverá repetir esse comportamento a quantidade de vezes do conjunto de treinamento. Podemos ver a tela que contém essa funcionalidade é mostrada na figura A.1.

Figura A.1 Tela do cadastro de sistema

Após o *input* de todas as assinaturas necessárias, o usuário deve então clicar no botão "Cadastrar"para que as assinaturas sejam mandadas para o servidor com o intuito de serem cadastradas. O aplicativo então mostra o status do cadastro através de um *pop-up*.

A.2 Validação

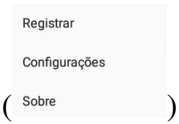
Para a validação de uma assinatura, o usuário deve estar na tela principal do aplicativo. Uma vez nessa tela, deve-se inserido o CPF e a assinatura e então clicar no Validar. A resposta da validação vai ser apresentada através de um *pop-up*. Podemos ver a tela que contém essa funcionalidade na imagem A.2

(a) Tela Principal (b) Tela Resultado da Validação

Figura A.2 Tela principal e de validação

A.3 Sobre

Para acessar a tela de Sobre deve-se utilizar o menu dos três pontos (⋮) e então clicar no item



do menu que tenha escrito "Sobre"( ) . Uma vez carregada essa tela, será descrito informações básicas sobre o software assim como as referências às bibliotecas. Podemos ver a tela que contém essa funcionalidade é mostrada na figura A.3.

Figura A.3 Tela de Sobre do sistema

A.4 Configuração

Para acessar a tela de Configuração, deve-se utilizar o menu dos três pontos e então clicar no botão que seja referente a Configuração. Uma vez carregada essa tela, as funções de *log verboso*, em que o sistema registra mais informações através do *logcat* e o *ip* do servidor estarão disponíveis para serem alterados. Podemos ver a tela que contém essa funcionalidade é mostrada na figura A.4.

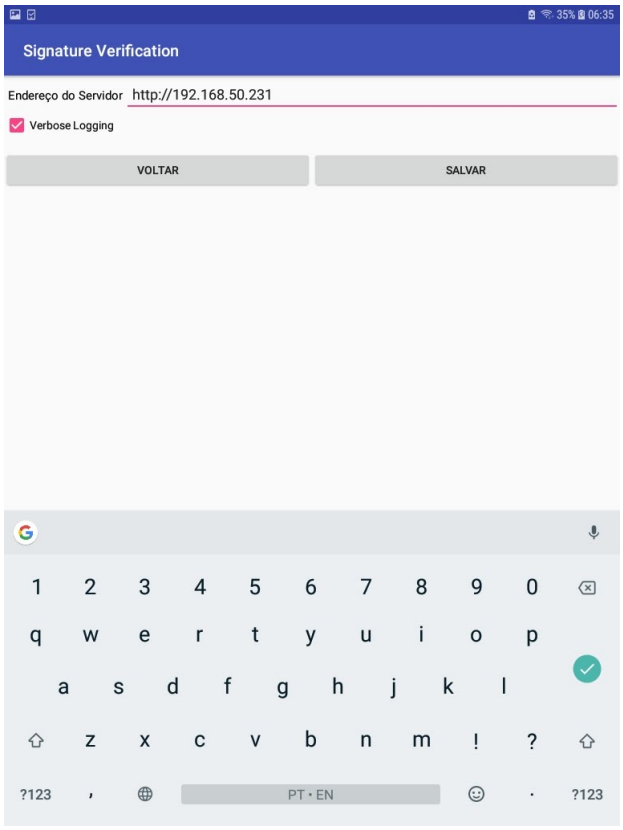


Figura A.4 Tela de Configuração do sistema

Referências Bibliográficas

[ASK15] M. Arora, H. Singh, and A. Kaur. Distance based verification techniques for online signature verification system. In *2015 2nd International Conference on Recent Advances in Engineering Computational Sciences (RAECS)*, pages 1–5, Dec 2015.

[CXL16] Zhili Chen, Xinghua Xia, and Fangjun Luan. Automatic online signature verification based on dynamic function features. In *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 964–968. IEEE, aug 2016.

[DHR15] Beatrice Drott and Thomas Hassan-Reza. On-line handwritten signature verification using machine learning techniques with a deep learning approach. Master's thesis, Lund University, 2015. Student Paper.

[FBFa15] Deivison Pinheiro Franco, Felipe Barboza, Mariana Pompeo Freitas, and Nágila Magalhães Cardoso Cardoso and. Uma ferramenta computacional forense para verificação de autenticidade de assinaturas manuscritas através de processamento digital de imagens e redes neurais artificiais. *Computer on the Beach*, page 10, 2015.

[FSSF15] Mohsen Fayyaz, Mohammad Hajizadeh Saffar, Mohammad Sabokrou, and Mahmood Fathy. Feature representation for online signature verification. *CoRR*, abs/1505.08153, 2015.

[GCH15] Yasmine Guerbai, Youcef Chibani, and Bilal Hadjadji. The effective use of the one-class svm classifier for handwritten signature verification based on writer-independent parameters. *Pattern Recognition*, 48(1):103 – 113, 2015.

[GS94] David Garlan and Mary Shaw. An introduction to software architecture. *CMU Software Engineering Institute Technical Report*, page 39, January 1994.

[Hel00] G. Held. *Server Management*. Best Practices. CRC Press, 2000.

[HO04] Milton Roberto HEINEN and Fernando S. Osório. Biometria comportamental: Pesquisa e desenvolvimento de um sistema de autenticação de usuários utilizando assinaturas manuscritas. *INFOCOMP (UFLA), UFLA - Univ. Federal de Lavras*, 3:32–37, 2004.

[HSO17] Luiz G. Hafemann, Robert Sabourin, and Luiz S. Oliveira. Learning features for offline handwritten signature verification using deep convolutional neural networks. *Pattern Recognition*, 70(Supplement C):163 – 176, 2017.

[KR05] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, Mar 2005.

[LMvdH+11] M. Liwicki, M. I. Malik, C. E. v. d. Heuvel, X. Chen, C. Berger, R. Stoel, M. Blumenstein, and B. Found. Signature verification competition for online and offline skilled forgeries (sigcomp2011). In *2011 International Conference on Document Analysis and Recognition*, pages 1480–1484, Sept 2011.

[MMDG14] Ram P. Krish Javier Galbally Marcos Martinez-Diaz, Julian Fierrez and Javier Galbally. Mobile signature verification: feature robustness and performance comparison. *IET Biometrics*, 3:267–277(10), December 2014.

[MMR+01] K. R. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, Mar 2001.

[NFM+12] Heber V. Nogueira, Arlindo R. Galvão Filho, Simone C. Martinez, Gustavo T. Laureano, and Clarimar J. Coelho. Verificação da autenticidade de assinaturas manuscritas utilizando o comportamento médio das características. *VIII Workshop de Visão Computacional*, 2012.

[PM92] S. K. Pal and S. Mitra. Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on Neural Networks*, 3(5):683–697, Sep 1992.

[SBM14] N. Sae-Bae and N. Memon. Online signature verification on mobile devices. *IEEE Transactions on Information Forensics and Security*, 9(6):933–947, June 2014.

[Sen09] Pavel V. Senin. Csdl technical report 09-08. Technical report, University of Hawaii, April 2009.

[SLF10] Amanda Monteiro Sizo, Adriano Del Pino Lino, and Eloi Luiz Favero. Uma proposta de Arquitetura de Software para Construçãoo e Integraçãoo de Ambientes Virtuais de Aprendizagem. *RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação*, pages 17 – 30, 12 2010.

[Som06] Ian Sommerville. *Engenhariade Software*. Pearson Education, 8 edition, 2006.

Este volume foi tipografado em LaTeX na classe UFPEThesis (www.cin.ufpe.br/~paguso/ufpethesis).