Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Implementação de um método para Reconhecimento On-line de Assinaturas

Augusto César Gonçalves de Azevedo

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Orientador
Prof. Dr. Pedro de Azevedo Berger

Brasília
2011

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Computação — Licenciatura

Coordenador: Prof. Dr. Homero Luiz Piccolo

Banca examinadora composta por:

Prof. Dr. Pedro de Azevedo Berger (Orientador) — CIC/UnB
Prof. Dr. Marcus Vinicius Lamar — CIC/UnB
Prof. Dr. Camilo Chang Dórea — CIC/UnB

CIP — Catalogação Internacional na Publicação

Azevedo, Augusto César Gonçalves de.

Implementação de um método para Reconhecimento On-line de Assinaturas / Augusto César Gonçalves de Azevedo. Brasília : UnB, 2011. 257 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2011.

1. biometria, 2. assinatura, 3. autenticação

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Implementação de um método para Reconhecimento On-line de Assinaturas

Augusto César Gonçalves de Azevedo

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Prof. Dr. Pedro de Azevedo Berger (Orientador)
CIC/UnB

Prof. Dr. Marcus Vinicius Lamar
CIC/UnB

Prof. Dr. Camilo Chang Dórea
CIC/UnB

Prof. Dr. Homero Luiz Piccolo
Coordenador do Curso de Computação — Licenciatura

Brasília, 09 de fevereiro de 2011

Dedicatória

Dedico a minha mãe Elaine, meu pai Roque e meu irmão Eduardo.

Agradecimentos

Agradeço em primeiro lugar pela oportunidade de realizar meu curso de graduação na UnB.

Agradeço também a meus professores que me mostraram o caminho, inspiraram, e cederam seu tempo e conhecimento.

Não posso deixar de agradecer a meus amigos e colegas, pelos bons momentos e por sua compreensão.

Acima de tudo, agradeço a meus pais pelo incentivo, apoio e dedicação, e ao meu irmão Eduardo, desbravador do LaTeX e do Gnuplot, que me ajudou a driblar seus pequenos detalhes sórdidos...

Em especial, agradeço à Luísa pelo carinho, amizade e incetivo.

Resumo

No universo dos sistemas informatizados, métodos de autenticação biométrica vem ganhando grande popularidade como alternativas confiáveis aos tradicionais sistemas baseados em senha. Diferentemente dessa forma de autenticação que é baseada na detenção de um determinado conhecimento, a metodologia utilizada para a verificação de identidade por meio da biometria baseia-se na análise de características físicas e comportamentais dos usuários, buscando identificar comportamentos e traços físicos que caracterizam sua individualidade. O uso de assinaturas está imbuído nas sociedades humanas há muitos séculos como instrumento para firmar contratos, acordos, tratados e realizar as mais diversas transações. Como forma de autenticação biométrica em sistema informatizados, ainda é pouco popular. Entretanto, sua sistemática baseia-se em um comportamento já adquirido e internalizado pelo ser humano, o que o torna ideal para certos tipos de verificação que exigem menor nível de segurança.

O foco deste trabalho está na implementação de um método para reconhecimento *online* de assinaturas. O que significa que utiliza-se de assinaturas capturadas por meio de mesas digitalizadoras sensíveis à pressão, que agregam maior quantidade de informações, conferindo maior especificidade a assinatura fornecendo mais subsídios para comparação, além de tornar mais difícil sua falsificação. Também são realizadas comparações entre os algoritmos DTW e *FastDTW* na realização do alinhamento de assinaturas, identificando vantagens e desvantagens de sua utilização e incorporando-os na implementação de uma aplicação para coleta e verificação de assinaturas.

Palavras-chave: biometria, assinatura, autenticação

Abstract

In a world of computerized systems, methods of biometric authentication are gaining great popularity as reliable alternatives to traditional password-based systems. Unlike this latter form of authentication, which is based on the possession of certain knowledge, the methodology used for identity verification through biometrics is based on the analysis of physical and behavioral characteristics of the users, seeking to identify behavior and physical traits that characterize their individuality. The use of signatures has been imbued in human societies for many centuries as an instrument to consolidate contracts, agreements and treaties, and to conduct various transactions. As a form of biometric authentication in computerized systems, it still isn't very popular. However, the biometric's systematics relies on a behavior already acquired and internalized by human beings, which makes it ideal for certain types of verification of lower security level.

The focus of this study is on implementing a method for recognizing on-line signatures. This means that it makes use of signatures captured by pressure-sensitive tablets, which adds more information and results in greater specificity to the signature, therefore providing more subsidies for comparison. In addition, it makes falsification a lot harder. This research also compares the algorithms DTW and FastDTW on the realization of the alignment of signatures, identifying advantages and disadvantages of their use and incorporating them in the implementation of an application for collecting and verifying signatures.

Keywords: biometry, signature, authentication

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com o aumento constante do volume de informação produzida, coletada e armazenada em mídias, cresce também a preocupação com a segurança destes dados no que se refere ao seu local de armazenamento, restrições e políticas de acesso. Questões relativas ao sigilo de informações sensíveis são alvo de investimentos substanciais de entidades governamentais, bem como de corporações e grandes empresas da iniciativa privada, cuja atividade fim depende do armazenamento e sigilo de determinadas informações, como é o caso das instituições financeiras.

Em se tratando de informações sigilosas, faz-se necessário restringir seu acesso apenas a determinado grupo de pessoas. Para então permitir que determinada pessoa acesse dados restritos, em um sistema informatizado, será preciso certificar-se de que a pessoa em questão é de fato quem afirma ser. Adicionalmente será necessário verificar se esta pessoa está autorizada a acessar os dados por ela solicitados para, somente então, disponibilizá-los. Os tradicionais sistemas baseados em senha, ainda hoje, são a forma mais utilizada de autenticação de usuários, devido a sua simplicidade e baixo custo de implementação. O princípio básico do funcionamento destes sistemas encontra-se na posse de determinado conhecimento. Para cada usuário deve-se atribuir um conjunto de caracteres (símbolos) que comporão uma senha. Presume-se que somente o usuário a quem a senha foi atribuída detém o conhecimento desta. Toda vez que houver necessidade de autenticação do usuário, este deverá informar sua senha conforme previamente estabelecido.

Desse modo, partindo do princípio de que cada par (usuário, senha) é conhecido somente pelo próprio Usuário em questão, se esta senha, quando solicitada, for informada corretamente, pode-se assumir que o Usuário é de fato a pessoa que afirma ser.

É sabido que existem diversos problemas com esse método de autenticação como o grau de fragilidade da senha e a guarda da mesma pelas pessoas que a detêm. A fragilidade ou força da senha está diretamente relacionados a sua complexidade (quantidade de símbolos combinados para compor a senha e tamanho do alfabeto utilizado) e, principalmente, com a capacidade das pessoas de manter sua senha em sigilo. Senhas muito complexas são difíceis de lembrar, enquanto que senhas demasiadamente simples são mais susceptíveis a quebra pelas técnicas de força bruta.

No cotidiano atual, sistemas exigem a guarda de senhas que podem ter as mais diferentes características, como por exemplo: senhas numéricas, comumente utilizadas em transações bancárias, senhas alfanuméricas usadas para autenticação de contas de e-mail e sistemas informatizados, e para acesso à computadores.

Guardar um grande número de senhas, além de tornar-se um inconveniente, poderá vir a ser um problema se não for feita de forma adequada. Idealmente uma pessoa deveria memorizar todas as suas senhas e deveria, também, ser a única a deter esse conhecimento. Qualquer outra pessoa somente deveria saber dessas senhas pela revelação explícita e espontânea do próprio usuário.

Na prática percebe-se que a maioria das pessoas tem dificuldade para memorizar e lembrar da grande quantidade de senhas que possuem, e recorrem à escrita, anotando-as em uma agenda ou pedaço de papel. Tal ação fragiliza a segurança dos sistemas pois torna possível que outras pessoas não autorizadas tenham acesso a essas senhas sem consentimento do proprietário. Outro complicador diz respeito ao requisito de confiabilidade[1] dos sistemas de senhas, que indica que a mesma seja trocada periodicamente. Essa orientação por não ser levada a prática com o devido rigor, faz com que aumente o índice de fragilidade.

Se anotar senhas aumenta a vulnerabilidade do sistema, criando a possibilidade de que outras pessoas tenham acesso a ele indevidamente, em contrapartida, pessoas tem dificuldade para memorizar um grande número de senhas, principalmente se forem mais complexas.

A partir de problemas como este pode-se perceber a necessidade de investimento em pesquisas para o desenvolvimento de formas alternativas para autenticação[2] de usuários em sistemas informatizados. Procurando-se métodos mais eficientes, que facilitem a interação do usuário com os sistemas, e principalmente, que sejam confiáveis e seguros.

Nesse contexto, sistemas de autenticação baseados em biometria vêm ganhando grande popularidade como alternativas mais confiáveis que os sistemas baseados em senha.

> **Biometria** Ciência da verificação de identidade baseada em características fisiológicas ou comportamentais [16].

Diferentemente da autenticação por senha que é baseada na posse de um determinado conhecimento, a metodologia utilizada para a verificação de identidade por meio da biometria baseia-se na análise de características físicas e comportamentais dos usuários, e busca identificar comportamentos e traços físicos que caracterizam sua individualidade.

São exemplos de comportamentos que podem ser utilizados para autenticação :

- fala;
- movimento dos olhos;
- jeito de andar;
- movimentação do mouse;
- padrões de escrita;

Como características físicas mais comumente utilizadas podemos exemplificar:

- padrões de íris;

[1] Qualidade ou estado daquele ou daquilo em que se pode confiar (dicionário Michaelis).

[2] Ato ou efeito de autenticar, reconhecer como próprio, verdadeiro ou legítimo (escrito ou documento particular) (dicionário Michaelis).

- impressões digitais;
- padrões de vasos sanguíneos das mãos e pés;

> Uma boa Biometria é Universal, Única, Permanente e Coletável [8].

Quanto mais única for a característica observada, menor será a possibilidade de dois usuários diferentes apresentarem esta característica de forma idêntica. Como descrito em [8], uma boa biometria é:

Universal
Sendo possível a observação desse aspecto em qualquer indivíduo de uma população.

Única
Não há dois indivíduos em uma população para os quais essa característica se apresente de forma idêntica.

Permanente
A característica não se altera ao longo do tempo.

Coletável
Deve ser possível medir/capturar dados que representem a característica observada para que possam ser comparados com dados de outras amostras.

Dentre as diversas técnicas de autenticação biométrica existentes, uma das que se destaca é a realizada pela comparação de impressões digitais. Nos últimos anos, os equipamentos necessários para coletar impressões digitais tornaram-se mais acessíveis, sofrendo consideráveis reduções de preço ou passaram a vir embutidas em equipamentos como computadores portáteis (*notebooks*) e até mesmo em telefones celulares.

O uso de impressões digitais como forma de autenticação foi publicado pela primeira vez em 1880, na revista Nature, pelo Dr. Henry Faulds, e aplicado na identificação de criminosos. Foi observado que as impressões digitais de duas pessoas sempre serão diferentes, mesmo entre gêmeos idênticos, e a probabilidade de encontrar duplicidade é de um em um bilhão de chances.

De modo geral, a impressão digital atende aos quatro requisitos citados em [8]. Além de tudo a coleta de amostras de impressão digital é simples. Hoje encontra-se equipamentos de baixo custo capazes de executar esta tarefa, tendo portanto, grande aceitação do público e bons níveis de confiabilidade.

1.1 Reconhecimento de Assinaturas

Outra forma de biometria, menos popular, foi alvo deste estudo: o reconhecimento de assinaturas. Essa biometria é baseada em aspectos comportamentais do usuário, e não em características físicas. Logo, a forma de assinar de determinada pessoa pode sofrer

variações ao longo do tempo em função da variação desses aspectos comportamentais. Além disso, assinaturas são mais fáceis de reproduzir que impressões digitais ou padrões de iris.

A vantagem desse tipo de sistema é a sua ampla difusão e aceitação pelo público, o que o torna ideal para certos tipos de verificação de menor nível de segurança. Assinaturas são utilizadas há várias décadas, para firmar acordos e compromissos por meio de contratos e tratados assinados pelas partes envolvidas. Essa biometria tem maior aceitação por ser baseada em um comportamento já adquirido e internalizado pelo ser humano.

O problema do reconhecimento de assinatura é classificado em duas modalidades de acordo com os dados disponíveis para a realização da verificação.

Verificação *off-line* (estática)
: Baseia-se apenas na imagem da assinatura, o que significa que temos acesso aos resultados do processo de escrita somente após estar completo; tem grande aplicação em sistemas bancários para processamento de cheques e autenticação de documentos.

Verificação *on-line* (dinâmica)
: Utiliza assinaturas capturadas por mesas digitalizadoras sensíveis à pressão, conhecidos como *tablets*. Neste caso, temos acesso aos sinais amostrados durante o processo de escrita. Além da forma da assinatura, estes dispositivos são capazes de capturar informações dinâmicas como: a velocidade com que foi realizada cada parte da assinatura, o tempo decorrido, a quantidade de pontos, a pressão de cada ponto, bem como movimentos onde não há contato da caneta com a superfície. Essas informações conferem maior especificidade a assinatura, fornecendo maior quantidade de subsídios para comparação entre assinaturas, e assim a sua falsificação torna-se muito mais difícil.

Em vista disso, a verificação *on-line* de assinaturas, objeto deste estudo, é mais confiável que a verificação de assinaturas *off-line*.

Pode-se perceber também que, com algumas vantagens, a biometria baseada em reconhecimento de assinaturas traz também alguns desafios, como a maior variabilidade intrapessoal em um dado momento e ao longo do tempo.

Ao comparar assinaturas busca-se extrair características que:

- Maximizem a distância interpessoal entre assinaturas de diversas pessoas, de forma que seja mais complicado uma pessoa se passar por outra;
- Miniminize a distância intrapessoal entre assinaturas da mesma pessoa, fazendo com que a variabilidade inerente a esse tipo de sistemas seja minimizada.

Para a verificação *on-line* de assinatura, amostras da assinatura de um usuário (denominadas assinaturas de referência) são previamente inseridas no sistema. Assim, quando um dado usuário, que se apresenta como determinado indivíduo, entra com uma assinatura no sistema (assinatura de teste), essa será comparada com as assinaturas previamente armazenadas (assinaturas de referência), desse mesmo indivíduo. Se o grau de dissimilaridade ultrapassar determinado limite preestabelecido o usuário é rejeitado.

Durante a verificação a assinatura é comparada com todas as assinaturas do conjunto de referência. Essa comparação gera valores que representam a dissimilaridade da assinatura de teste em relação a cada uma das pertencentes ao conjunto. Deve-se selecionar

um método para gerar um valor que represente a dissimilaridade desta assinatura, para então compará-la com um valor limite e assim, determinar sua autenticidade.

Alguns métodos mais comuns são: a utilização do valor de distância mínimo, máximo ou da média dos valores encontrados [10]. Estudos reportam a ocorrência das menores taxas de erro com a utilização do valor de distância mínimo [10].

1.2 Problema

O problema a ser estudado é o descrito a seguir:

- Dada uma assinatura atribuída a um determinado indivíduo, determinar se esta assinatura foi, de fato, escrita por este indivíduo, sendo capaz de identificar falsificações.

Diversas técnicas já foram exploradas pela literatura buscando superar este desafio, e bons resultados foram alcançados utilizando:

- Modelos Escondidos de Markov [21];
- Redes Neurais Artificiais [19];
- Algoritmos Genéticos [13];
- Análise do Componente Principal [6], dentre outras.

1.3 Motivação

A motivação para a realização deste trabalho encontra-se:

- na necessidade de desenvolver novas formas de autenticação mais eficientes e amigáveis para os diversos sistemas de informação baseados em computadores usados diariamente;

- na possibilidade de aprimorar os modelos de autenticação existentes;

- no incremento do uso de biometria como forma complementar de autenticação de usuários/indivíduos/pessoas;

- no potencial de mercado, devido as enorme possibilidades de aplicações práticas e por ser uma modalidade de biometria pouco explorada comercialmente.

1.4 Objetivos

Dentro do contexto de biometria e reconhecimento de assinaturas, consultar a literatura especializada buscando identificar melhores práticas e estratégias utilizadas para realizar a tarefa de autenticar um usuário por meio de sua assinatura.

1.4.1 Objetivos Específicos

- Implementar um método *on-line* de reconhecimento de assinaturas, buscando as maiores taxas de acerto possíveis e minimizando taxas de erro.

- Comparar o algoritmo DTW com o FastDTW na realização do alinhamento de assinaturas, identificando vantagens e desvantagens da utilização de cada um deles.

- Implementar uma aplicação para coleta e verificação de assinaturas.

1.5 Levantamento Bibliográfico

Antes de se chegar a uma proposta para abordar o problema verificação de assinaturas, fez-se necessário uma pesquisa bibliográfica para conhecer algumas das técnicas utilizadas em outros trabalhos, relativos a este tema, publicados ao longo dos últimos anos. Dentre os trabalhos avaliados dedicou-se maior atenção aos de reconhecimento *on-line*. Nesse contexto, foram relacionados alguns trabalhos que são objeto de estudo dentro do escopo desta monografia: verificação *on-line* de assinaturas.

Em [12] Lei e Govindaraju realizam um estudo comparativo a respeito da consistência de processos de extração de parâmetros de assinaturas realizados em métodos de verificação *on-line*. Seus estudos indicam que características simples como coordenadas X e Y, velocidade de escrita e ângulo em relação ao eixo X estão estre as mais consistentes. O artigo cita ainda, o uso do DTW para comparação entre assinaturas por meio do cálculo de distâncias Euclideanas entre os pontos.

Jain at al [8] apresentam um método para verificação on-line de assinaturas baseado no DTW utilizando limiarização para decidir sobre a autenticidade das assinaturas. Seus achados apontam o uso de valores limite específicos para cada usuário mais eficientes do que o uso de um valor comum para a realização da classificação. Os resultados mostram também melhor desempenho com o uso de valores normalizados, sendo FRR = 2,8% e FAR = 1,6% as melhores taxas de erro obtidas em uma base de 1232 assinaturas de 102 indivíduos.

Um método utilizando HMM é apresentado em [20]. Os autores reportam uma taxa de erro total de 6,19% utilizando o algoritmo Baum-Welch, tanto para treinamento quanto classificação, de 496 amostras fornecidas por 31 indivíduos.

Outros métodos, utilizados para verificação *off-line*, são apresentados em [4][19].

Durante a avaliação dos trabalhos apresentados neste capítulo, encontrou-se com frequência três métricas: taxa de falsos positivos (FAR – False Aceptance Rate), falsos negativos (FRR – False Rejection Rate) e taxa igual de erro (EER – Equal Error Rate). Esses valores são utilizados para comparar e medir a eficiência dos algoritmos.

1.6 Produtos Gerados

São dois os principais produtos gerados por este estudo: um algoritmo de verificação de autenticidade de assinaturas e um protótipo de um sistema de autenticação. Os mesmos serão brevemente descritos nos próximos parágrafos e detalhados em capítulos posteriores.

1.6.1 Algoritmo de Verificação

Nos capítulos que seguem, será demonstrada a implementação de um algoritmo de verificação de assinaturas. Essa implementação toma como base o método exposto em [10], que usa o DTW para alinhar assinaturas e para calcular um indicador de dissimilaridade. Foram realizados experimentos com extração de parâmetros de assinaturas. Diferentemente do exposto em outros trabalhos, os resultados melhoraram com a utilização de informações de pressão das assinaturas como é apresentado na seção 3.3.2. Avaliou-se a performance do método utilizando dois classificadores distintos: um linear, que gera um valor de dissimilaridade para os dados coletados, e um classificador SVM[3] que recebe vários parâmetros para realizar a classificação. Ambos foram implementados utilizando valores normalizados para que fossem independentes dos usuários, ou seja, o mesmo classificador é utilizado para todos os usuários. Os valores limite utilizados na classificação são os mesmos independentemente do usuário. Seu funcionamento será melhor detalhado nos capítulos subsequentes.

1.6.2 Protótipo do Sistema

O outro produto deste trabalho é o protótipo de um sistema implementado em Java que permite: a coleta de assinaturas com o uso de mesas digitalizadoras; testes de validação de assinaturas utilizando os algoritmos implementados neste trabalho; a coleta de metainformações a respeito destas assinaturas e seus donos objetivando gerar um banco de dados que possibilite levantamentos estatísticos de características das assinaturas e usuários. Esse banco de dados possibilitará a realização de estudos para aprimoramento dos algoritmos de verificação e também a avaliação de desempenho dos algoritmos existentes e novos, além da execução de testes utilizando as assinaturas que o compõe. Este protótipo será detalhado no apêndice 5.

1.7 Organização

Este trabalho está dividido em 6 capítulos mais um apêndice, sendo que o primeiro é esta introdução no qual consta uma breve contextualização do problema de autenticação de usuários, comparando brevemente os sistemas de autenticação baseados em senha com os sistemas baseados em biometria.

Nos capítulos seguintes estão expostos algoritmos e técnicas que formam a base deste trabalho, que ficaram assim divididos:

- O capítulo 2 aborda, brevemente, técnicas de programação dinâmica que são a base para outros algoritmos deste trabalho. Aborda o algoritmo DTW, sua finalidade e seu funcionamento. Bem como o FastDTW, que é uma aproximação do algoritmo DTW e que visa diminuir sua classe de complexidade, possibilitando o manuseio de maiores volumes de dados.

[3] **SVM (Support Vector Machine ou Máquina de Vetor de Suporte)** é um conjunto de métodos de aprendizagem supervisionada que analisam dados e reconhecem padrões. São utilizados para a classificação e análise de regressão. O algoritmo SVM original foi inventado por Vladimir Vapnik e o padrão atual proposto por Corinna Cortes e Vapnik Vladimir [3]. O SVM padrão recebe um conjunto de dados de entrada, e prevê, para cada entrada de dado, a qual dentre duas possíveis classes ela pertence.

- O capítulo 3 detalha o método proposto neste trabalho para a realização da tarefa de verificar assinaturas.
- Os resultados obtidos com a realização dos experimentos serão apresentados no capítulo 4.
- O capítulo 5 discorre a respeito da implementação realizada e descreve brevemente sua arquitetura, estrutura do projeto, funcionamento e telas como também o processo de instalação e configuração.
- O capítulo 6 apresenta as conclusões obtidas e propõe pontos a serem explorados em trabalhos futuros.
- O apêndice A traz parte do código fonte da implementação realizada.

Capítulo 2

Algoritmos DTW, *FastDTW* e Programação Dinâmica

Neste capítulo serão abordados alguns aspectos sobre programação dinâmica, que é a técnica chave para a resolução do problema de alinhamentos de assinaturas, realizado pelo algoritmo DTW. Este texto é baseado na seção 10.13 do livro Numerical Receipes [15]. Está descrito também o funcionamento do algoritmo DTW e do *FastDTW*, que é uma variação próxima desse, criada por Stan Salvador e Philip Chan, e portanto este capítulo está baseado também nos trabalhos deles [18].

2.1 Programação Dinâmica

Programação dinâmica (PD) é uma técnica de construção de algoritmos para a solução de problemas computacionais, e particularmente problemas de otimização combinatória. Em geral, problemas de PD apresentam duas principais características:

Subestrutura ótima
: Ocorre quando uma solução ótima de um problema contém em seu interior soluções ótimas de subproblemas.

Superposição de subproblemas
: Característica de algoritmos recursivos que analisam o mesmo problema diversas vezes.

Para aplicarmos PD, é necessário que o problema a ser resolvido possa ser representado como uma sequência de escolhas, cada uma com determinado custo. Uma vez determinada essa sequência de escolhas, deseja-se minimizar o custo total do problema e maximizar o benefício. Desse modo, para que seja possível simplificar um problema utilizando programação dinâmica deve ser possível quebrá-lo em uma sequência ordenada e discreta de estágios, cada um com diferentes estados. Estes estágios e estados formam um grafo direcionado[1] (conforme ilustrado na figura 2.1). Deseja-se percorrer este grafo

[1] **Grafo direcionado** (grafo orientado, grafo dirigido ou dígrafo) é um par G = (V,A) de um conjunto V, cujos elementos são chamados vértices ou nodos e A um conjunto de pares ordenados de vértices, chamados arcos, arestas direcionadas, ou setas (e às vezes simplesmente arestas com o conjunto correspondente chamado E ao invés de A, representado G = (V,E)).

a partir de um determinado estado inicial ($i = 0$) até um estado final ($i = N$). Escolhas possíveis que ligam um determinado estado j de um estágio i à um estado k de um estágio $i + 1$ são arestas no grafo. Seu custo é denotado por $c_{jk}(i)$. Sem perda de generalidade pode-se conectar todos os estados de um estágio i a todos os estados de um estágio $i + 1$, porém deve-se atribuir $c_{jk}(i) = \infty$ a caminhos proibidos.

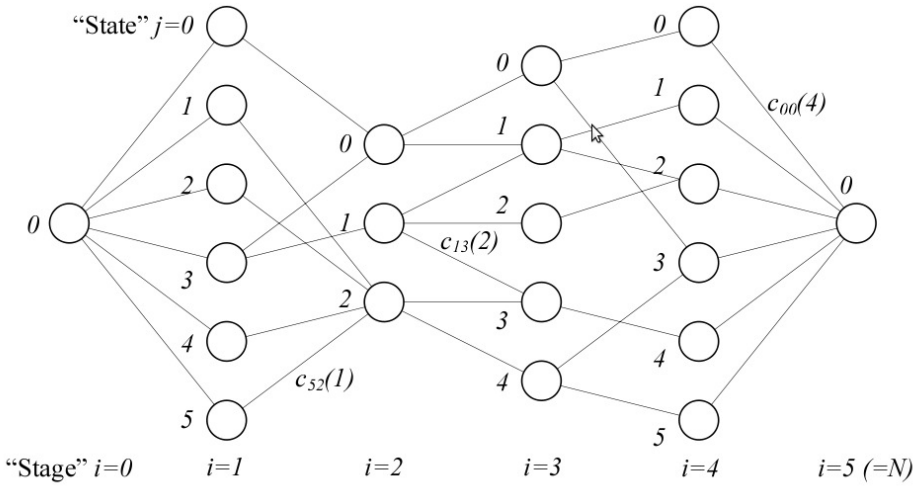

Figura 2.1: Problema clássico de programação dinâmica onde o objetivo é achar o caminho de menor custo entre o ponto de partida e o de chegada. Retirado de [15].

A chave da programação dinâmica é chamada Bellman, Dijkstra ou Viterbi dependendo do campo de treinamento do utilizador. Conforme a figura 2.2, a ideia é que se realize uma única varredura em um grafo direcionado de estágios ordenados, da esquerda para a direita, atribuindo a cada aresta o valor do custo para que o estado localizado no estágio $i + 1$ seja atingido a partir do respectivo estado localizado no estágio i. A cada estado será atribuído um valor que representa o custo do melhor caminho utilizado para atingi-lo.

Quando o estágio final é atingido passa-se a conhecer o custo mínimo global para atingi-lo. O segundo passo para a resolução do problema é retroceder a partir do estágio final localizando qual estado anterior levou à melhor solução (*backtrack*) de modo que quando chegarmos ao estado inicial conheceremos o conjunto de escolhas realizadas que representam a melhor solução, concluindo, desse modo, a resolução do problema.

Em grande parte dos casos, a questão principal em problemas de PD, consiste em encontrar uma organização inteligente do problema, de forma a minimizar o número de estados em cada estágio, e evitar, nas palavras de Bellman, a "maldição da dimensionalidade". Algumas vezes a ordem dos estágios pode não ser cronológica, refletindo somente a decomposição de determinado problema em uma forma mais conveniente. Em [15] é possível encontrar um algoritmo genérico para a resolução de problemas de programação dinâmica.

A seguir será apresentado um exemplo (adaptado de [15]) que se aproxima do problema de alinhamento de assinaturas.

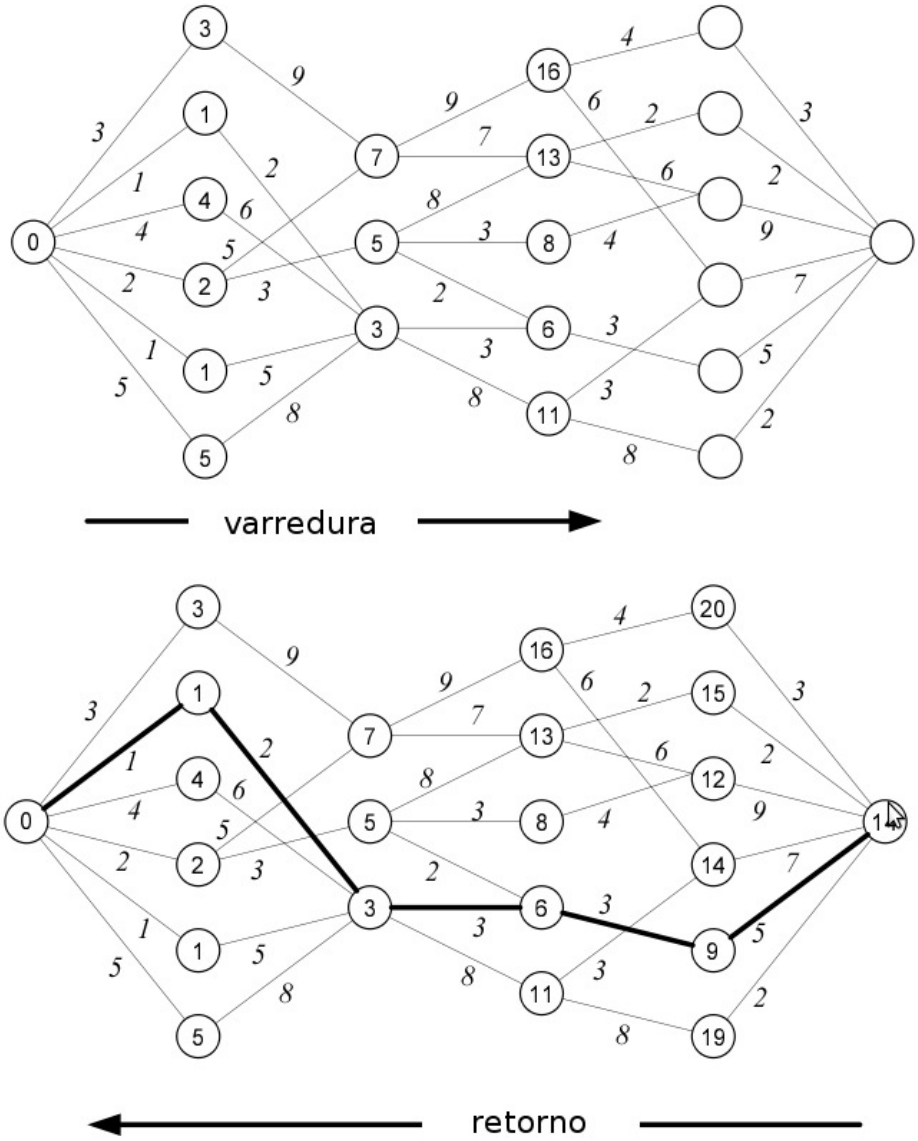

Figura 2.2: Ilustração de duas etapas do processo de resolução de um problema de programação dinâmica utilizando o algoritmo Bellman-Dijkstra-Viterbi. Adaptado de [15].

2.1.1 Exemplo: Alinhamento de Sequências de DNA

Sequencias de DNA de diferentes organismos provenientes de um ancestral comum podem divergir ao longo do tempo pela deleção, inserção ou substituição de bases nitrogenadas em algumas das sequências destes organismos. Deseja-se encontrar o melhor alinhamento entre duas dadas sequências. Para isso, é permitido inserir intervalos em qualquer uma das sequencias, porém ao final serão recompensados os casamentos corretos de pares de bases e penalizados intervalos em branco e casamentos incorretos.

As sequências, antes do alinhamento, são as apresentadas na figura 2.3. Um possível alinhamento é o apresentado na figura 2.4.

Para este alinhamento existem seis recompensas, uma penalidade por incompatibilidade no casamento das bases (destacado em negrito), outras quatro penalidades pela

G A A T T C A G T T A
G G A T C G A

Figura 2.3: Exemplo de duas sequências de DNA antes do alinhamento.

G *A* A T T C A G T T A
G *G* A T C G A

Figura 2.4: Exemplo de um possível alinhamento entre duas sequências de DNA.

existência de quatro lacunas. Para o cálculo do custo inicia-se em zero e soma-se uma unidade para cada penalidade encontrada (casamento incorreto ou espaço em branco). Casamentos corretos não alterarão o valor do custo, de modo que quanto menor o for valor final melhor será o alinhamento.

Needleman e Wunsch [14] indicam que este problema é passível de solução por programação dinâmica de forma que nos permite encontrar todos os possíveis alinhamentos entre as duas sequências, além de identificar o melhor alinhamento possível. Para isso será utilizada uma matriz A_{MxN} onde M e N são os tamanhos da primeira e da segunda sequências respectivamente somados de uma unidade cada um (conforme a tabela 2.1).

A correspondência entre as sequências consiste em um caminho traçado ao longo da matriz, partindo do canto inferior esquerdo e terminando no canto superior direito. Todas as posições da matriz são avaliadas, de uma em uma, começando da que está preenchida com zero (canto inferior esquerdo) e avançando (uma posição por vez) em um dos seguintes sentidos:

para a direita
: o que corresponde a inserção de uma lacuna na sequência que está na vertical, e portanto uma penalidade

Tabela 2.1: Matriz bidimensional utilizada para realizar o alinhamento das sequências de DNA apresentadas no exemplo.

A												
G												
C												
T												
A												
G												
G												
	0											
		G	**A**	**A**	**T**	**T**	**C**	**A**	**G**	**T**	**T**	**A**

para cima
: que analogamente corresponde a inserção de uma lacuna na sequência que está na horizontal, igualmente uma penalidade

ou para a diagonal superior direita
: que corresponde a um casamento correto caso as bases nitrogenadas desta posição sejam iguais nas duas sequências, e uma penalidade caso sejam diferentes

Para organizar o processo e facilitar a avaliação da matriz, preenche-se, coluna por coluna, começando pela primeira (de baixo para cima). Uma vez completa, passa-se para a segunda e assim por diante. A cada nova célula, primeiro verifica-se a ocorrência de casamento correto entre as bases nitrogenadas das duas sequências naquela posição. Em caso positivo o valor da célula passa a ser o mesmo contido na célula da diagonal esquerda inferior, a não ser que este valor seja maior algum dos valores da célula à esquerda ou inferior acrescidos de uma unidade. Neste caso o valor da célula passa a ser o menor dentre os dois somado de uma unidade.

Preenchida toda a matriz, o valor de cada célula representa o custo do melhor caminho para se chegar da primeira célula até ela. Sendo assim, o valor preenchido na ultima célula do canto superior direito representa o custo do melhor alinhamento entre as duas sequências.

Para determinar o caminho de menor custo, ou seja, o melhor alinhamento, necessita-se de um passo a mais. É preciso fazer o caminho inverso do final da matriz até a primeira célula (chamado de *backtrack*, conforme mencionado anteriormente). Esse processo é iniciado da última célula (superior direita) e percorre o caminho inverso, sempre buscando o estado anterior que levou à melhor solução, priorizando diagonais quando seu custo for a um deslocamento lateral. Pode haver mais de uma caminho que represente o melhor custo. Finalmente, o último passo é traduzir esse caminho na série de pares ordenados que o representam, e alinhar as sequências conforme o caminho encontrado.

2.2 Introdução ao DTW

Dynamic Time Warping (DTW) é um algoritmo para encontrar o alinhamento não linear ótimo entre duas se sequências de dados temporais onde uma delas pode apresentar alterações, estando parcialmente alongada ou reduzida em relação à outra, ao longo do eixo tempo. Esse algoritmo também pode ser usado para identificar regiões correspondentes ou mesmo determinar a similaridade de duas sequências. No reconhecimento de assinaturas, pode-se fazer uma analogia com a escrita de uma letra "g", por exemplo. O comprimento da "perna" (arco inferior) da letra pode variar consideravelmente a cada vez que o usuário assina. Essa analogia pode ser estendida à assinatura como um todo. Quando há pouco espaço para assinar, é necessário diminuir a escala da assinatura.

O DTW é frequentemente utilizado no reconhecimento de fala para determinar se as formas de duas ondas representam a mesma frase. Podem haver variações no formato de onda de uma frase falada, como o intervalo de tempo entre uma palavra e outra, e a duração do som de determinados fonemas, mas no geral serão sempre similares.

Além do reconhecimento de fala, o algoritmo DTW se mostrou útil para diversas outras aplicações [2] [9] como reconhecimento de assinaturas evidentemente), mineração de dados, reconhecimento de gestos, robótica e medicina.

A figura 2.5 exemplifica "distorções"/variações que podem ocorrer em sequências similares.

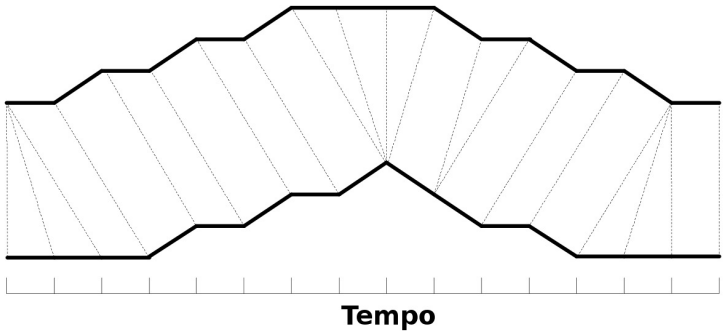

Figura 2.5: Variações entre duas sequências temporais. Adaptado de [18].

2.3 *Dynamic Time Warping* (DTW)

Para determinar o grau de similaridade de duas sequências de valores referenciados no tempo é preciso utilizar um método de medição. Uma forma de realizar essa medição é utilizando a distância Euclideana entre cada um dos pontos de mesmo índice das duas sequências. Porém, este método não é bom para identificar deslocamentos na sequência. Tomando duas sequências idênticas, uma porém, deslocada ao longo do eixo tempo, é possível que o cálculo da distância Euclideana considere-as bastante diferentes uma da outra.

Dadas duas sequências referenciadas no tempo

$$X = x_1, x_2, \ldots, x_i, \ldots, x_{|X|} \tag{2.1}$$

$$Y = y_1, y_2, \ldots, y_i, \ldots, y_{|Y|} \tag{2.2}$$

para encontrar o melhor alinhamento entre elas o algoritmo DTW utiliza técnicas de programação dinâmica, construindo uma matriz com |X| linhas por |Y| colunas (de forma similar ao exemplo de alinhamento de sequências de DNA). O cálculo inicia-se pela primeira célula à esquerda da linha mais inferior e expande-se ao longo da matriz. Os valores das células adjacentes são calculados com base nesta primeira célula. Do mesmo modo, os valores de cada célula são calculados com base no valor das células imediatamente anteriores, até que toda a matriz seja preenchida. O cálculo ocorre sempre no sentido de baixo para cima e da esquerda para a direita (figura 2.6).

Desse modo o valor de cada célula (i, j) da matriz representa o custo do melhor caminho de (1,1) até (i,j). Portanto o valor de (|X|,|Y|) representa o custo do melhor alinhamento entre as sequências e logo a distância entre X e Y.

Temos então um caminho C que representa o melhor alinhamento entre as duas sequências, tal que

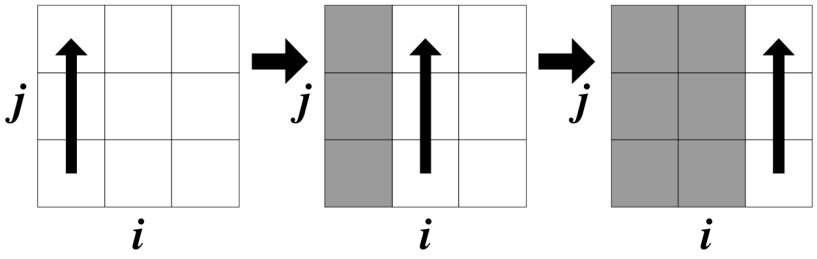

Figura 2.6: Sentido da avaliação das células da matriz de custo. Retirado de [18].

$$C = \{\ c_1, c_2, \ldots, c_K \ \mid \ max(|X|, |Y|) \ \leq \ K \ \leq \ |X| + |Y|\ \} \tag{2.3}$$

onde K é o tamanho do caminho

$$c_1 = (1, 1) \tag{2.4}$$

o primeiro elemento do caminho

$$c_k = (i, j) \tag{2.5}$$

o k-ésimo elemento que compõe o caminho, com i e j os índices do i-ésimo elemento de X e do j-ésimo elemento de Y respectivamente

$$c_K = (|X|, |Y|) \tag{2.6}$$

o K-ésimo elemento que compõe o caminho.

A construção deste caminho é restringida pela seguintes regras: todos os índices de cada uma das sequências deve fazer parte da composição do caminho, representado na figura 2.7, sendo que i e j devem crescer monotonicamente. Logo

$$c_k = (i, j), \quad c_{k+1} = (i', j') \qquad \{i \leq i' \leq i + 1, \quad j \leq j' \leq j + 1\} \tag{2.7}$$

Dado o caminho C, seu custo é obtido da seguinte maneira

$$Dist(C) = \sum_{k=1}^{K} Dist(x_{c_{ki}}, y_{c_{kj}}) \tag{2.8}$$

2.3.1 Reduzindo o tempo de execução do DTW

Como é possível notar, o custo deste algoritmo é polinomial quadrático (O(Nˆ2)), o que faz com que seu tempo de execução aumente consideravelmente rápido a medida que o volume dos dados de entrada cresce, tonando-o proibitivo para certos tipos de aplicações.

Existem algumas técnicas para aumentar a velocidade de execução do DTW porém a maior parte delas implica em perda de precisão. Essas técnicas podem ser classificadas em três diferentes categorias [18]:

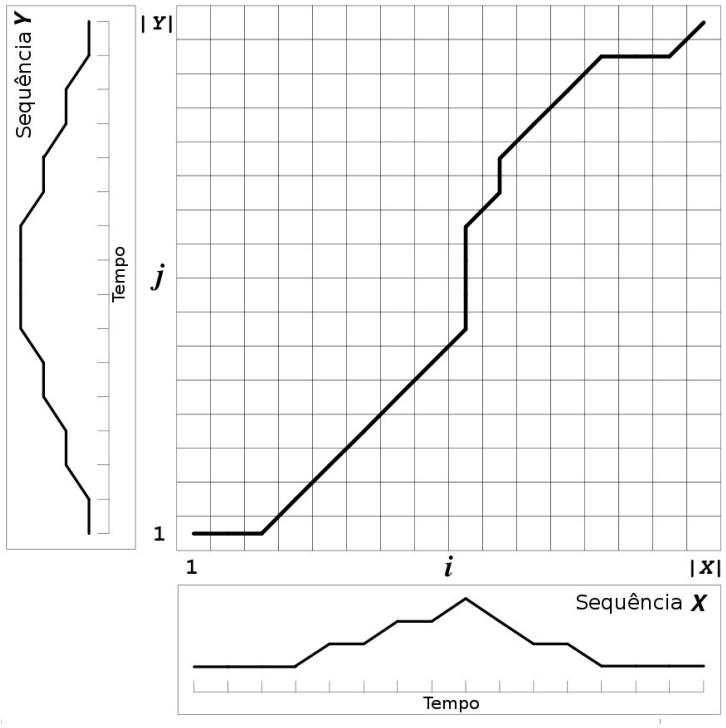

Figura 2.7: Matriz de custo para avaliação do DTW. Adaptado de [18].

Regras de contenção

Limitam o número de células que serão avaliadas no cálculo DTW, restringindo a área avaliada da matriz.

Abstração de dados

Realiza a execução do DTW em uma amostragem reduzida dos dados.

Indexação

Utilização de funções limitadoras para reduzir a quantidade de vezes que o DTW será executado durante operações classificação ou delimitação de sequências temporais.

A utilização de regras de contenção ou limitantes é a técnica mais utilizada para acelerar a execução do DTW. Duas das mais comuns são a *Sakoe-Chiba Band* [17] e a *Itakura Parallelogram* [7], ambas exemplificadas na figura 2.8.

A técnica de "Abstração de dados" acelera a execução do DTW utilizando como entrada para o algoritmo uma amostragem reduzida dos dados [2][9]. A figura 2.9 exemplifica a abstração de dados no cálculo do DTW. No quadro mais à esquerda tem-se a amostra em sua resolução original. É gerada uma amostragem reduzida retirando-se 4 em cada grupo de cinco amostras de cada umas das sequências comparadas. Esse processo reduz consideravelmente a quantidade de dados, tornando a amostra mais fácil de tratar.

Uma vez encontrado o caminho que representa a menor soma no conjunto de menor resolução, esse caminho é mapeado novamente para a resolução original da matriz de custo (figura 2.9). Naturalmente a precisão no cálculo do DTW aumenta conforme aumenta o nível de abstração dos dados.

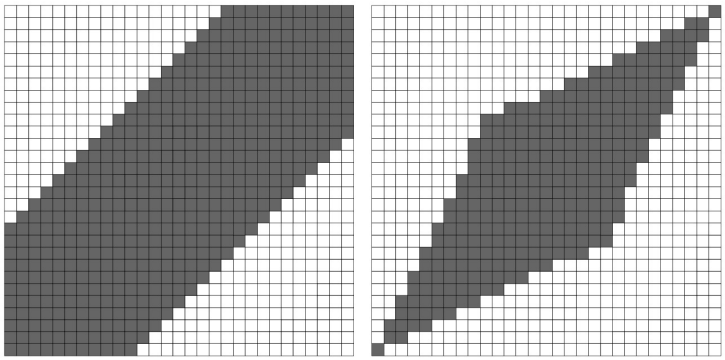

Figura 2.8: Duas regras de contenção: *Sakoe-Chiba Band* (esquerda) e *Itakura Parallelogram* (direita). Retirado de [18].

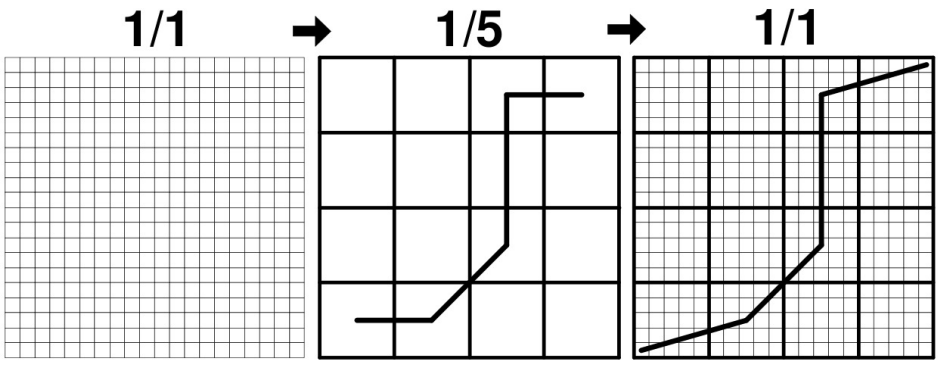

Figura 2.9: Acelerando a execução do DTW com Abstração de Dados. Retirado de [18].

Já a técnica de Indexação reduz consideravelmente o número de vezes que o algoritmo DTW precisa ser executado em determinadas tarefas em que há repetição como a clusterização de um conjunto de sequências temporais e a comparação de uma sequência com diversas outras afim de encontrar a que mais se assemelhe. Essa técnica não reduz a velocidade de execução do DTW em si, mas aumenta a velocidade de execução de determinadas tarefas pela redução do número de execuções do DTW.

2.4 *FastDTW*

O algoritmo *FastDTW* é uma aproximação muito boa do algoritmo DTW, porém com tempo de execução linear O(N). Sua abordagem é uma combinação de duas das técnicas descritas na seção 2.3.1, aplicação de regras de contenção e abstração de dados, trabalhadas de maneira multinível, de modo a agilizar a execução do DTW.

A abordagem multinível referenciada é realizada com três operações chave:

1) Redução

Reduz a taxa de amostragem da sequência, de modo que represente a mesma curva de forma tão precisa quanto possível, com menos pontos. É possível comparar esse procedimento à redução de resolução de uma imagem digital.

2) Projeção

Encontrar um caminho que represente a distância mínima em resolução mais baixa, e projetar esse caminho na matriz de resolução mais alta.

3) Refinamento

Aperfeiçoar o caminho projetado a partir da matriz de baixa resolução realizando ajustes.

A Redução consiste em diminuir o tamanho (ou resolução) de uma sequência referenciada no tempo, realizando a média dos valores de pares adjacentes de pontos ao longo desta sequência. A sequência resultante é reduzida por um fator de dois em relação à sequência original. Esse procedimento pode ser executado de maneira recursiva em uma sequência, criando diversas resoluções diferentes da sequência original. O DTW é executado na sequência de menor resolução, e o caminho encontrado, projetado na matriz de custo da sequência de resolução mais alta. Considerando que a resolução decresce em um fator de dois, cada ponto da matriz de menor resolução corresponde, no mínimo a quatro pontos na matriz de maior resolução. O caminho projetado na matriz de maior resolução é utilizado como heurística durante o processo de Refinamento, de forma que o melhor caminho será encontrado em regiões vizinhas ao caminho projetado, de modo que o tamanho da área de avaliação é controlado por um raio ao longo do caminho projetado.

No DTW padrão temos custo O(Nˆ2) pois é preciso avaliar toda a matriz de custo para garantir que a solução ideal seja encontrada. Enquanto o tamanho das sequências cresce linearmente, o tamanho da matriz de custo cresce quadraticamente. No *FastDTW* a matriz é avaliada completamente apenas no nível de menor resolução. Nos níveis acima, é avaliada apenas nas regiões vizinhas ao caminho projetado, de acordo com o tamanho de raio especificado. Levando em consideração que o tamanho do caminho cresce linearmente, do mesmo modo que o tamanho das sequências, a abordagem multinível tem custo O(N).

O primeiro passo do algoritmo é criar todas as resoluções a serem avaliadas. A figura 2.10 ilustra quatro diferentes resoluções da sequência que aparece na figura 2.7. O traço em preto corresponde ao melhor caminho encontrado para cada uma das resoluções. A área sombreada em cinza escuro representa o caminho encontrado na resolução baixa, projetado projetado para a resolução mais alta. A área em cinza escuro, o raio adicional que será avaliado na matriz, aumentando, dessa forma, as chances de encontrar a solução ótima. Na figura 2.10 o raio está definido em 1.

Na etapa de Refinamento a região marcada em cinza funciona como regra de contenção, restringindo a área da matriz que será avaliada pelo DTW. Uma vez que o caminho é refinado, ele é projetado novamente para uma resolução maior. Esse processo é realizado repetidamente até que se obtenha o caminho na resolução original dos dados.

Observe que o caminho encontrado pelo *FastDTW* na figura 2.10 é o caminho ideal encontrado pelo DTW. A diferença é que o *FastDTW* avalia somente as áreas sombreadas, enquanto o DTW avalia toda a matriz. Mesmo considerando todas as diferentes resoluções avaliadas no *FastDTW*, são avaliadas 4+16+44+100 = 164 enquanto o DTW avalia todas as 256 (16ˆ2) células da matriz. Em um primeiro momento este ganho pode não parecer muito significativo, principalmente pelo esforço adicional necessário para a criação das outras quatro resoluções. No entanto, a região avaliada pelo *FastDTW* cresce linearmente, acompanhando o comprimentos das sequências, visto que o raio avaliado ao longo do

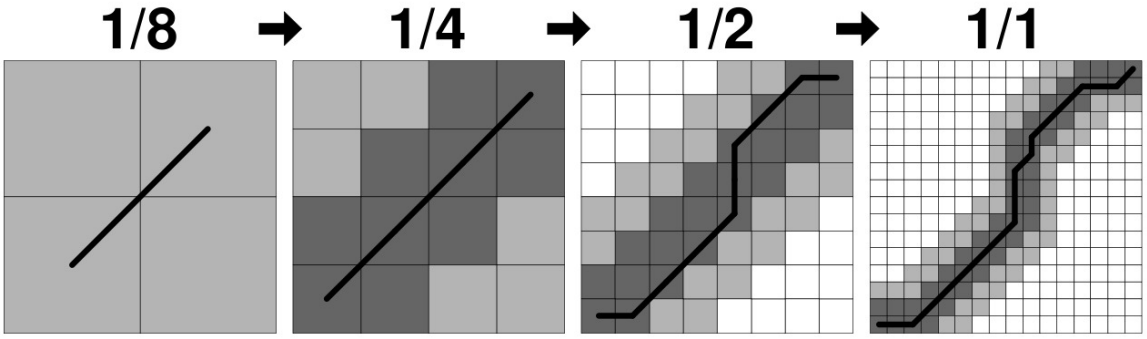

Figura 2.10: Quatro diferentes resoluções avaliadas durante a execução do algoritmo *FastDTW*. Retirado de [18].

Tabela 2.2: Taxa média de erro na execução de aproximações para o DTW conforme o tamanho do raio utilizado. Dados obtidos de [18].

	raio				
	0	*1*	*10*	*20*	*30*
FastDTW	19,2%	8,6%	1,5%	0,8%	0,6%
Abstração	983,3%	547,9%	6,5%	2,8%	1,8%
Faixa	2749,2%	2385,7%	794,1%	136,8%	9,3%

caminho é constante em todas as resoluções. Já o DTW sempre terá crescimento $N\hat{}2$, mesmo com o comprimentos das sequências crescendo linearmente.

É importante ressaltar que na maiorias dos casos o caminho encontrado pelo *FastDTW* será muito próximo do ideal, porém existe uma margem de erro que varia de acordo com o tamanho do raio determinado. Entretanto, caso o raio seja definido com tamanho tão grande quanto o de uma das sequências o algoritmo passa a ter a mesma eficiência do DTW, retornando sempre o caminho ótimo, porém com custo de execução O(N^2).

2.4.1 Eficiência

Para demonstrar a eficiência do algoritmo o *FastDTW* é comparado a outros métodos de aproximação existentes. São feitas também, medições para determinar a velocidade de execução do algoritmo, comparando-se seu tempo de execução com o DTW para sequências de mesmo tamanho.

A tabela 2.2 retirada de [18] mostra o erro médio encontrado para três algoritmos que aproximam o DTW. Todos eles foram executados utilizando valores de raio iguais a 0, 1, 10, 20 e 30. O taxa de erros do *FastDTW* é consideravelmente menor que a dos demais algoritmos, principalmente para pequenos valores de raio. A taxa de erro começa a se aproximar de 0% quando o raio é definido igual ou superior a 10. O método de abstração de dados é impreciso para pequenos valores de raio, mas começa a ser razoavelmente

Tabela 2.3: Tempo de execução dos algoritmos DTW e *FastDTW* em relação do tamanho da sequência de dados e do raio de avaliação. Dados obtidos de [18].

	Tamanho da Sequência			
	100	*1.000*	*10.000*	*100.000*
DTW	0,02	0,92	57,45	7969,59
***FastDTW* (raio=0)**	0,01	0,02	0,38	67,94
***FastDTW* (raio=100)**	0,02	0,06	8,42	207,19

preciso quando executado com raios maiores. O algoritmo de banda é muito impreciso para todos os raio configurações, exceto 30.

Com relação ao tempo de execução, o algoritmo *FastDTW* foi significativamente mais rápido que o DTW em todos os casos. Segundo [18] o *FastDTW* é 50 vezes mais rápido que o DTW usando-se valores de raio = 0 e 150 vezes mais rápido com raio = 100. Isso para casos em que as sequências de dados ultrapassam 150mil pontos. Alguns dos resultados obtidos com o algoritmo são exibidos na tabela 2.3. Podemos observar que para sequências de tamanho menor que 1000 a diferença no tempo de execução é muito pequena. O tamanho médio das assinaturas é de 300 pontos, variando de acordo com a taxa de amostragem dos dispositivo de captura.

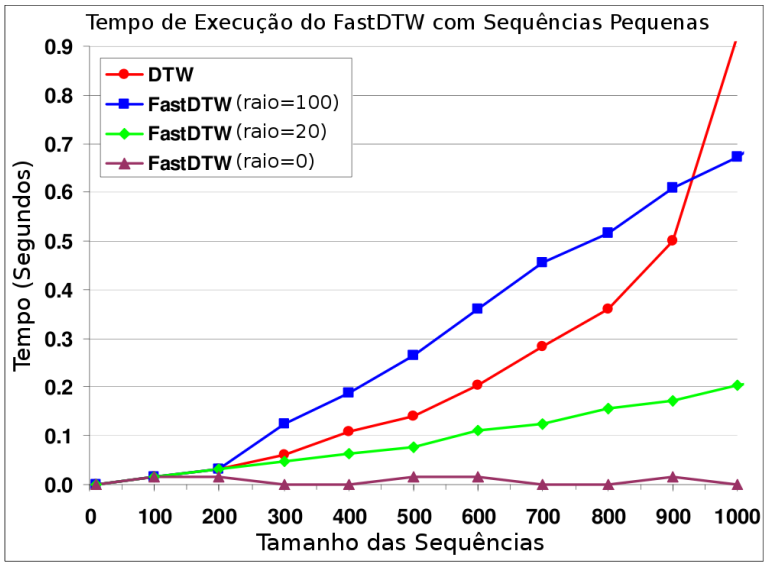

Figura 2.11: Eficiência do *FastDTW* comparado ao DTW para sequências pequenas. Adaptado de [18].

Capítulo 3

Algoritmo de Verificação - Método Proposto

Neste capítulo será descrita a implementação do algoritmo de verificação proposto que, de forma geral, compartilha a mesma abordagem de alguns sistemas existentes, tomando como base o método exposto em [10] conforme citado anteriormente. A o diferencial do método exposto em [10] encontra-se principalmente na forma extração de características das assinaturas e seus respectivos conjuntos de referência. Foram utilizados valores normalizados para tornar os classificadores independentes de usuários de forma semelhante ao demonstrado em [10].

Será vista a metodologia utilizada para realizar a captura de dados, pré-processamento, alinhamento, comparação entre assinaturas e treinamento de classificadores.

De modo geral, o processo de verificação proposto possui as etapas a seguir: realiza-se uma etapa de cadastramento na qual o usuário fornece 5 amostras de sua assinatura, as quais serão chamadas "conjunto de referência"(R_{ID}). Utilizado-se o conjunto de referência para determinar parâmetros que serão indicadores do grau de variabilidade entre as assinaturas do usuário. Este conjunto de dados juntamente com o conjunto de referência será armazenado em um banco de dados sendo pré-requisito para posterior verificação de autenticidade das assinaturas deste usuário.

Quando o sistema recebe uma assinatura para que seja feita sua verificação de autenticidade, compara-se esta assinatura com cada uma das demais assinaturas que compõem o conjunto de referência do usuário ao qual a assinatura é atribuída. Com essa comparação, gera-se indicadores de dissimilaridade entre as assinaturas. Corrigi-se os valores de distância mínimo, máximo e distância da assinatura modelo[1] por um percentual de igualdade entre as assinaturas. Utiliza-se estes valores para compor um vetor de características juntamente com valores que representam o que pode-se denominar informalmente como "energia interna" de diversas partes da assinatura, todos normalizados por seus respectivos valores médios encontrados no conjunto de referência. Este vetor de características será utilizado no processo de classificação, detalhado em seções seguintes, juntamente com as demais etapas e processos citados anteriormente.

[1] Assinatura modelo é aquela que apresenta o menor valor médio de distância para as demais assinaturas do conjunto de referência.

3.1 Aquisição de dados

Para testes e avaliação de eficiência do algoritmo foram utilizadas as assinaturas que compõem o *Visual Supcorpus* do banco de dados SUSIG (*An On-line Handwritten Signature Database, Associated Protocols and Benchmark Results*) [11] que é distribuído gratuitamente.

O *Visual Supcorpus*, conforme [11], é composto por amostras coletadas com a mesa digitalizadora sensível à pressão *Interlink Electronics's ePad-ink* (similar ao da figura 3.1) que possui uma tela de LCD acoplada para fornecer ao usuário retorno do que está sendo feito. Isso torna o processo mais próximo da assinatura em papel, sendo mais natural e confortável para o usuário, melhorando a qualidade das amostras. A tela de LCD do dispositivo possui dimensões de 3x2,20 polegadas com resolução de 300dpi. Sua frequência de amostragem é 100Hz, fornecendo as coordenadas x e y de cada ponto da trajetória da assinatura, a pressão no eixo z (aferida com 128 níveis de precisão) e o *timestamp*[2].

Figura 3.1: Dispositivo de captura de dados semelhante ao utilizado para a captura de assinaturas que compõe o *Visual Subcorpus* do SUSIG.

Esse banco de dados é composto por assinaturas cedidas por 100 pessoas distintas (29 mulheres e 71 homens), a maioria, estudantes e funcionários da Universidade de Sabanci na Turquia, com idades variando entre 21 e 52 anos. Para o processo de coleta, cada indivíduo recebeu uma breve explicação a respeito do objetivo da coleta dos dados, sem no entanto, fornecer maiores detalhes sobre o funcionamento de um sistema de verificação de assinaturas.

Primeiramente foi solicitado a cada indivíduo que fornecesse 20 amostras de sua assinatura habitual sem impor restrições quanto a forma de assinar. Essa coleta foi dividida em duas sessões com aproximadamente uma semana de intervalo nas quais foram coletadas 10 amostras. Em seguida, foi solicitado a cada indivíduo que forneceu sua assinatura, que falsificasse a assinatura de outro indivíduo do grupo, escolhida aleatoriamente. Para

[2] ***Timestamp*** é o registro da hora de ocorrência de um evento com precisão de milissegundos.

Tabela 3.1: Composição do *Visual Supcorpus* do SUSIG.

Pacote de Dados	**Tipo**	**Usuários**	**Amostras /Usuário**	**Tamanho**
Sessão 1	Autêntica	100	10	1000
Sessão 2	Autêntica	100	10	1000
Falsificação Hábil	Falsa	100	5	500
Falsificação Muito Hábil	Falsa	100	5	500
Validação	Autêntica/Falsa	10	10/10	200

realizar as falsificações, o indivíduo teve acesso a amostras da assinatura que teve de falsificar, coletadas na primeira sessão, bem como oportunidade de assistir diversas vezes a animações da realização da assinatura autêntica. Além disso, pôde praticar várias vezes antes de realizar a falsificação. Quando estivessem satisfeitos com a qualidade de suas falsificações, era solicitado a estes indivíduos que fornecessem 5 amostras. Foram coletadas também 5 falsificações realizadas com maior perícia (cujo processo de obtenção não é explicitado), totalizando 10 falsificações por indivíduo. A tabela 3.1 resume a composição do banco de dados.

Com o objetivo de resguardar as pessoas que contribuíram para a construção do SUSIG, algumas das assinaturas que originalmente compunham *Visual Supcorpus* não são distribuídas para a comunidade científica pelo fato de suas imagens terem sido divulgadas publicamente. Portanto foram disponibilizadas apenas a 2820 amostras que correspondem a 1880 assinaturas originais e 940 falsificações de 94 usuários distintos.

Além deste banco de dados, foram utilizadas amostras capturadas com uma mesa digitalizadora sensível à pressão da linha *Wacom Bamboo Fun* modelo MTE450, cuja área de captura apresenta dimensões de 5,8 polegadas de largura por 3,7 de altura; resolução de 2.540 linhas por polegada; 512 níveis de sensibilidade à pressão; frequência de amostragem de 140Hz. Este modelo fornece coordenadas x e y, pressão e *timestamp* para cada um dos pontos da trajetória da assinatura. No entanto, esse modelo não possui tela de LCD acoplada para fornecer retorno ao usuário.

3.2 Pré-processamento

A maioria dos sistemas existentes realiza uma reamostragem da assinatura, a fim de remover pontos redundantes acelerando as comparações, ou mesmo para obter uma representação da assinatura baseada na forma e remover as dependências de tempo. No entanto, reamostragem também acarreta em significativa perda de informações, visto que os dados aparentemente redundantes nos trazem informações sobre as características de velocidade de escrita do assinante.

Kholmatov at al [10] reportam que a não reamostragem compensa significativamente o fato de não termos a velocidade normalizada. Além de que a reamostragem reduz significativamente o desempenho do DTW visto que as assinaturas seriam compostas por conjuntos de pontos equidistantes temporal ou espacialmente. É importante ressaltar que a comparação realizada pelo DTW não se baseia na forma da assinatura, portanto o fato

Figura 3.2: Mesa digitalizadora *Wacom Bamboo Fun* (MTE450) utilizada para captura de amostras de assinaturas.

de conseguir-se capturar a forma da assinatura, mesmo reduzindo a quantidade de dados, não é relevante para este método.

3.3 Extração de Parâmetros

Os parâmetros utilizados para verificação de assinaturas são classificados em dois tipos: locais, que se referem a pontos específicos da assinatura e globais, que se referem a assinatura como um todo. Como exemplo de característica local temos a pressão em determinado ponto da assinatura. Tempo total da assinatura e velocidade média são exemplos de características globais.

Em experimentos, foi verificado que o processo de assinar apresenta maior variabilidade quanto a parâmetros globais como velocidade média, tempo total, pressão média, de modo que dificulta-se a inferência quanto a autenticidade da assinatura baseando-se nesses parâmetros. De um modo geral, assinaturas tendem a apresentar maior consistência em partes específicas como no início, ou mesmo nas primeiras letras de cada palavra. Com isso, pretende-se explicitar o fato de que existem alguns trechos da assinatura que são realizados com maior atenção e cuidado do que outros, o que faz com que apresentem maior consistência, e portanto, pouca variação de uma amostra para outra. É na análise desses pontos que parâmetros locais se destacam.

A seguir serão vistas as informações e características selecionadas para realizar o processo de classificação.

3.3.1 Deslocamento

Para comparação entre pontos de diferentes assinaturas foram consideradas as diferenças entre pontos consecutivos (Δx, Δy) da assinatura. Segundo [10], esse método resultou no menor erro em relação a dois outros métodos testados: o primeiro, usando as coor-

denadas de cada ponto relativas ao primeiro ponto da assinatura; o segundo utilizando diferenças de curvatura entre pontos consecutivos da assinatura. É importante ressaltar Δx e Δy são invariantes com relação a translação. Sua utilização em conjunto com o algoritmo de alinhamento escolhido (DTW) captura de forma satisfatória similaridades entre assinaturas e ignora diferenças "irrelevantes" como pequenas alterações de comprimento ou ângulo entre pontos.

3.3.2 Pressão

Com relação à pressão, diferentemente do relatado em [10], nota-se um pequeno, porém significativo, aumento na precisão do algoritmo, quando utilizada essa informação na comparação de assinaturas. Foram realizados experimentos nos quais extraiu-se informações de pressão de dois modos diferentes, com o objetivo de identificar o impacto do uso destas informações sobre o desempenho alcançado pelos classificadores. Essas informações são utilizadas em conjunto ou separadamente, dependendo do classificador, conforme será explicado na seção 3.6.

No primeiro caso, a informação de pressão relativa a cada ponto da assinatura é passada conjuntamente com Δx e Δy para o algoritmo de alinhamento (DTW), que trabalha com vetores multidimensionais. Conseguiu-se uma abordagem mais eficiente no segundo caso. Em seus estudos, Kholmatov at al [10] relatam que foi possível notar um aumento na quantidade de pressão empregada na realização de falsificações. Utilizando-se dessa informação foi feita a opção por calcular a soma dos quadrados da pressão de cada ponto da assinatura (denominado "energia interna", conforme já mencionado) de acordo com a equação 3.1, para tentar identificar a variação nos padrões de pressão de falsificações. Entretanto, essa informação calculada para a assinatura como um todo, do mesmo modo que as demais características globais, não tem muito significado. Sabe-se que em determinadas partes de uma assinatura é empregada maior força do que em outras. Essa dinâmica tende a se manter similar. Desse modo, resolveu-se dividir a assinatura em N-partes iguais, de modo que cada parte contém 1/N do total de pontos da assinatura. Calculou-se então a energia interna de cada partição, conseguindo assim, uma inferência melhor da variação da pressão no decorrer da assinatura. Em experimentos, identificou-se N igual a 32 como sendo o número de partições que resultou nas melhores taxas de acerto (conforme veremos na seção 4.3).

$$\text{Energia Interna} = \sum_{i=\alpha}^{\beta} p[i]^2 \tag{3.1}$$

onde p corresponde à pressão no i-ésimo ponto da partição $[\alpha - \beta]$ de uma assinatura. As velocidades local e global são consideradas apenas indiretamente durante o processo de alinhamento. Como a mesa digitalizadora coleta os dados com amostragem constante, locais de maior velocidade na assinatura apresentarão um menor acúmulo de pontos. O inverso também é válido, locais com menor velocidade apresentarão maior acúmulo de pontos. Diferenças de velocidade causarão a ocorrência de diversos pontos estranhos que serão penalizados pelo DTW.

3.4 Alinhamento da Assinatura

Tanto na etapa de cadastramento do usuário quanto na verificação, é necessário alinhar duas assinaturas para estabelecer um grau de similaridade entre elas. Por ser amplamente utilizado para alinhar vetores de tamanhos diferentes [1], utilizou-se o algoritmo DTW (equação 3.2) para comparar amostras de assinaturas que, nada mais são que vetores de pontos no espaço, de modo que este é exatamente o problema que o DTW ataca. Além disso, nunca existirão duas amostras idênticas de uma assinatura, de forma que um conjunto não pode ser comparado ao outro utilizando igualdade simples entre os pontos.

Entenda-se por assinatura um vetor de tamanho N igual a quantidade de pontos da assinatura coletada, no qual cada elemento consiste de um outro vetor tridimensional composto por Δx, Δy e pressão. Onde Δx e Δy correspondem as diferenças entre pontos consecutivos (deslocamento) dadas em coordenadas cartesianas.

Conforme abordado no capítulo 2 o DTW encontra o melhor alinhamento não linear para duas sequencias de dados, gerando um valor que representa a menor distância global entre essas duas sequencias. A equação abaixo retirada de [10] apresenta o cálculo da distancia global entre duas assinaturas.

$$C[i,j] = \text{Min} \begin{cases} C[i-1,j] + \gamma, \\ C[i,j-1] + \gamma, \\ C[i-1,j-1] + \text{Dist}(S_1[i], S_2[j]) \end{cases} \tag{3.2}$$

onde $S_i[j]$ denota o j-ésimo ponto na trajetória da i-ésima assinatura, γ é uma penalidade constante que pode ser opcionalmente adicionada quando a soma de células não diagonais, e

$$\text{Dist}(x,y) = \begin{cases} 0 & \text{if} ||x-y|| < \theta, \\ ||x-y|| - \theta & \text{demais casos.} \end{cases} \tag{3.3}$$

onde θ é um valor de limite constante e opcional utilizado de forma a ignorar pequenas variações entre pontos.

3.4.1 Penalidade adicional para o alinhamento de assinaturas

Como forma adicional de penalidade para a distância global resultante do alinhamento de assinaturas, foi desenvolvido um coeficiente baseado no percentual de alinhamento ótimo da assinatura de menor comprimento. Isso é feito do seguinte modo: a quantidade de diagonais que compõe o caminho que representa a menor distância global entre duas assinaturas dentro de uma matriz de custo pode ser, no máximo, igual ao tamanho da menor assinatura. Denominou-se PD o percentual de diagonais do menor caminho da matriz de custo em relação a assinatura de menor comprimento. O cálculo da penalidade é feito conforme a equação 3.4.

$$Penalidade = (1 + (1 - PD))^4 \tag{3.4}$$

Multiplica-se então valor de penalidade obtido pela distância global encontrada pelo DTW.

3.4.2 DTW versus FastDTW

Conforme apresentado em capítulos anteriores, o algoritmo DTW possui custo polinomial quadrático, o que pode consumir muitos recursos de memória e processamento, dependendo do tamanho das sequências de dados a serem comparadas. O FastDTW [18] é uma aproximação muito boa desse algoritmo, cujo custo pode ser ajustado conforme apresentado no capítulo 2.4. Um custo menor, entretanto, aumenta o erro, visto que esse algoritmo é uma aproximação. Dependendo do raio escolhido, é provável que na maioria dos casos o FastDTW seja equivalente ao DTW.

Em experimentos constatou-se que devido ao tamanho relativamente pequeno das assinaturas e do aumento de capacidade dos processadores o DTW foi executado com uma velocidade satisfatória para o problema apresentado, de forma que não se justificou a utilização do FastDTW nesse contexto. Portanto, apenas para fins de comparação, realizou-se alguns testes em um mesmo cenário utilizando o DTW e o FastDTW com diferentes raios para verificar impactos sobre as taxas de erro do algoritmo de verificação de assinaturas em cada um dos casos. Os resultados estão expostos no capítulo 4.

3.5 Cadastramento de usuários

Durante a etapa de cadastramento o usuário deve fornecer um determinado número de assinaturas para compor o conjunto de referência (R_{ID}). Do modo similar trabalhos existentes [10], definiu-se em 5 assinaturas o tamanho desse conjunto, que é utilizado para, por meio de comparações, determinar variações na forma de assinar do usuário. Utiliza-se esses dados posteriormente nos processos de treinamento de classificadores e verificação.

Primeiramente compara-se todas as assinaturas do conjunto de referência par a par utilizando o DTW, conforme a seção 3.4. Denomina-se "assinatura modelo" a assinatura que apresentar a menor distância média para as demais assinaturas do conjunto de referência. Em seguida calcula-se algumas estatísticas que caracterizam a variação encontrada no conjunto de referência. Calcula-se as médias das

- distâncias de cada assinatura de referência para seu vizinho mais próximo ($\overline{d_{\mathrm{min}}}(R_{ID})$)
- distâncias de cada assinatura de referência para seu vizinho mais distante ($(\overline{d_{\mathrm{max}}}(R_{ID})$)
- distâncias de cada assinatura de referência para a assinatura modelo ($(\overline{d_{\mathrm{modelo}}}(R_{ID})$)

Tabela 3.2: Composição dos conjuntos de treinamento e verificação.

Conjunto	**Usuários**	**Amostras**	**Percentual**
Treinamento	47	1410(47/20)	50%
Verificação	47	1410(47/20)	50%

Esses valores são similares aos calculados para uma assinatura na fase de verificação conforme ilustrado na figura 3.3, e servirão para normalizar os valores de distância global mínimo, máximo e modelo (valor encontrado quando comparada a assinatura em verificação à assinatura modelo do R_{ID}) encontrados para assinaturas em verificação com relação ao R_{ID}.

A figura 3.3 ilustra as assinaturas do conjunto de referência sendo comparadas a uma assinatura candidata Y, onde X_{i} representa a i-ésima assinatura de referência fornecida pelo usuário e Y representa a assinatura a ser verificada. X_{M} é a assinatura modelo para a qual as distâncias médias entres as assinaturas de referência são as mínimas possíveis dentre todo o conjunto de referência.

Com o mesmo propósito de normalizar valores encontrados, tanto na fase de treinamento quanto na verificação, calculamos a quantidade de energia interna de cada uma das 32 partições das assinaturas do R_{ID} conforme explicado na seção 3.3.2.

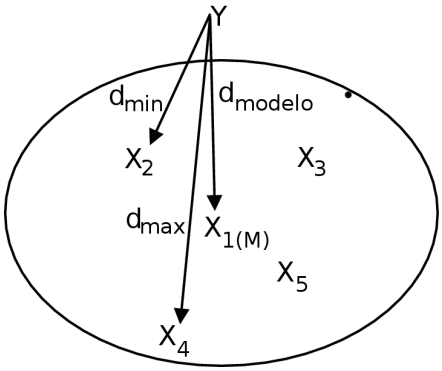

Figura 3.3: Representação das distâncias de alinhamento utilizadas no processo de verificação. Adaptado de [10].

3.6 Treinamento dos classificadores

Dos 94 usuários disponibilizados no *Visual Supcorpus* separamos os 47 primeiros para compor conjunto de treinamento, totalizando 1410 amostras. Os demais usuários (47 usuários, 1410 amostras) foram utilizados para compor o conjunto de validação, conforme detalhado na tabela 3.2.

Foram utilizados dois tipos de classificadores distintos para diferenciar assinaturas genuínas e falsificações. Um classificador Linear e um do tipo SVM (*Support Vector Ma-*

chine). Estes classificadores recebem como entrada conjuntos distintos de características que serão detalhados a seguir, juntamente com seus processos de treinamento.

3.6.1 Classificador Linear

Um dos métodos utilizados para realizar a classificação das assinaturas é a limiarização. Por meio de comparações da assinatura candidata com as demais assinaturas do conjunto de referência é obtido um valor que representa o grau de similaridade dessa assinatura com o referido conjunto. O método de obtenção desse valore é explicitado a seguir. Determina-se então um valor limite λ constante para o qual caso os valores gerados estejam abaixo deste λ a assinatura em questão é considerada autêntica, caso contrario é considerada falsa.

As primeiras 5 amostras genuínas de cada usuário serão utilizadas para formar seu conjunto de referência (R_{ID}). Cada uma das assinaturas remanescentes (Y) do conjunto de treinamento é comparada com as demais assinaturas do conjunto de referência (R_{ID}) a que supostamente pertencem para que sejam obtidos os seguintes valores: ($d_{\min}(Y, R_{\mathrm{ID}})$), ($d_{\max}(Y, R_{\mathrm{ID}})$) e ($d_{\mathrm{modelo}}(Y, R_{\mathrm{ID}})$) (conforme ilustrado na figura 3.3).

Esses valore são então normalizados pelas médias correspondentes obtidas no conjunto de referência, resultando no seguinte vetor tridimensional de características (F_{Y}).

$$F_Y = \begin{bmatrix} d_{\min}(Y, R_{\mathrm{ID}})/\overline{d_{\min}}(R_{\mathrm{ID}}) \\ d_{\max}(Y, R_{\mathrm{ID}})/\overline{d_{\max}}(R_{\mathrm{ID}}) \\ d_{\mathrm{modelo}}(Y, R_{\mathrm{ID}})/\overline{d_{\mathrm{modelo}}}(R_{\mathrm{ID}}) \end{bmatrix} \tag{3.5}$$

Esse processo de normalização é realizado da mesma maneira tanto para assinaturas de treinamento durante o processo de treinamento quanto para assinaturas de teste durante o processo de teste.

É calculada então a variância (equações 3.6 e 3.7) destes três valores obtidos reduzindo, desta maneira, a dimensionalidade deste vetor de três para uma. A partir destes valores o classificador será calibrado buscando o melhor valor limite que separe assinaturas genuínas de falsificações.

O cálculo da variância é realizado conforme segue.

Dado X uma variável aleatória, tal que

$$X = \{d_{\min}, d_{\max}, d_{\mathrm{modelo}}\} \tag{3.6}$$

Se $\mu = \mathrm{E}(X)$ é o valor esperado (média) de X então a variância é

$$\mathrm{var}(X) = E((X - \mu)^2) \tag{3.7}$$

3.6.2 Classificador SVM

Classificadores SVM apresentam a vantagem de podermos utilizar vários dados como entrada, de modo que para seu treinamento utilizamos todos os dados obtidos na seção 3.6.1 e adicionamos ainda ao conjunto de características, informações de pressão.

Cada assinatura Y do conjunto de treinamento é dividida em N=32 partes iguais e então é calculada a "energia interna" de cada uma das partes conforme descrito na seção 3.3.2, resultando em um vetor de tamanho N no qual cada elemento representa o valor encontrado para cada uma das partes da assinatura. Estes valores são normalizados pelas respectivas médias encontradas nas assinaturas do conjunto de referência.

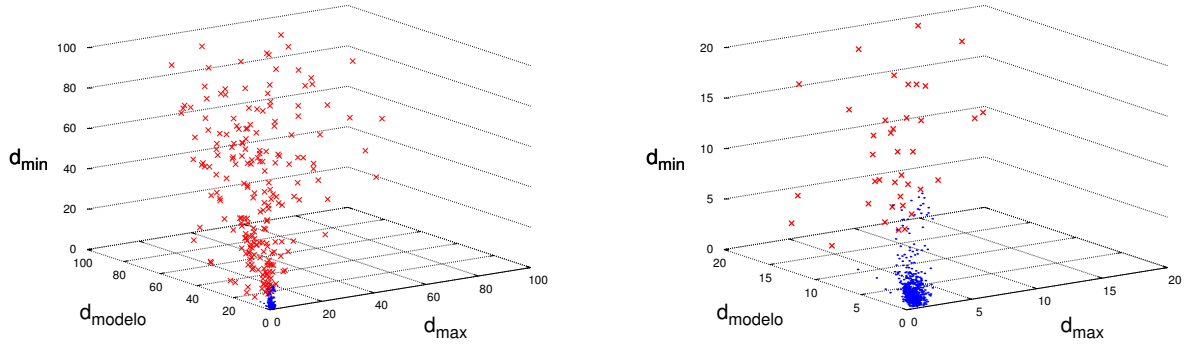

Figura 3.4: Gráfico mostrando os pontos de assinaturas genuínas (em azul) e falsificações (marcadas com "x"em vermelho) com relação ao vetor tridimensional. O gráfico da direita é uma ampliação do intervalo [0-20] do gráfico à esquerda.

Note que estão sendo modeladas as classes de assinaturas genuínas e falsas durante a fase de treinamento, de modo que uma vez concluída não será necessário treinar novamente os classificadores à medida que novos usuários sejam inseridos no sistema.

3.7 Verificação

Para realizar a verificação de uma assinatura, primeiramente precisamos compará-la com todas as assinaturas do conjunto de referência do usuário. Desta comparação utilizaremos o valor de distância para a assinatura modelo (d_{modelo}), o maior valor de distância (d_{max}), o menor valor de distância (d_{min}) e os valores de energia interna de cada partição da assinatura. Esses valores serão normalizados pelos respectivos valores médios encontrados no conjunto de referência extraídos na seção 3.3. O vetor de características resultante é utilizado para determinar se a assinatura é autêntica ou falsa utilizando os classificadores descritos na seção 3.6.

Capítulo 4

Resultados Obtidos

Neste capítulo serão abordadas as observações e resultados dos experimentos realizados. Também serão explicitados os dados coletados e as análises realizadas.

4.1 Seleção de características para os classificadores

Conforme descrito na seção 3.3.1, optou-se por utilizar o deslocamento (Δx, Δy), relativo ao primeiro ponto da assinatura, para descrever sua forma. Em conjunto com informações de pressão e tempo esses dados foram utilizados pelo DTW para comparar as assinaturas e gerar um indicador de dissimilaridade entre elas.

Os valores gerados pelo DTW na comparação com as assinaturas do conjunto de referência, geraram três estatísticas básicas (d_{min}, d_{max}, d_{modelo}) calculadas conforme descrito na seção 3.5. Adicionalmente, a partir do cálculo da *variância* desses três elementos foi gerado um quarto valor de referência.

Nos experimentos, utilizando o classificador linear combinado com cada um destes valores isoladamente, observou-se que a *variância* foi o indicador que obteve maior sucesso em separar assinaturas autênticas das falsificações, utilizando limiarização, tornando-se assim o principal valor de referência.

A figura 4.1 ilustra o desempenho do classificador linear realizando a classificação por limiarização com cada um dos valores de referência utilizados separadamente. Os gráficos contidos nessa figura representam as curvas ROC obtidas em cada uma das configurações. Quanto mais próxima dos eixos X e Y a curva estiver, significa uma maior eficiência do algoritmo. O classificador ideal teria sua curva ROC exatamente em cima dos eixos, de modo que o erro sempre seria zero. Pode-se observar também que a curva é mais alongada no eixo X do que no Y, o que significa que é muito mais difícil de conseguir uma taxa de falsos positivos igual a zero do que uma taxa de falsos negativos igual a zero.

Esses dados, juntamente com dados de energia interna, são combinados em um vetor de classificação utilizando o SVM. Os resultados estão expostos na seção 4.3 e 4.4.

4.2 DTW versus *FastDTW*

Verificou-se que o classificador linear, quanto utilizado em conjunto com o *FastDTW* com valor do raio igual a 3 ou superior, mantém as mesmas taxas de erro do que quando

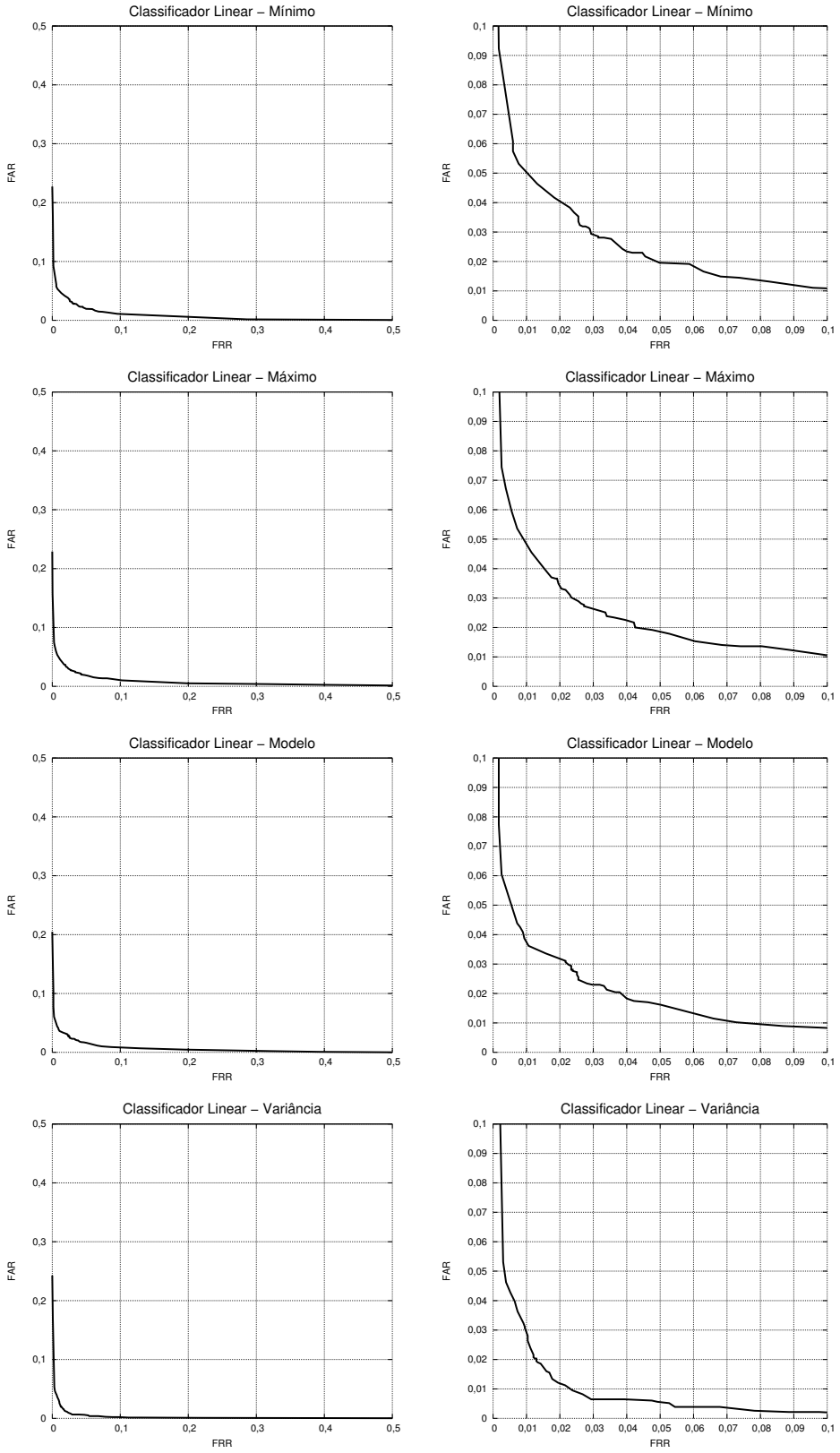

Figura 4.1: Curvas ROC obtidas com uso do classificador linear. Comparação de resultados utilizando diferentes parâmetros como referência ($d_{\min}$, $d_{\max}$, d_{modelo} e *variância* entre os três valores). Cada imagem do lado direito é uma ampliação do primeiro quadrante e do gráfico correspondente a sua esquerda.

Tabela 4.1: Comparação de desempenho do classificador SVM utilizando o DTW e o *FastDTW* com diferentes valores de raio.

Método	**FAR(%)**	**FRR(%)**	**TOTAL(%)**
DTW	1,276	1,191	2,467
FastDTW			
Raio = 0	1,787	1,191	2,978
1	1,446	1,276	2,722
3	**1,276**	**1,191**	**2,467**
5	1,276	1,191	2,467
15	1,276	1,191	2,467
30	1,276	1,191	2,467

utilizado com o DTW. Entretanto, notou-se que, em função do volume relativamente pequeno de dados processados para a realização do processo de verificação, utilizando o algoritmo proposto, e a crescente capacidade de memória e de processamento dos computadores, não se faz necessária a utilização do *FastDTW*, O DTW executa com velocidade satisfatória nos *hardware* hoje existentes no mercado.

Estas considerações não levam em conta a disponibilização do serviço em grande escala, visto que não é objetivo deste trabalho implementar um serviço escalável e performático. Não foram feitas medições de qualquer tipo, com relação a tempo de execução, dos algoritmos em cada um dos casos.

A tabela 4.1 compara a performance do classificador linear nas duas situações: utilizando o DTW e utilizando o *FastDTW* ambos com diferentes valores de raio. Ficou evidenciando que valores de raio maiores do que 3 não melhoram as taxas de erro do algoritmo de verificação, e indicando que o *FastDTW* retornou valores muito similares ao DTW nestes casos.

4.3 Uso de informações de Pressão

Com relação ao uso de pressão, verificou-se uma melhora no desempenho do classificador SVM quando alimentado com essas informações de diferentes partições de uma assinatura, conforme descrito na seção 3.3.2.

A tabela 4.2 demonstra o desempenho do classificador SVM utilizando diferentes quantidades de partições (distintas e de igual tamanho) da assinatura, para a extração de informações de pressão. Observou-se que o melhor desempenho do classificador aconteceu com a utilização de 32 partições, visto que, com a utilização de quantidade de partições maior que 32, o número de falsos negativos (FRR) cresce mais rapidamente e que o número de falsos positivos (FAR) decai.

A figura 4.2 ilustra os valores médios de pressão, normalizados, encontrados em cada partição das assinaturas do *Visual Subcorpus* do SUSIG. Os pontos em azul pertencem a assinaturas autênticas, e os vermelhos a falsificações. É possível notar claramente a concentração dos pontos azuis entre os valores 0 e 2. Já os pontos vermelhos apresentam

Tabela 4.2: Comparação de desempenho do classificador SVM utilizando diferentes quantidades de partições para o cálculo de "energia interna".

	Taxas de erro		
Partições	**FAR(%)**	**FRR(%)**	**TOTAL(%)**
1	1,652	1,826	3,478
2	1,739	1,391	3,130
4	1,652	0,782	2,434
8	1,276	1,021	2,297
16	1,361	1,276	2,637
32	**1,276**	**1,119**	**2,395**
48	1,191	1,531	2,722
64	1,021	2,723	3,744

um espalhamento maior, indicando inconsistência dos valores de pressão da assinatura da amostra em relação aos encontrados no conjunto de referência.

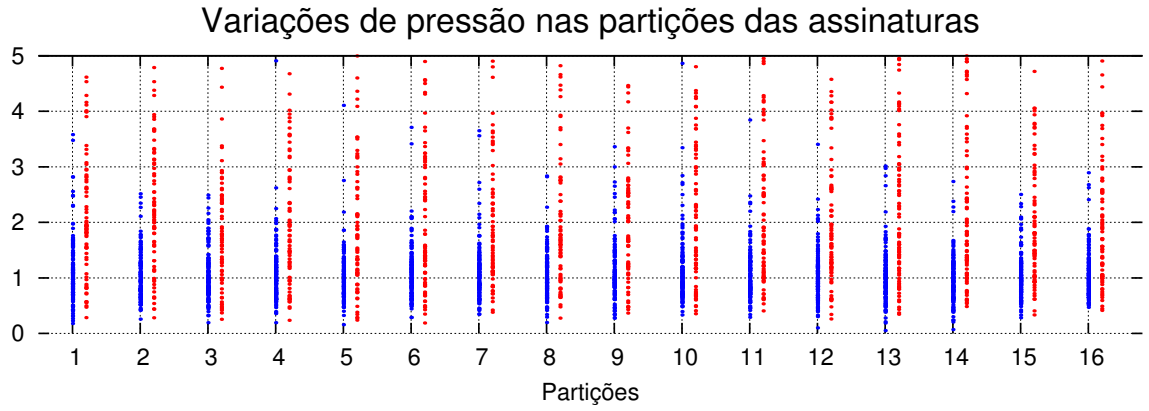

Figura 4.2: Variações da energia interna das assinaturas nas partições. Os valores em X representam as partições. Os pontos em azul representam assinaturas autênticas, e os pontos em vermelho pertencem a falsificações. Note como os pontos em azul se concentram no intervalo [0,2].

4.4 Eficiência do algoritmo

Apesar do classificador linear ter alcançado um resultado razoavelmente bom na separação das assinaturas autênticas das falsificações, o classificador SVM teve o melhor desempenho. Deve-se ressaltar que o classificador SVM permite a utilização do vetor de características mais complexo, avaliando uma quantidade maior de informações.

A tabela 4.3 contém os resultados obtidos com o classificador linear. É apresentado apenas o EER obtido com cada um dos parâmetros de referência, pois o EER é um bom indicador da eficiência geral do algoritmo, visto que representa uma situação onde EER =

Tabela 4.3: Comparação de desempenho do classificador Linear utilizando diferentes parâmetros. como referência ($d_{\min}$, $d_{\max}$, d_{modelo} e *variância* entre os três valores).

Método	**EER (%)**
Mínimo	2,936
Máximo	2,723
Modelo	2,553
Variância	**1,361**

FAR = FRR. Alterando os valores limite utilizados no classificador linear é possível obter um FAR ou um FRR tão pequeno quanto se queira, porém isso implica em um grande aumento do erro total do alorritmia (soma do FAR com o FRR).

O menor erro total alcançado com o SVM foi de 2,395% (FAR=1,276% e FRR=1,119%), tabela 4.2. Ao aumentar o número de partições para avaliação de pressão reduziu-se o FAR para 1,021% , porém o erro total aumentou para 3,744%.

Com o classificador linear, o melhor EER conseguido foi de 1,361% (tabela 4.3), que está ligeiramente acima do SVM. Esses resultados foram obtidos utilizando-se a Variância como valor de referência para o classificador. Ajustando-se o classificador, foi possível conseguir um de FAR=0,936% associado a um FRR=1,87%, cuja taxa de crescimento é mais amena que a encontrada no classificador SVM.

Capítulo 5

SignRec - Descrição da Implementação Realizada

Neste capítulo descreveremos brevemente a implementação realizada para possibilitar a coleta e criação de um banco de dados de amostras de assinaturas. Este sistema, denominado "SignRec" está integrado com os algoritmos de verificação, utilizando-se tanto do classificador linear quanto do SVM para demonstração do processo de verificação de uma assinatura. Elencaremos as tecnologias, *framework's*[1] e API's[2] utilizadas na implementação, bem como o modelo de dados utilizado para persistência das informações em banco de dados. Mostraremos também, o passo-a-passo para a configuração do ambiente necessário para executar a aplicação. Por fim, veremos exemplos da execução dos casos de uso "Cadastrar usuário" e "Validar assinatura".

5.1 Introdução

O sistema "SignRec" foi desenvolvido para dois principais propósitos: possibilitar a construção de uma base de dados de amostras de assinaturas de modo que pudéssemos utilizá-la para realizar experimentos e verificar acurácia do algoritmo de verificação implementado; o segundo propósito principal é demonstrar de modo prático o algoritmo em funcionamento.

Devido ao fato de termos conseguido uma base de dados de assinaturas pronta (SUSIG) para a realização de testes e validação do algoritmo, não foi necessário construirmos uma base própria.

O sistema foi construído em Java, com o auxílio de diversas API's, bibliotecas e tecnologias livres de modo que ele é compatível com os sistemas operacionais Linux, Mac OS

[1] Um ***framework***, ou arcabouço, em desenvolvimento de software, é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica. Um *framework* pode atingir uma funcionalidade específica, por configuração, durante a programação de uma aplicação. Ao contrário das bibliotecas, é o *framework* quem dita o fluxo de controle da aplicação, chamado de Inversão de Controle.

[2] **API**, de ***Application Programming Interface*** (ou Interface de Programação de Aplicações) é um conjunto de rotinas e padrões estabelecidos por um *software* para a utilização das suas funcionalidades por programas aplicativos que não querem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços. Por exemplo, um sistema operacional possui uma grande quantidade de funções na API, que permitem ao programador criar janelas, acessar arquivos, criptografar dados, etc.

X e Windows. Aprofundaremos este assunto na seção seguinte. A maior parte dos testes foram executados no Mac OS X, embora tenhamos executando com sucesso o sistema no Linux e Windows.

5.2 Arquitetura e estrutura do projeto

Conforme mencionado na seção anterior o sistema foi construído utilizando Java, em conjunto com diversas API's e *framework's* e tecnologias abertas, disponíveis gratuitamente na internet. São eles:

JPen
: uma API que implementa as bibliotecas em código nativo para a comunicação com os *driver's* das mesas digitalizadores, possibilitando a recuperação dos dados capturados por elas diretamente por meio de API's Java; é a peça chave do sistema possibilitando sua integração de forma simples com o hardware necessário para a execução das tarefas propostas;

OpenSwing
: um *framework* para construção de aplicações RIA [5] (*Rich Internet Aplications*[3]) em camadas, baseadas no *Swing*[4] , o que possibilita a construção de interfaces para aplicativos *desktop* (que é o nosso caso);

JFreeChart
: uma API para renderização de gráficos em interfaces *Swing/AWT*;

MySQL
: sistema gerenciador de banco de dados relacionais, escolhido para realizar a persistência dos dados de nosso sistema;

libsvm
: uma implementação de código aberto de SVM com API's para treinamento e validação de dados em diversas linguagens de programação;

FastDTW
: implementação dos algoritmos DTW e FastDTW realizadas pelos próprios criadores do FastDTW (tivemos que realizar algumas adaptações nas API's para que pudéssemos integrá-las melhor com o sistema);

Podemos dizer que o OpenSwing é o esqueleto da aplicação fornecendo uma arquitetura em três camadas juntamente com a maior parte dos componentes necessários para a geração de telas e demais recursos de apresentação do sistema. Os demais componentes são acessados por meio de interfaces disponibilizadas ao usuário, construídas com o OpenSwing.

[3] ***Rich Internet Application*** (Aplicações Ricas para Internet) são aplicações web que tem características e funcionalidades de *software* tradicionais do tipo *desktop*. Em geral transferem todo o processamento da interface para o navegador da internet, porém mantém a maior parte dos dados (estado do programa, banco de dados) no servidor de aplicação.

[4] ***Swing*** é o principal conjunto de ferramentas para construção de interfaces gráficas em Java. Compõe parte do JFC (*Java Foundation Classes*).

5.2.1 Estrutura do Projeto

O sistema é fornecido como um pacote JAR executável e como um projeto do Eclipse[5]. Sua estrutura de arquivos é a seguinte:

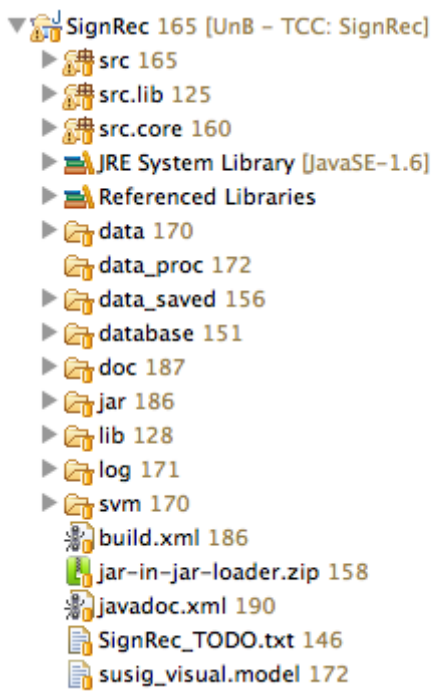

Figura 5.1: Estrutura de diretórios do projeto do Eclipse.

Os diretórios **src**, **src.lib** e **src.core** contém os arquivos-fonte do projeto, de modo que:

src
: contém os arquivos que estendem o OpenSwing e são responsáveis pela renderização das interfaces do sistema e persistência de dados; nele encontra se a classe `Menu.java`, pertencente ao pacote "`main`", é a classe de entrada que inicia a aplicação;

src.lib
: contém os fontes de bibliotecas de terceiros acopladas ao sistema, como é o caso do FastDTW e da libsvm;

src.core
: contém os componentes principais implementados para os sistema como é o caso das entidades que estruturam os dados de assinaturas; os algoritmos para verificação, cadastro, comparação e classificação; classes utilitárias como é o caso do `GerenciadorTeste.java`, utilizado para verificar a performance do algoritmo frente

[5] **Eclipse** é um IDE (*Integrated Development Environment* ou Ambiente Integrado de Desenvolvimento) desenvolvido em sua maior parte usando Java, utilizado para o desenvolvimento de aplicações em diversas linguagens de programação.

a uma determinada massa de dados; o `VizualizadorAssiantura.java` utilizado para renderizar a imagem de uma assinatura a partir dos dados capturados pela mesa digitalizadora; no pacote "`assinador`" temos a classe `Assinador.java`, responsável por possibilitar renderizar a tela e instanciar os componentes necessários para a captura de assinaturas.

Os demais diretórios contém outros recursos necessários para o funcionamento do sistema.

data
: é utilizado para armazenar as assinaturas do SUSIG utilizadas para realizar os testes com o algoritmo;

data_proc
: contém os mesmos dados armazenados em "`data`" após a realização do pré-processamento, necessário para compatibilizar os formatos de arquivo com o formato utilizado pelo sistema;

data_saved
: contém dados de assinaturas capturadas pelo SignRec e exportadas pelo usuário;

database
: contém o modelo de entidade e relacionamento utilizado para criação do banco de dados relacional, bem como *script's* SQL para a geração do banco de dados;

doc
: contém os arquivos do Javadoc[6] gerado pelo script do Apache Ant[7] `javadoc.xml`;

jar
: contém a distribuição do sistema empacotada em um arquivo JAR, juntamente com suas dependências;

lib
: contém os arquivos binários de dependências externas do sistema;

log
: diretório utilizado para salvar os arquivos de registro e resultados das execuções de testes;

svm
: contém os arquivos distribuídos com a *libsvm* necessários para realizar o treinamento da rede neural;

- `build.xml` - Script ANT utilizado para gerar a versão empacotada (JAR);

[6] **Javadoc** é um gerador de documentação criado pela Sun Microsystems para documentar a API dos programas em Java, a partir do código-fonte. A documentação é gerada no formato HTML.

[7] **Apache Ant** é uma ferramenta criada para automatizar tarefas do processo de construção de *software*. Similar ao Make, foi originalmente criado para ser utilizado com Java. Utiliza arquivos XML para descrever seus processo de construção.

- Arquivos "`*.model`" são redes SVM treinadas para utilização pelo sistema;
- O arquivo `SignRec_TODO.txt` contém a lista de alguns itens para implementação futura.

5.2.2 Modelo de Dados

Optamos pela utilização de um modelo de dados simplificado, tendo em vista que utilizamos uma base de dados pré montada para a realização dos testes de acurácia do algoritmo.

O diagrama MER[8] (figura 5.2) representa o banco de dados utilizado pelo sistema.

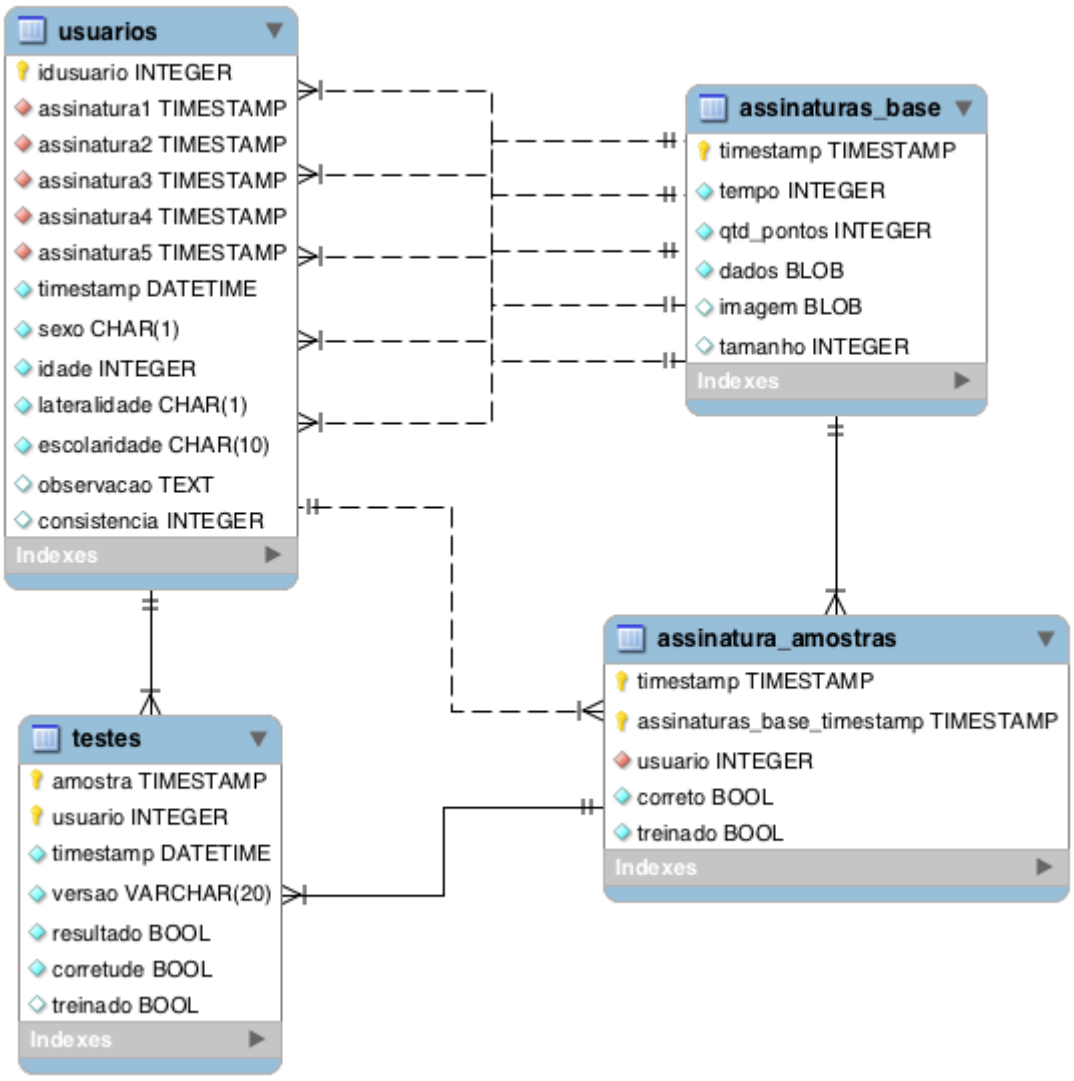

Figura 5.2: Modelo de dados utilizado na implementação do sistema.

[8]**MER - Modelo de Entidades e Relacionamentos** é um modelo abstrato cuja finalidade é descrever, de maneira conceitual, os dados a serem utilizados em um sistema de informações ou que pertencem a um domínio.

Optamos por possibilitar a coleta de alguns dados dos usuários para a realização de estatísticas e tentativa de identificação de padrões em grupos de usuários classificados de acordo com as características observadas como faixa etária, sexo, lateralidade e escolaridade. Para isso utilizamos a tabela usuários.

As assinaturas capturadas são armazenadas na tabela `"assinaturas_base"` e são vinculadas diretamente à tabela `"usuarios"` caso façam parte da composição do conjunto de referência de determinado usuário; caso elas não pertençam ao R_{ID}, são vinculadas a tabela `"assinatura_amostras"` utilizada para identificar assinaturas elegíveis para a realização de testes. Esta tabela armazena informações sobre autenticidade da amostra (campo booleano "correto" marcado como verdadeiro se a assinatura for autêntica e falso no caso contrário) e o indicativo do grau de qualidade de falsificações com o campo "treinado" marcado como verdadeiro, caso o falsificador tenha treinado a execução da assinatura do usuário.

A tabela `"testes"` foi criada com o objetivo de armazenar resultados de testes realizados com as amostras vinculadas à tabela `"assinatura_amostras"`. Esse caso de uso não foi implementado no sistema e está listado como um dos pontos para implementação futura. Os testes realizados com o algoritmo desenvolvido não foram integrados diretamente ao SignRec.

5.3 Instalação e Configuração de Ambiente

5.3.1 Requisitos

Os requisitos mínimos para instalação e utilização do sistema são:

Hardware

- Uma mesa digitalizadora que utilize a porta USB para comunicação com o computador (recomendamos os produtos do fabricante *Wacom* por apresentarem boa qualidade e compatibilidade com diversos sistemas);
- Um computador doméstico com configurações mínimas necessárias para rodar o sistema operacional escolhido, juntamente com a Máquina Virtual Java da Oracle. Recomendamos ao menos o uso de um processador de dois núcleos com 1GB de memória RAM para maior disponibilidade de recursos e melhor experiência do usuário.

Software

- Demostraremos a instalação em sitema operacional Mac OS X versão *Snow Leopard* (10.6.5). Ressaltamos que ele foi executado com sucesso também no Linux (2.6) e Windows (XP, Vista e 7);
- Máquina Virtual Java (JRE) distribuída pela Oracle, versão 6 ou superior (para Linux e Windows, no caso do Mac a Apple utiliza uma distribuição customizada que acompanha o sistema operacional na versão 10.6.5);

- *Driver* correspondente da mesa digitalizadora utilizada, específico para a versão do seu sistema operacional. Utilize preferencialmente a versão distribuída pelo fabricante do dispositivo, ou similar. MySQL Server 5.x ou superior;

5.3.2 Configurando o ambiente

Vamos demonstrar o passo-a-passo da instalação no Mac. Para evitar a instalação do MySQL como um serviço do sistema operacional e facilitar a distribuição do banco de dados pré-montado criamos uma máquina virtual utilizando o *Oracle Virtualbox* v3.2.12 onde instalamos uma distribuição Linux leve (distribuição DSL), juntamente com o servidor de banco de dados MySQL, com o banco de dados e tabelas já criados. Portanto, haverá uma sequência de passos para aqueles que preferirem instalar o servidor diretamente como um serviço do sistema operacional e outro para os que preferirem virtualizar o servidor de banco de dados com o auxílio do Virtualbox.

Primeiramente instale a máquina virtual java e os *driver's* da mesa digitalizadora. Em seguida configuraremos o banco de dados.

Configurando o Banco de Dados

A aplicação estabelece um ordem de prioridade para realizar o conexão com o banco de dados. Como primeira opção é realizada uma tentativa de estabelecer conexão com o servidor local. Caso essa tentativa falhe, a aplicação tenta conectar com o servidor, mapeado no arquivo "`hosts`" do sistema operacional, pelo endereço "signrec.database" (que é o caso do servidor virtualizado, como veremos a seguir).

Instalando o MySQL como um serviço do sistema operacional

1. Utilize o executável do MySQL para instalá-lo no sistema. No Mac OS X você pode utilizar o arquivo "`mysql-5.5.8-osx10.6-x86.dmg`" que está gravado no diretório "Programas" do DVD. Instale tanto o "`mysql-5.5.8-osx10.6-x86.pkg`" quanto o "`MySQLStartupItem.pkg`" e o "`MySQL.prefPane`";

    1.1. Ao que parece, existe um problema no instalador desta versão do MySQL (5.5.8) para Mac OS, portanto para que o banco de dados funcione corretamente é necessário realizar os procedimentos a seguir (no aplicativo Terminal):

        1.1.1. `shell> sudo nano /usr/local/mysql/support-files/mysql.server` (Encontre a configuração que define o "basedir" e altere-o conforme abaixo)

        1.1.2. `basedir=/usr/local/mysql`

        1.1.3. `datadir=/usr/local/mysql/data`

2. No Terminal, utilize os comandos a seguir para inicializar o MySQL:

    2.1. `shell> sudo /Library/StartupItems/MySQLCOM/MySQLCOM start` (Digite sua senha caso necessário)

3. Defina a senha do usuário "root" do MySQL utilizando os comandos a seguir:

3.1. `shell> sudo ln -s /usr/local/mysql/bin/mysqladmin /usr/bin/`
(Digite sua senha caso necessário)

3.2. `shell> mysqladmin -u root password <password>`

3.3. `mysqladmin -u root -h ‘hostname‘ password <password>`
(Tenha certeza que o comando hostname na segunda linha está entre crases (‘), assim o shell pode substituí-lo pelo nome da sua máquina)

4. Para disponibilizar o binário "`mysql`" no arquivo de recursos do shell, facilitando o acesso por linha de comando utilize o comando a seguir:

4.1. `shell> sudo ln -s /usr/local/mysql/bin/mysql /usr/bin/`

4.2. `(Digite sua senha caso necessário)`

5. Copie o arquivo `SignRec/database/modelo_dados.sql` para o diretório home do seu usuário "`/Users/USUARIO/`";
(Ex.: `/Users/augusto/`)

6. No Terminal utilize o comando "`cd`" para retornar ao diretório home do seu usuário;

7. Utilize o comando "`pwd`" para verificar se você está no diretório correto;

8. Utilize o comando "`ls`" para verificar se o arquivo "`modelo_dados.sql`" encontra-se neste diretório;

9. Conecte com o servidor de banco de dados e crie o banco de dados utilizando os comandos a seguir:

9.1. `shell> mysql localhost -u root -p`
(Digite a senha definida para o usuário root)

9.2. `mysql> create database signrec;`

9.3. `mysql> use signrec`

9.4. `mysql> \. ./modelo_dados.sql`

10. Seu banco de dados está criado. Agora você precisa configurar as permissões de acesso.

10.1. `mysql> grant all privileges on signrec.* to signrec@localhost identified by 'signrec';`
(Caso queira instalar o mysql em um servidor remoto troque *localhost* por '%' com as aspas simples, da forma como está escrito)

10.2. `mysql> exit`

11. Seu banco de dados está corretamente configurado para ser acessado pelo SignRec.

Virtualizando o servidor de banco de dados com o *Virtualbox*

1. Instale o *Virtualbox* a partir do pacote distribuído pelo desenvolvedor ou utilize o pacote que encontra-se no CD que acompanha este trabalho;

2. Execute o aplicativo *Virtualbox*;

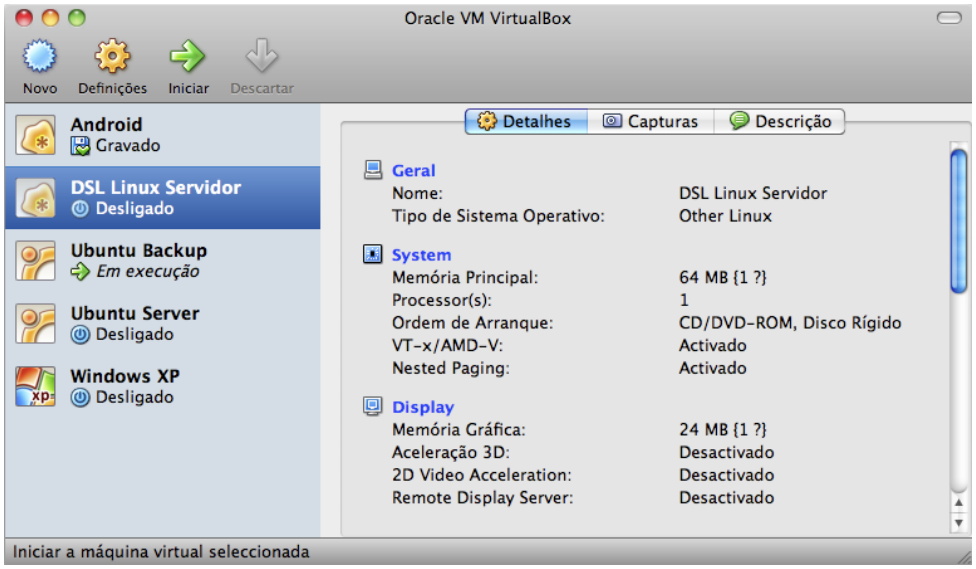

Figura 5.3: Tela inicial do aplicativo *Oracle VM VirtualBox*.

3. Será carregada uma interface similar à ilustrada na figura 5.3
4. Criaremos uma nova máquina virtual. Clique na opção "Novo" localizada no canto superior esquerdo da tela do aplicativo (será aberta uma janela similar à ilustrada na figura 5.4);
5. Clique em "*Continue*";

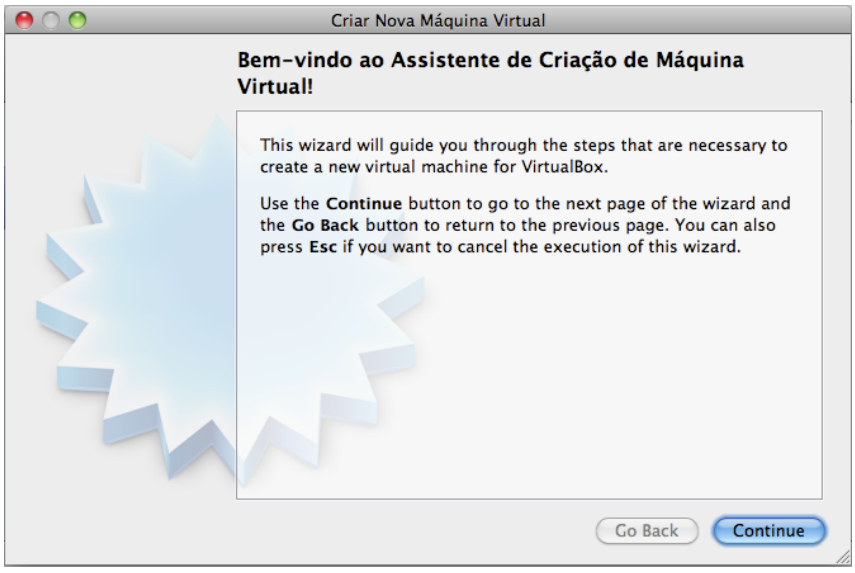

Figura 5.4: Criando uma nova máquina virtual no aplicativo *Oracle VM VirtualBox*.

6. Na tela seguinte escolha um nome para a sua máquina virtual (sugestão: "DSL *Linux* Servidor"). Marque *Linux* como sistema operacional e na versão escolha "*Other Linux*" (figura 5.5);
7. Clique em "*Continue*";

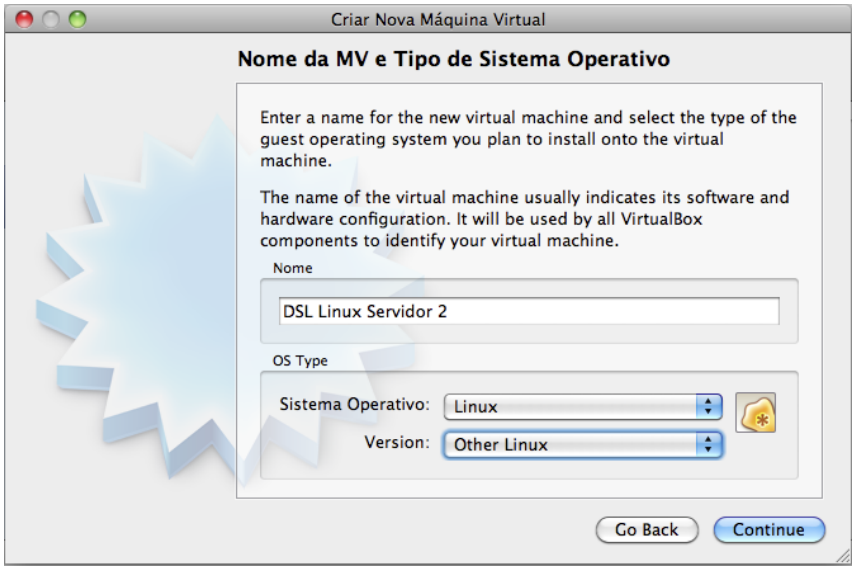

Figura 5.5: Determinando o sistema operacional que será instalado na máquina virtual.

8. Na tela seguinte, selecione 64MB de memória RAM para nossa máquina virtual (figura 5.6);
9. Clique em "*Continue*";

Figura 5.6: Selecionando a quantidade de memória RAM para nossa VM.

10. Marque a opção "*Boot Hard Disk*";
11. Selecione "*Use existing hard disk*";
12. Selecione o disco virtual "`DSL Linux Servidor.vdi`" distribuído no CD que acompanha este trabalho (figura 5.7);
13. Clique em "*Continue*";

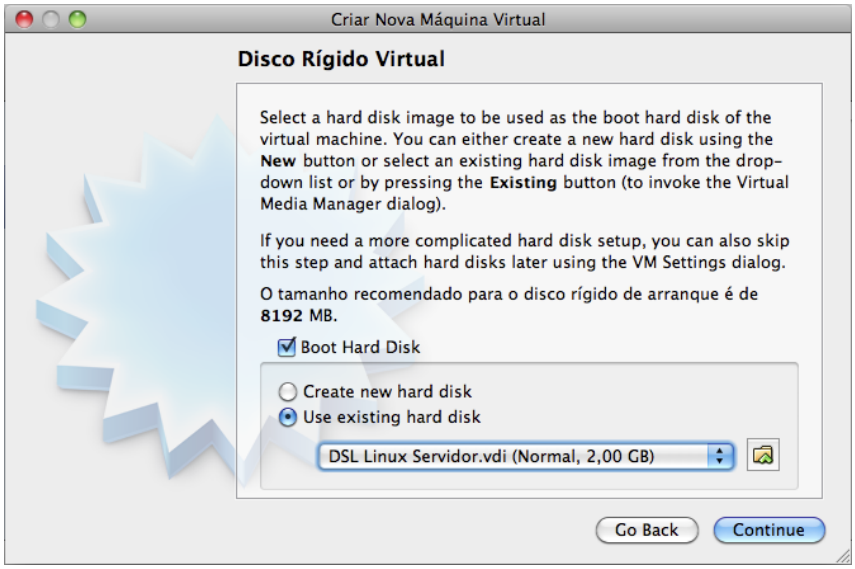

Figura 5.7: Selecionando o disco virtual que será utilizado pela VM.

14. Clique em "*Done*".

Sua máquina virtual já está criada. Falta apenas ajustar a configuração de rede para que você possa acessar o servidor de banco de dados virtualizado.

15. Selecione a máquina virtual que você acaba de criar e clique na opção "Definições" (representada pela engrenagem amarela no canto superior esquerdo da janela principal do *Virtualbox*);
16. Será aberta uma janela similar à ilustrada na figura 5.8;
17. Selecione a opção "Rede";

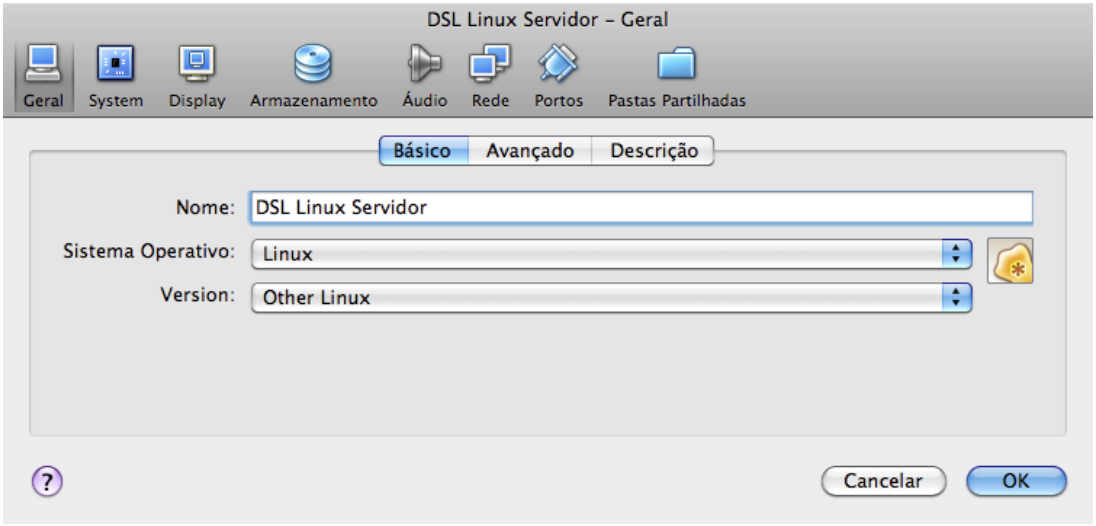

Figura 5.8: Alterando as configurações da máquina virtual.

18. Marque a opção “Activar Adaptador de Rede”;
19. No item “Associado a” marque a opção “*Host-only Adapter*” conforme a figura 5.9;
20. Clique em “*Ok*”;

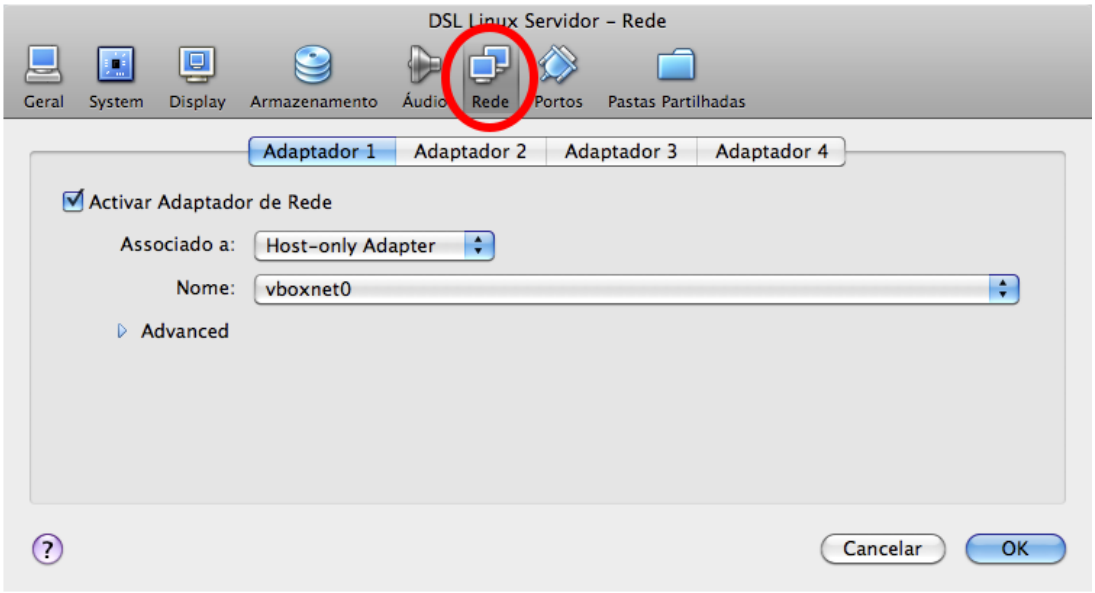

Figura 5.9: Alterando as definições de Rede para a VM.

Você acaba de ajustar a interface de rede correta para que possamos acessar a máquina virtual. Ainda temos que informar ao SignRec o número do IP atribuído a sua máquina virtual.

21. Inicialize a máquina virtual com um duplo clique sobre ela na janela principal do *Virtualbox*.
22. Quando sua máquina virtual estiver inicializada você verá uma tela similar à figura 5.10;
23. Efetue o login na máquina virtual utilizando os dados:
    23.1. Usuário: dsl
    23.2. Senha: (em branco)
24. Digite o comando “`ifconfig`” para podermos confirmar o endereço IP da VM;
25. Confirme se o endereço IP é 192.168.56.101. Ele está localizado na interface “eth0” campo “inet addr” na segunda linha (figura 5.11)
26. Volte para o sistema nativo e abra o aplicativo “Terminal” (figura 5.12);
27. Edite o arquivo “`/private/etc/hosts`” incluindo a seguinte linha ao final “192.168.56.101 signrec.database”; substitua o IP, caso seja diferente, pelo que você consultou nos passos anteriores (lembre-se de que as aspas não fazem parte do conteúdo que deve ser inserido no arquivo); você pode utilizar o aplicativo "nano"para editar o arquivo com o seguinte comando “`sudo nano /private/etc/hosts`”;

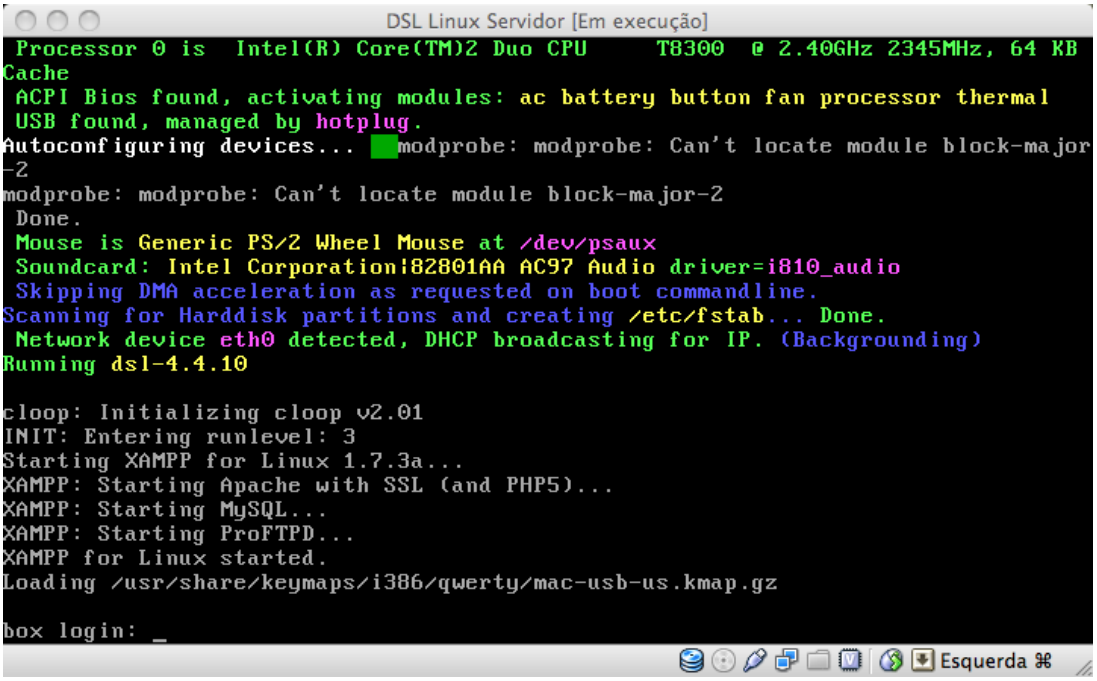

Figura 5.10: Tela de entrada do *DSL Linux* após a inicialização.

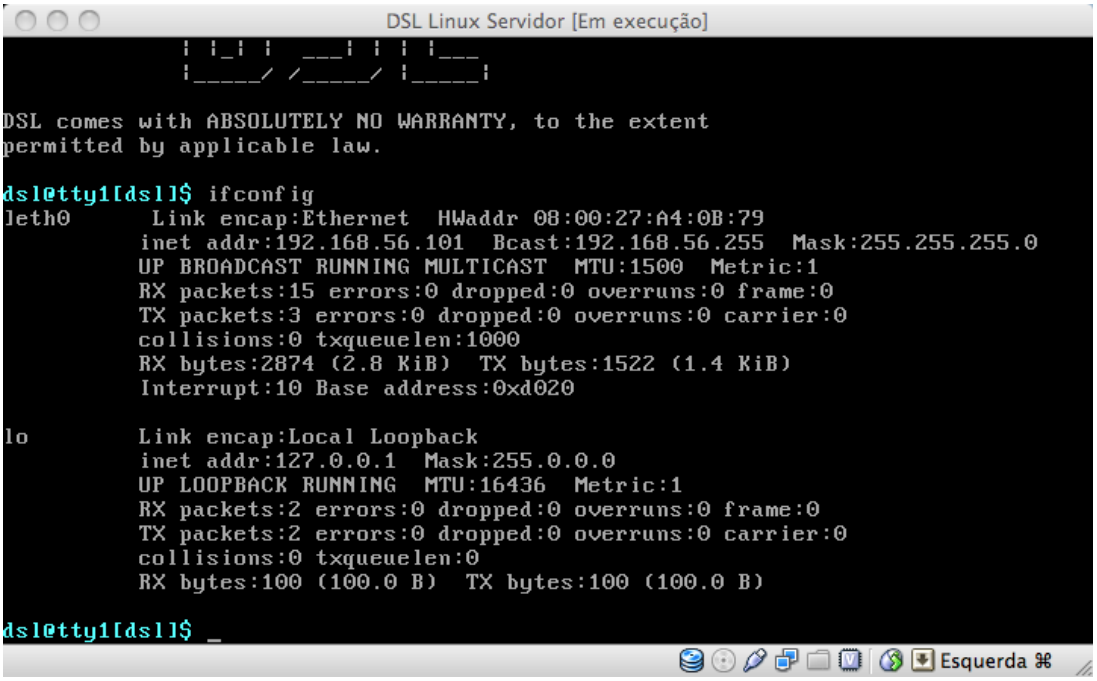

Figura 5.11: Resultado do comando "`ifconfig`" no *DSL Linux*.

27.1. Será solicitada a senha do administrador do sistema;

27.2. Ao final seu arquivo "`hosts`" estará parecido com o da figura 5.13;

27.3. Utilize Ctrl+X para sair;

27.4. O aplicativo perguntará se você deseja salvar as alterações; responda "Sim" e tecle "Enter";

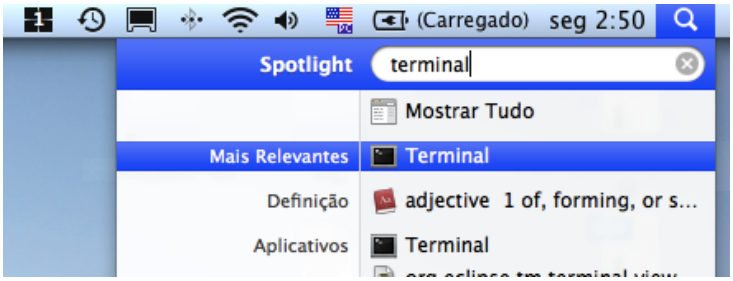

Figura 5.12: Abrindo o aplicativo "Terminal" no Mac OS X.

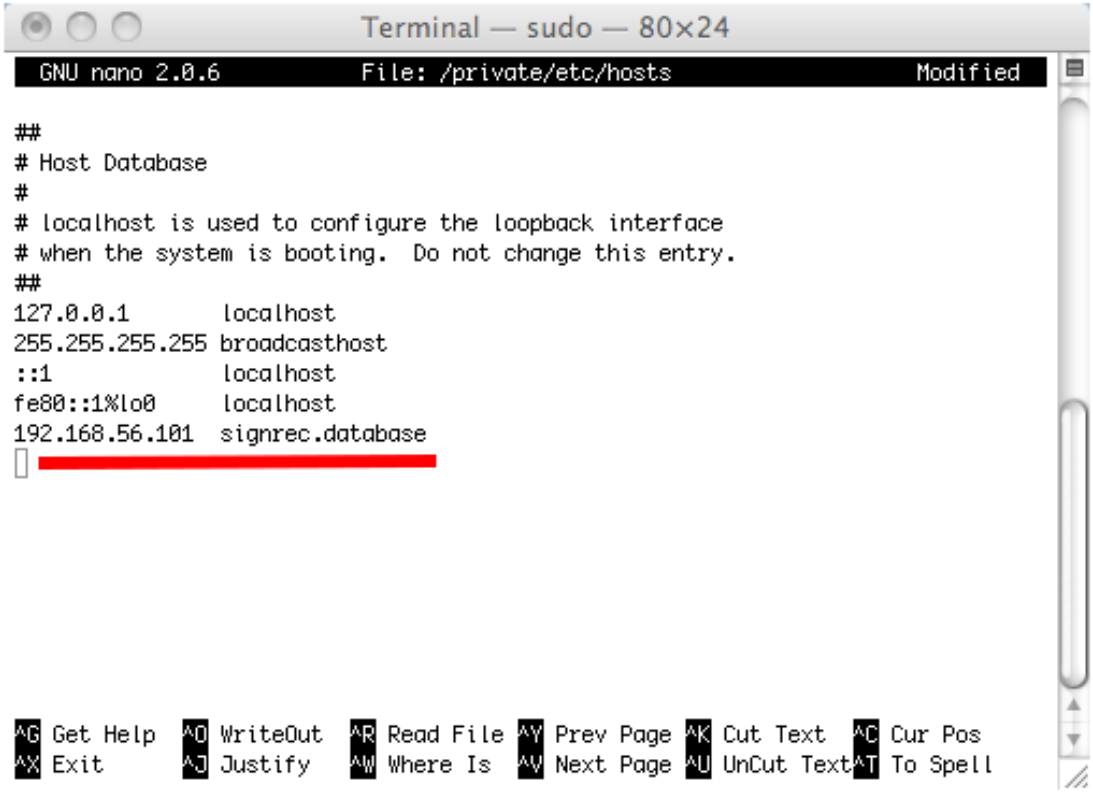

Figura 5.13: Editando o arquivo "`hosts`"do sistema operacional hospedeiro.

27.5. Para testar se as alterações estão em vigor reinicie o aplicativo “Terminal” e digite o comando “`ping signrec.database`” (ele deve retornar o IP da sua VM conforme a figura 5.14); para interromper a execução do ping utilize Ctrl+C;

Seu ambiente agora está configurado e pronto para a execução do SignRec. Inicie aplicativo com um duplo clique sobre o arquivo “`SignRec.jar`” localizado no diretório “`SignRec/jar/`”. Será apresentada uma tela similar à figura 5.15.

5.4 Visão geral do Sistema

Agora que nosso sistema está configurado, vamos realizar a execução de dois casos de uso para vê-lo funcionando na prática. Realizaremos o cadastramento de um usuário e a verificação de autenticidade de uma assinatura para este usuários que cadastramos.

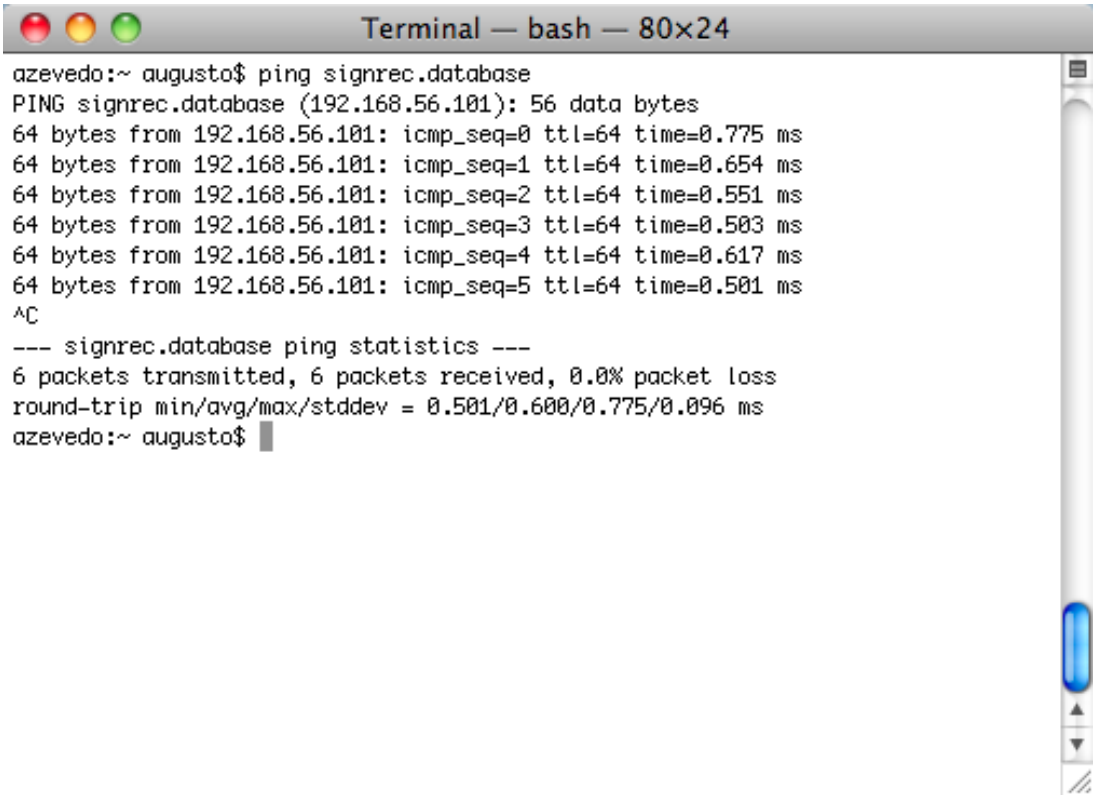

Figura 5.14: Testando o mapeamento de IP realizado.

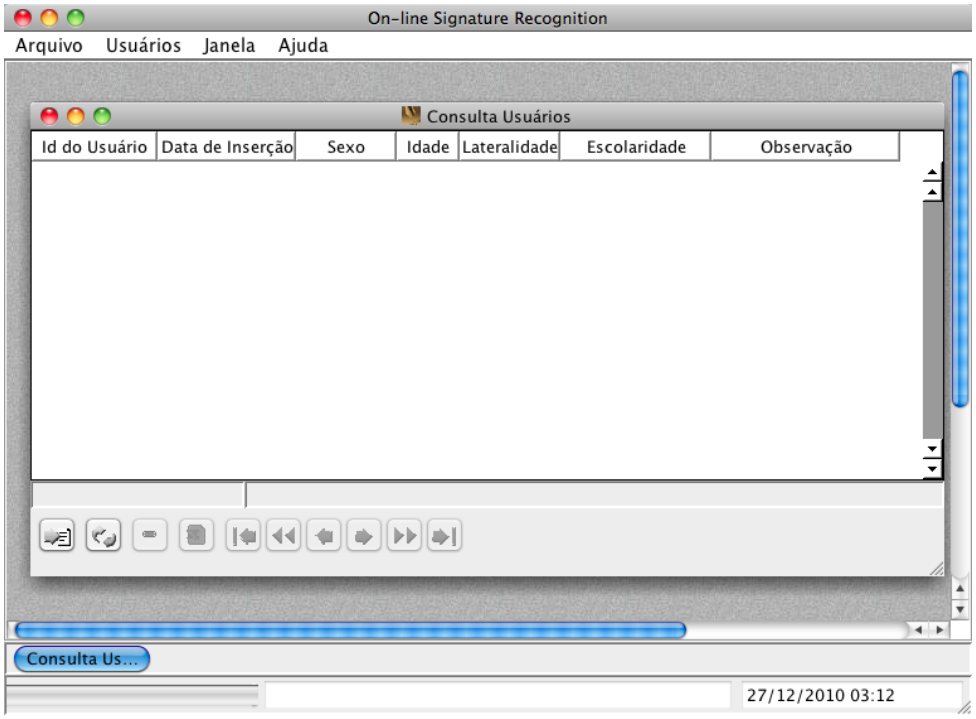

Figura 5.15: Tela inicial do SignRec.

5.4.1 Cadastrando um usuário

Quando o sistema é iniciado é aberta automaticamente a janela "Consulta Usuários". É por meio dela que realizaremos o cadastro. Para isso utilizaremos o botão localizado no canto inferior esquerdo da janela, cujo ícone é uma seta apontando para um formulário (figura 5.16).

Figura 5.16: Botão de inclusão de registro, tela inicial do SignRec.

Ao clicarmos nesse botão será aberta a tela ilustrada na figura 5.17, onde informaremos os dados de usuário que estamos cadastrando. Para coletar as assinaturas basta clicarmos nos botões A1 a A5. Essas serão as assinaturas que comporão o conjunto de referência do usuário que está sendo cadastrado.

Ao acionarmos um botão de captura (A1 - A5), a tela da figura 5.18 será aberta em modo "tela cheia" para possibilitar a captura da assinatura, uma vez que a área da superfície da mesa digitalizadora é mapeada de acordo com a área de exibição do monitor.

Basta que o usuário assine sobre a mesa digitalizadora para que os dados sejam capturados. Oriente o usuário a olhar para a mesa digitalizadora e não para a tela do computador. Caso o usuário cometa algum erro durante o processo de assinatura utilize a tecla "Shift" para descartar os dados e recomeçar o processo. Uma vez que o usuário esteja satisfeito com sua assinatura, tecle "Enter" para salvar e sair. Caso queira desistir apenas tecle "Esc". Note que durante o processo de captura, a assinatura deve ser desenhada na cor preta (figura 5.19). Caso esteja sendo desenhada em azul (figura 5.20), você deve verificar a instalação dos *driver's* da mesa digitalizadora.

Observe também que diferenças de pressão da caneta sobre a mesa digitalizadora fazem com que o diâmetro da linha seja alterado (figura 5.21).

Colete as 5 amostras e depois clique no botão salvar (representado por um disquete, conforme a figura 5.22). Note que o texto dos botões A1-A5 mudam da cor preta para azul após a coleta da assinatura.

Acabamos de adicionar um usuário aos sistema. O próximo passo é realizar a validação de uma assinatura deste usuário.

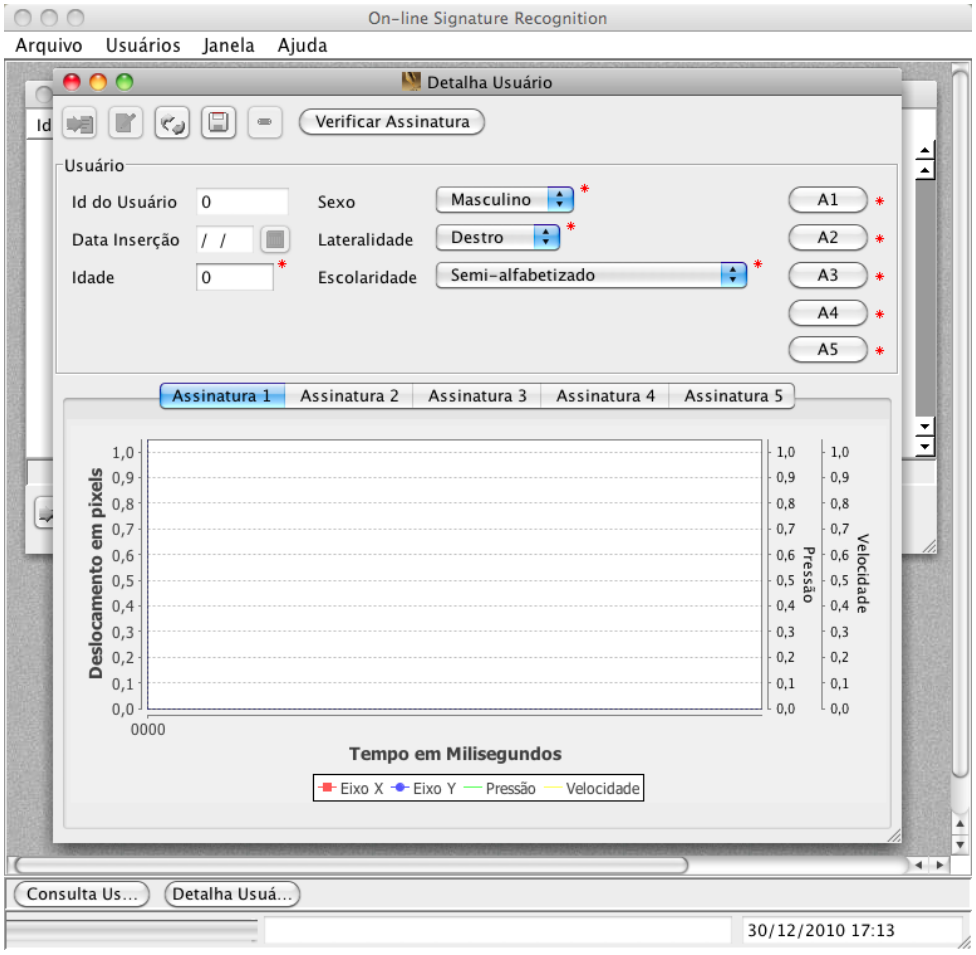

Figura 5.17: Formulário de cadastramento de usuários.

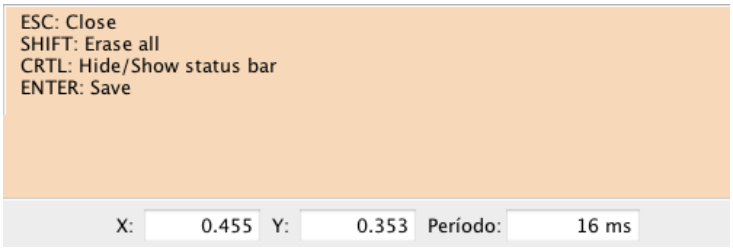

Figura 5.18: Tela de captura de assinatura (será aberta em modo tela cheia).

5.4.2 Validando uma assinatura

Para validar uma assinatura vá à janela "Consulta Usuários" que pode ser alcançada por meio do menu "Usuários > Cadastro de Usuários" (figura 5.23). Efetue um duplo clique sobre o usuário para o qual deseja verificar uma assinatura (figura 5.24). Clique no botão "Verificar Assinatura" (figura 5.25).

Colete a assinatura a ser validada clicando no botão "Coletar" (figura 5.26).

Uma vez coletada clique no botão "Verificar" (figura 5.27). O autenticidade da assinatura será verificada tanto pelo classificador Linear quanto pela SVM de forma independente. O veredicto de cada um dos classificadores será apresentado em região específica

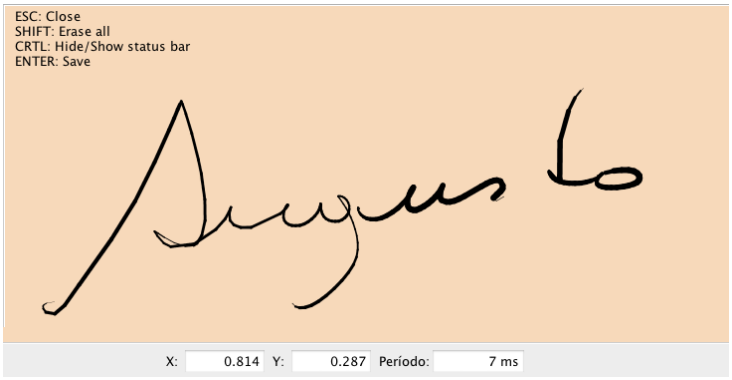

Figura 5.19: Exemplo de captura de assinatura.

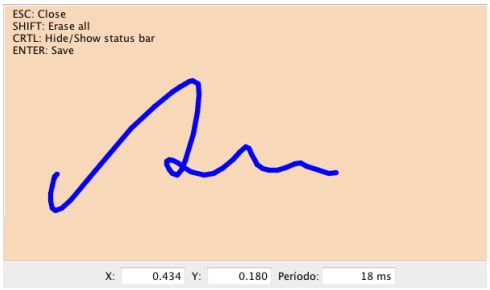

Figura 5.20: Exemplo de falha no carregamento dos *driver's* da mesa digitalizadora. Imagem da assinatura renderizada na cor azul e insensibilidade com relação à variações de pressão.

Figura 5.21: Demonstração do efeito observado na variação de pressão durante a captura de uma assinatura.

da tela, mostrando um retângulo de cor verde caso a assinatura seja classificada como autêntica, e vermelho caso seja classificada como falsa (figuras 5.28, 5.29 e 5.30).

As figura 5.31 ilustra uma funcionalidade do sistema que permite a visualização do movimento suspenso da caneta, ou seja, onde não houve contato com a superfície da mesa digitalizadora, durante o processo de assinatura.

As figura 5.32 o gráfico plotado pelo sistema utilizando as informações da assinatura.

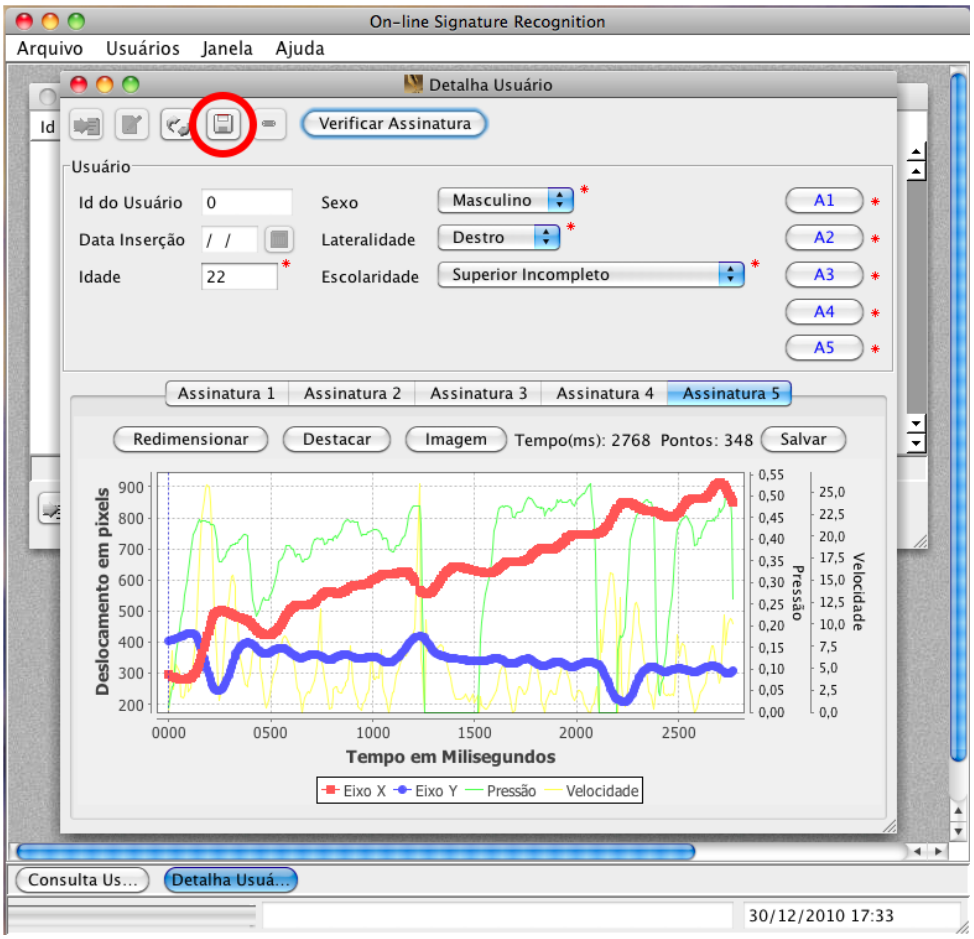

Figura 5.22: Finalizando o cadastro de um usuário. Em destaque, botão "Salvar".

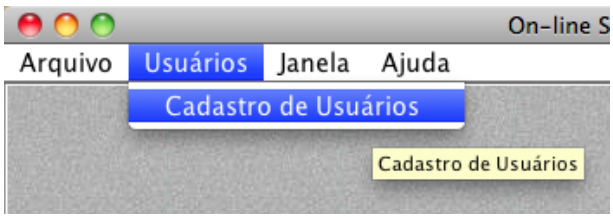

Figura 5.23: Acessando o cadastro de usuários por meio do menu da aplicação.

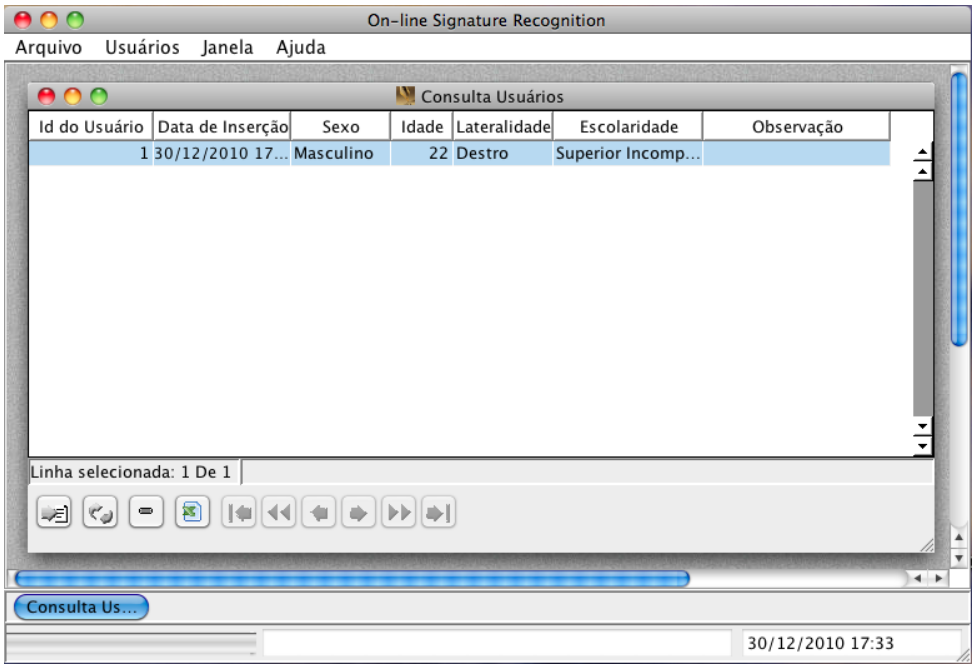

Figura 5.24: Tela de cadastro de usuários.

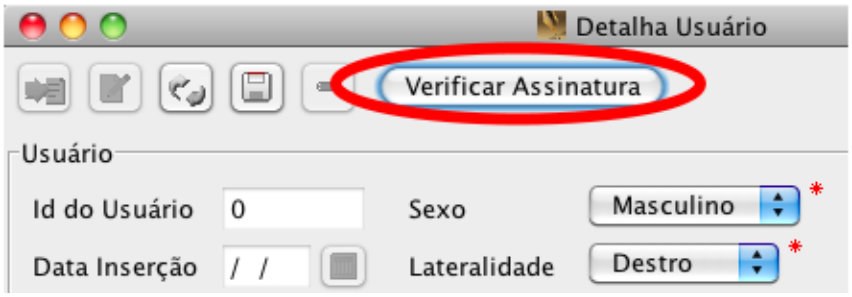

Figura 5.25: Botão para verificar a assinatura de um usuário na tela de detalhamento (figura 5.24).

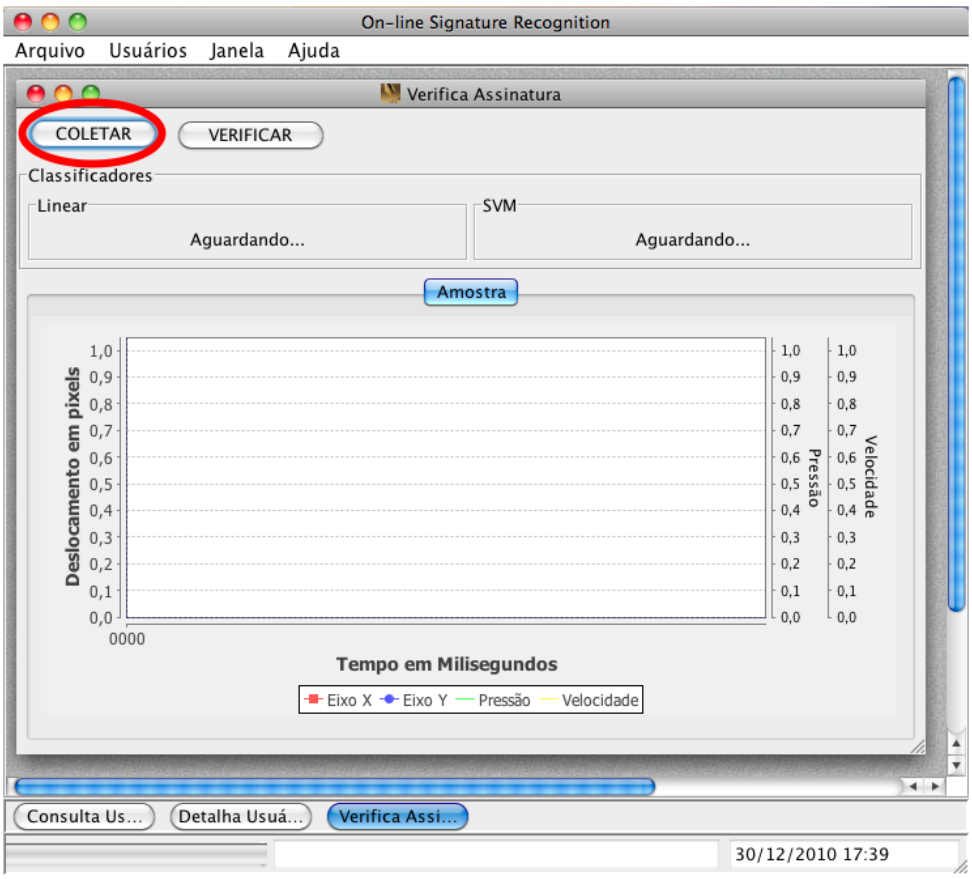

Figura 5.26: Tela para verificação de assinatura. Em destaque, botão para coletar a assinatura a ser verificada.

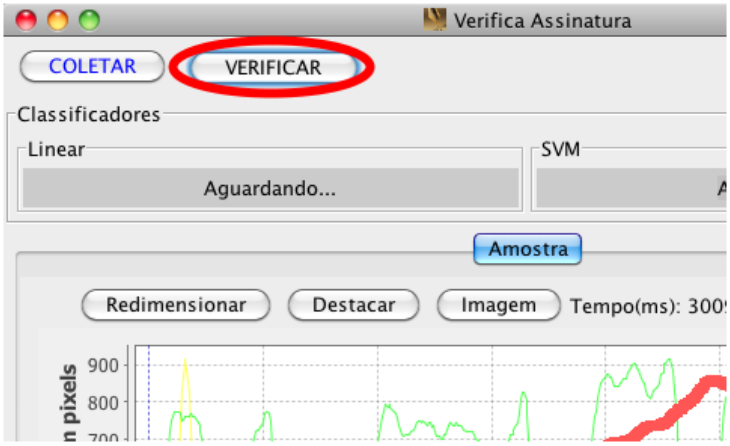

Figura 5.27: O botão "Verificar" inicia a comparação da assinatura coletada com as assinaturas de referência do usuário.

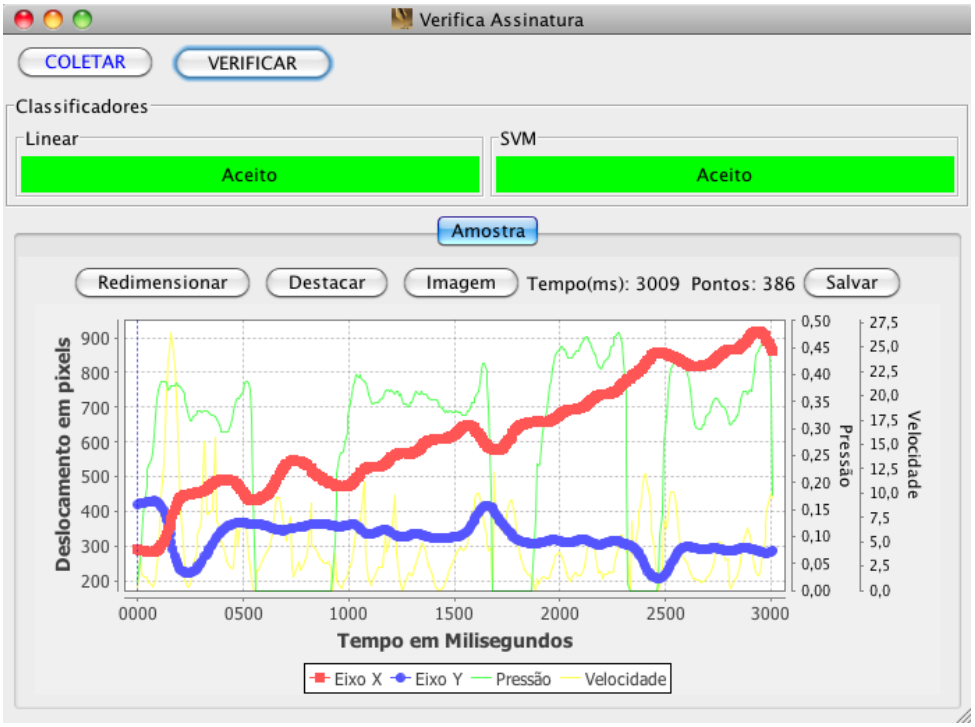

Figura 5.28: Exemplo de assinatura aceita por ambos os classificadores.

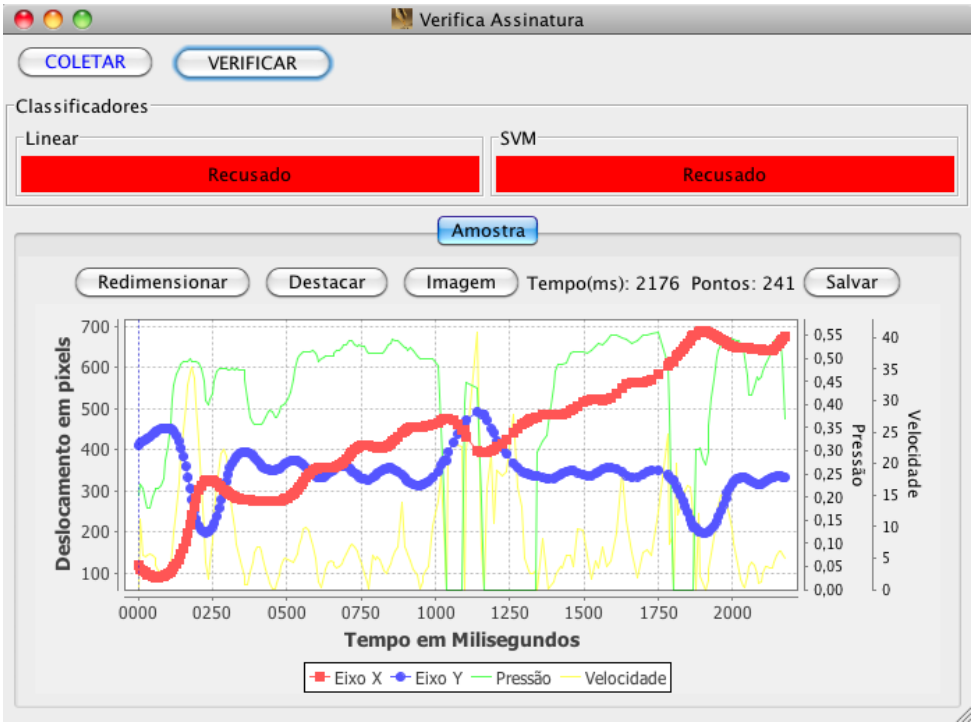

Figura 5.29: Exemplo de assinatura rejeitada por ambos os classificadores.

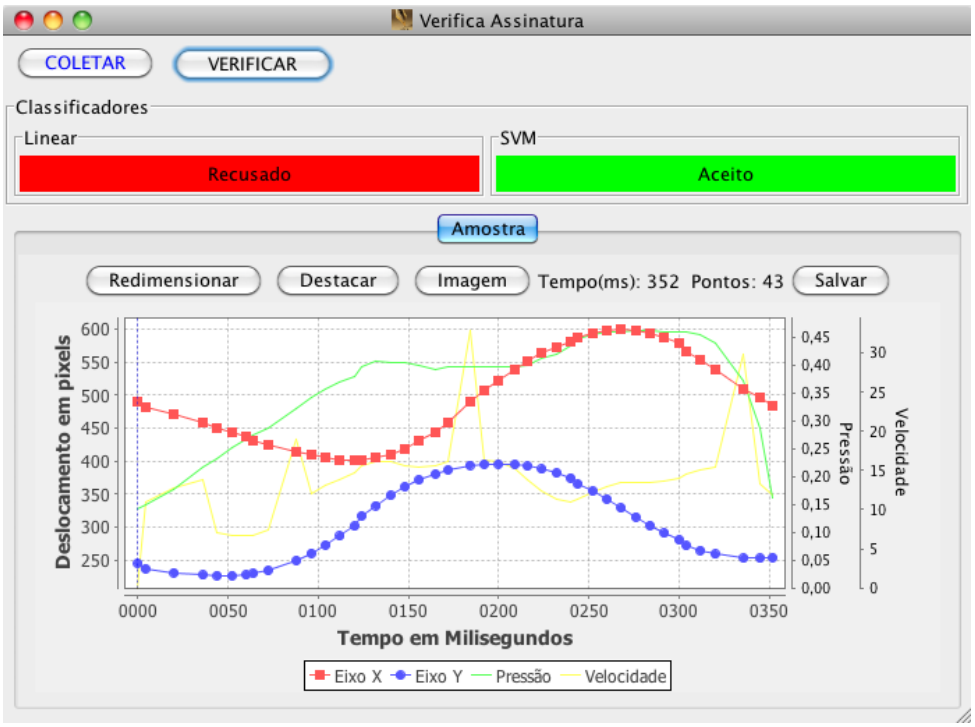

Figura 5.30: Exemplo de assinatura aceita por um classificador e rejeitada pelo outro. É mais comum em casos em que a assinatura é muito simples como é possível notar pelo gráfico plotado na imagem.

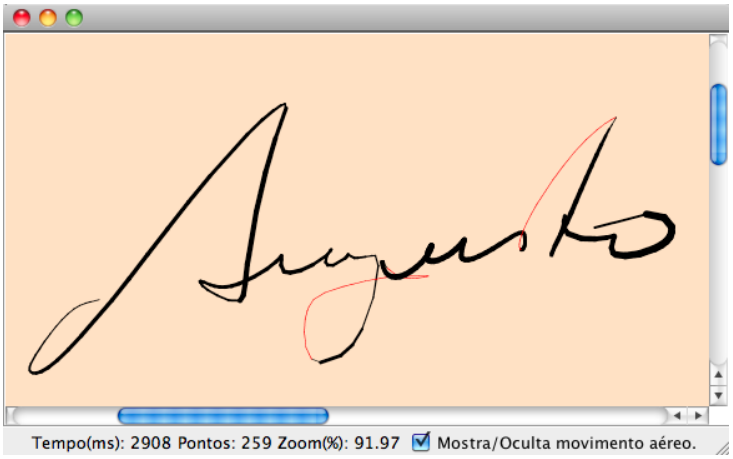

Figura 5.31: Exemplo da visualização de uma assinatura capturada. Note que as linhas em vermelho representam pontos onde não houve contato da caneta com a superfície da mesa digitalizadora.

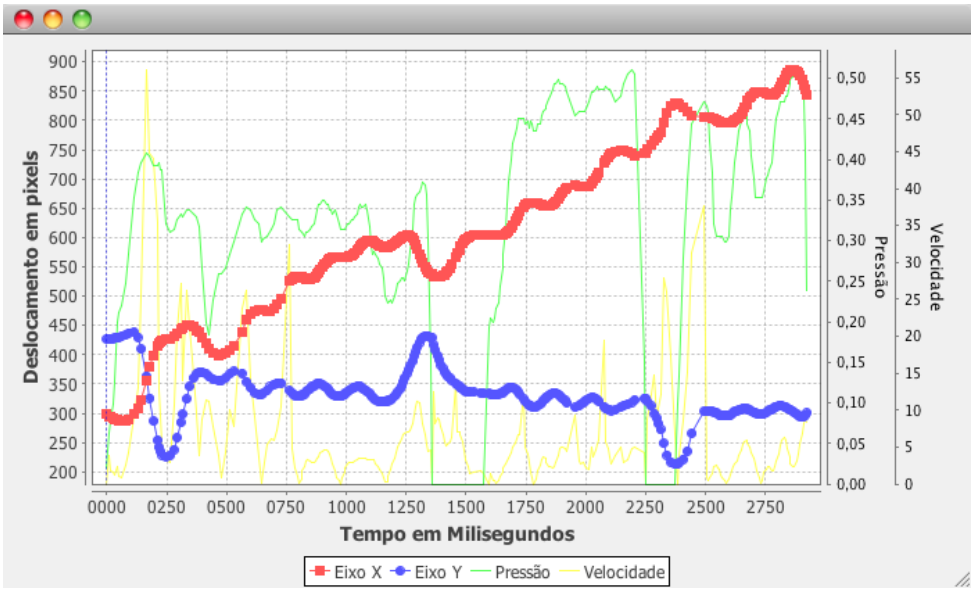

Figura 5.32: Exemplo da visualização do gráfico contendo os valores dos eixos x e y, pressão e um indicador de velocidade ao longo do tempo.

Capítulo 6

Conclusão

Este trabalho apresentou uma proposta de algoritmo de reconhecimento *on-line* de assinaturas, baseado em trabalhos já existentes. Foram feitos experimentos com modificações no algoritmo original, buscando aumentar sua eficiência, principalmente no que diz respeito ao aproveitamento de informações de pressão coletadas durante o processo de assinatura.

Comparou-se também o uso de um classificador linear, bastante limitado em relação ao tamanho conjunto de dados de entrada utilizados para tomada de decisão, visto que é utilizado apenas um valor que deve representar o grau de similaridade da assinatura em relação ao seu conjunto de referência; e um classificador SVM, que pode receber diversos parâmetros para compor sua heurística. Para a execução dos testes utilizou-se parte da composição do SUSIG [11], mais especificamente o *Visual Subcorpus*, conforme descrito no seção 3.1. Os resultados são descritos no capítulo 4.

Por fim agregou-se esses componentes em uma implementação denominada SignRec, que é um protótipo de um sistema para coleta, armazenamento e classificação de assinaturas, pensado para possibilitar a criação de um banco de dados de amostras de assinaturas com a finalidade de realizar testes em algoritmos de verificação *on-line*.

Alguns pontos ainda merecem atenção como é o caso da base de dados de assinaturas de teste. Hoje é possível inserir usuários no sistema, cadastrar assinaturas de referência e realizar a verificação, porém o cadastramento de assinaturas de teste para a execução de testes automatizados não foi implementado.

Conforme descrito no capítulo 5, a implementação é toda baseada em Java. Exceto pela comunicação com o *hardware* necessário para a coleta de dados. Utiliza-se também de recursos de diversos outros projetos de *software* de código aberto para criação da interface gráfica, acesso aos dados dos dispositivos e outas funcionalidades.

Foram realizadas ainda, comparações do algoritmo DTW com o *FastDTW* aplicados no processo de verificação de assinaturas, mais precisamente na etapa de alinhamento, onde é utilizado. Verificou-se que apesar do *FastDTW* ser consideravelmente mais rápido que o DTW no processamento de grandes volumes de dados, no contexto da comparação de assinaturas sua vantagem não é tão grande. Na média, o tamanho das assinaturas é de 300 pontos, aproximadamente, o que acaba não compensado o *overhead* da criação das diversas resoluções utilizadas no *FastDTW*, além de existir margem para erro em sua execução.

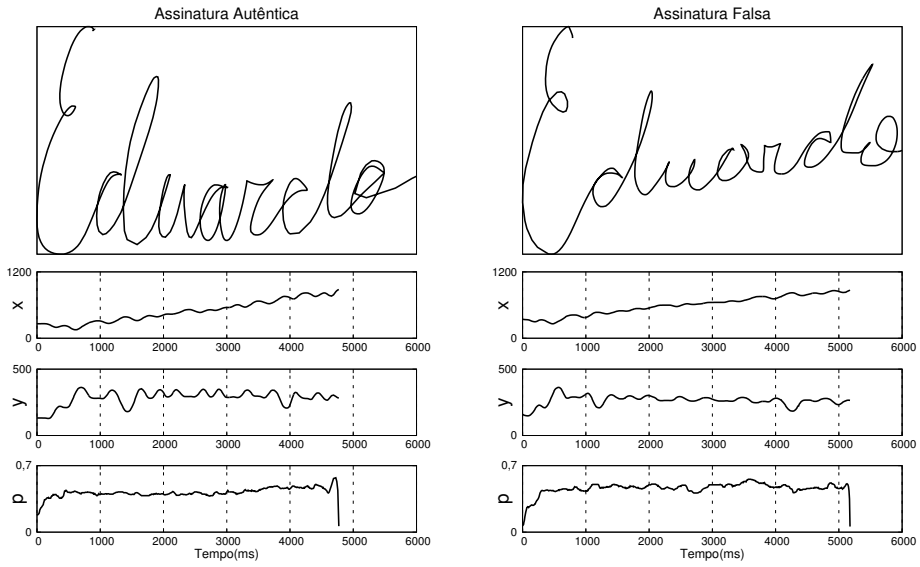

Figura 6.1: Comparação entre assinatura autêntica e falsificação.

Com relação ao processo de extração de parâmetros, a abordagem utilizada é muito generalista, tratando todas as assinaturas do mesmo modo. Do forma que existem diversas abordagens a serem exploradas, principalmente relacionadas a aplicação de processos de análise a estudos específicos de perícia grafoscópica. Pode-se ter uma ideia do tipo de problema existente nesta abordagem com o auxílio da figura 6.1. Apesar de o desenho da falsificação não estar muito próximo da assinatura original, essa amostra se enquadra no grupo de falsificações bem treinadas, tendo grande chance de ser aceita pelo classificador. Pelos gráficos, é possível perceber a semelhança entre as amostras. Tanto o deslocamento em y quanto em y, a pressão e o tempo de execução de ambas as assinaturas são semelhantes. Isso reforça o fato de que o método não leva em consideração a forma de assinatura de modo explícito. Isso ocorre de modo subjetivo.

Os resultados alcançados foram EER=1,276% e EER=1,361% com os classificadores SVM e linear respectivamente.

6.1 Trabalhos Futuros

Apesar de haver diversos trabalhos relacionados ao tema verificação de assinaturas, muitos pontos ainda necessitam de melhor atenção, pois trata-se de uma área de pesquisa com amplas possibilidades de aplicação prática.

A seguir estão listados alguns pontos que podem ser trabalhados como continuação e complementação deste trabalho.

- Verificar o efeito da substituição do cálculo de variância por Análise de Componente Principal para redução de dimensionalidade no treinamento e teste usando classificador Linear.

- Finalizar a implementação e utilizar o sistema de coleta de assinaturas para associar assinaturas a dados a respeito do indivíduo, de modo que permita a realização de estatísticas que permitam revelar padrões relacionados a faixas etárias, gêneros e grau de instrução. Espera-se com isto realizar uma avaliação buscando associar o grau de consistência das assinaturas à determinados perfis de usuários e assim, realizar tratamentos específicos por publico alvo, buscando melhorar a performance do algoritmo.

- Desenvolver uma abordagem que combine vários métodos de classificação buscando reduzir as taxas de falsos positivos sem, contanto, provocar o aumento dos falsos negativos.

Apêndice A

SignRec - Código Fonte

A.1 src.core

A.1.1 Pacote: com.augustoazevedo.signrec.assinador

- Assinador.java
- DataDisplay.java
- LevelsPanel.java
- MainPanel.java
- PenCanvas.java
- SampleRatePanel.java
- StatusPanel.java
- Utils.java

Listing A.1: Classe: Assinador.java

```
1  package com.augustoazevedo.signrec.assinador;
2
3  import java.awt.GraphicsDevice;
4  import java.awt.GraphicsEnvironment;
5  import java.io.IOException;
6  import java.util.logging.Logger;
7
8  import javax.swing.JComponent;
9  import javax.swing.JFrame;
10 import javax.swing.UIManager;
11
12 import com.augustoazevedo.signrec.assinador.vo.Assinatura;
13 import com.augustoazevedo.signrec.frame.AssinaturaControl;
14
15
16 /**
17  *
```

```
18  * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.↩
        com>
19  *
20  * 14/09/2010
21  *
22  */
23 public class Assinador {
24
25     private static final Logger L = Logger.getLogger(Assinador.class.↩
           getName());
26     private final JFrame f = new JFrame("SignRec - Assinador");
27     final PenCanvas penCanvas;
28     final MainPanel mainPanel;
29
30     public AssinaturaControl assinaturaControl;
31
32
33     public Assinador(Assinatura dados, AssinaturaControl assinaturaControl↩
           ) {
34
35         penCanvas = PenCanvas.getInstance(dados);
36
37         this.assinaturaControl = assinaturaControl;
38
39         mainPanel = new MainPanel(penCanvas, this);
40
41         setSupportCustomPKinds(true);
42         penCanvas.penManager.pen.levelEmulator.↩
               setPressureTriggerForLeftCursorButton(0.5f);
43
44         f.getContentPane().add(mainPanel.panel);
45
46         if (!f.isDisplayable()) {
47             f.setUndecorated(true);
48         }
49
50         GraphicsDevice gd = GraphicsEnvironment.↩
               getLocalGraphicsEnvironment().getDefaultScreenDevice();
51
52         //ALTERAR RESOLU O DA JANELA
53         //DisplayMode mode = new DisplayMode(1024, 768, 32, DisplayMode.↩
               REFRESH_RATE_UNKNOWN);
54         //gd.setDisplayMode(mode);
55
56         //JANELA FULL SCREEN
57         gd.setFullScreenWindow(f);
58
59         //JANELA MAXIMIZADA
60         //f.setExtendedState(JFrame.MAXIMIZED_BOTH);
61
62         //JANELA DE TAMANHO X,Y
63         //f.setPreferredSize(new Dimension(800,600));
64
65         f.setDefaultCloseOperation(JFrame. DISPOSE_ON_CLOSE);
66
67         f.setVisible(true);
```

```
68
69      }
70
71
72      public void setSupportCustomPKinds(boolean supportCustomPKinds) {
73      }
74
75
76      public JComponent getMainPanelComponent() {
77          return mainPanel.panel;
78      }
79
80
81      public void showDialog(JComponent parent, String title) {
82      }
83
84
85      public static void main(String... args) throws IOException, ↩
            NumberFormatException {
86          try {
87  //              if (false && System.getProperty("os.name").toLowerCase().↩
        contains("linux")) {
88  //                  UIManager.setLookAndFeel("com.sun.java.swing.plaf.gtk.↩
        GTKLookAndFeel");
89  //              } else {
90                    UIManager.setLookAndFeel(UIManager.↩
                          getSystemLookAndFeelClassName());
91  //              }
92          } catch (Exception ex) {
93              L.warning("The \"system\" look and feel couldn't be set.");
94          }
95
96          new Assinador(null,null);
97      }
98
99      public void dispose() {
100         f.dispose();
101         penCanvas.removerListener();
102         penCanvas.setDados(null);
103
104     }
105 }
```

Listing A.2: Classe: DataDisplay.java

```
1   /* [{
2   Copyright 2008 Nicolas Carranza <nicarran at gmail.com>
3
4   This file is part of jpen.
5
6   jpen is free software: you can redistribute it and/or modify
7   it under the terms of the GNU Lesser General Public License as published ↩
        by
8   the Free Software Foundation, either version 3 of the License,
9   or (at your option) any later version.
```

```
10
11 jpen is distributed in the hope that it will be useful,
12 but WITHOUT ANY WARRANTY; without even the implied warranty of
13 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14 GNU Lesser General Public License for more details.
15
16 You should have received a copy of the GNU Lesser General Public License
17 along with jpen.  If not, see <http://www.gnu.org/licenses/>.
18 }] */
19 package com.augustoazevedo.signrec.assinador;
20
21 import javax.swing.JComponent;
22 import jpen.Pen;
23
24 abstract class DataDisplay<
25     C extends JComponent>{
26     final C component;
27     private boolean isDirty;
28
29     DataDisplay(C component){
30         this.component=component;
31     }
32
33     void setIsDirty(boolean isDirty){
34         this.isDirty=isDirty;
35     }
36
37     boolean getIsDirty(){
38         return isDirty;
39     }
40
41     final void update(Pen pen){
42         if(getIsDirty())
43             updateImp(pen);
44         setIsDirty(false);
45     }
46
47     abstract void updateImp(Pen pen);
48 }
```

Listing A.3: Classe: LevelsPanel.java

```
1  package com.augustoazevedo.signrec.assinador;
2
3  import java.text.DecimalFormat;
4  import java.text.NumberFormat;
5  import java.util.EnumMap;
6  import java.util.Map;
7
8  import javax.swing.JPanel;
9  import javax.swing.JTextField;
10
11 import jpen.PLevel;
12 import jpen.PLevelEvent;
13 import jpen.Pen;
```

```
14 import jpen.event.PenAdapter;
15
16 class LevelsPanel{
17     final Map<PLevel.Type, Display> levelTypeToDisplay=new EnumMap<PLevel.↩
       Type, Display>(PLevel.Type.class);
18     {
19         for(PLevel.Type levelType: PLevel.Type.VALUES){
20             levelTypeToDisplay.put(levelType, new Display(levelType));
21         }
22     }
23
24     static class Display extends DataDisplay<JTextField>{
25
26         static NumberFormat FORMAT=new DecimalFormat("###0,000");
27
28         final PLevel.Type levelType;
29
30         Display(PLevel.Type levelType){
31             super(new JTextField(6));
32             this.levelType=levelType;
33             component.setHorizontalAlignment(JTextField.RIGHT);
34             component.setEditable(false);
35         }
36
37         @Override
38         void updateImp(Pen pen){
39             setValue(pen.getLevelValue(levelType));
40         }
41
42         public void setValue(float value){
43             component.setText(format(value));
44         }
45
46         String format(float value){
47             return FORMAT.format(value);
48         }
49     }
50
51
52     final JPanel panel = new JPanel();
53     {
54         for(PLevel.Type levelType:PLevel.Type.VALUES){
55             if(levelType.toString().equals("X") || levelType.toString().↩
                equals("Y") || levelType.toString().equals("Press o"))
56             panel.add(Utils.labelComponent(
57                         levelType.toString(),levelTypeToDisplay.get(↩
                            levelType).component
58                     ));
59         }
60     }
61
62
63     LevelsPanel(Pen pen){
64         pen.addListener(new PenAdapter(){
65                                         private Pen pen;
66                             @Override
```

```
67                              public void penLevelEvent(PLevelEvent ev){
68                                                  pen=ev.pen;
69                                  for(PLevel level: ev.levels){
70                                      Display display=levelTypeToDisplay.get↩
                                            (level.getType());
71                                                  if(display!=null)
72                                                      display.setIsDirty↩
                                                            (true);
73                                  }
74                              }
75                                                  @Override
76                                                  public void penTock(long ↩
                                                        availableMillis){
77                                                      if(pen==null)
78                                                          return;
79                                                      for(Display display: ↩
                                                            levelTypeToDisplay.↩
                                                            values())
80                                                          display.update(pen);
81                                                      pen=null;
82                                                  }
83                          });
84          for(PLevel.Type levelType: PLevel.Type.VALUES){
85              Display display=levelTypeToDisplay.get(levelType);
86              display.setValue(pen.getLevelValue(levelType));
87          }
88      }
89
90  }
```

Listing A.4: Classe: MainPanel.java

```
1   package com.augustoazevedo.signrec.assinador;
2
3   import java.awt.Dimension;
4   import java.awt.event.KeyEvent;
5   import java.awt.event.KeyListener;
6
7   import javax.swing.Box;
8   import javax.swing.JComponent;
9
10
11  /**
12   *
13   * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.↩
         com>
14   *
15   * 14/09/2010
16   *
17   */
18  class MainPanel {
19
20      final Box panel = Box.createVerticalBox();
21      StatusPanel statePanel;
22
```

```
23      MainPanel(final PenCanvas penCanvas, final Assinador controle) {
24          panel.add(penCanvas.scrollPane);
25          statePanel = new StatusPanel(penCanvas.penManager);
26          statePanel.panel.setMaximumSize(new Dimension(2000, 30));
27          panel.add(statePanel.panel);
28
29          panel.setFocusable(true);
30          panel.addKeyListener(new KeyListener() {
31
32              @Override
33              public void keyTyped(KeyEvent e) {
34              }
35
36              @Override
37              public void keyReleased(KeyEvent e) {
38              }
39
40              @Override
41              public void keyPressed(KeyEvent e) {
42                  int code = e.getKeyCode();
43                  if (code == KeyEvent.VK_CONTROL) {
44                      if (statePanel.panel.isVisible()) {
45                          statePanel.panel.setVisible(false);
46                      } else {
47                          statePanel.panel.setVisible(true);
48                      }
49                  } else if (code == KeyEvent.VK_SHIFT) {
50                      penCanvas.cleanPanel();
51                  }
52                  else if (code == KeyEvent.VK_ENTER) {
53                      //penCanvas.writeFile();
54                      if(controle.assinaturaControl!=null)
55                          controle.assinaturaControl.setOK();
56                      controle.dispose();
57                  }
58                  else if (code == KeyEvent.VK_ESCAPE) {
59                      controle.dispose();
60                  }
61              }
62          });
63      }
64
65      public JComponent getPanel() {
66          return panel;
67      }
68  }
```

Listing A.5: Classe: PenCanvas.java

```
1   package com.augustoazevedo.signrec.assinador;
2
3   import java.awt.BasicStroke;
4   import java.awt.Color;
5   import java.awt.Cursor;
6   import java.awt.Dimension;
```

```
7  import java.awt.Graphics;
8  import java.awt.Graphics2D;
9  import java.awt.GraphicsEnvironment;
10 import java.awt.Rectangle;
11 import java.awt.RenderingHints;
12 import java.awt.geom.Line2D;
13 import java.awt.geom.Point2D;
14 import java.awt.image.BufferedImage;
15 import java.io.IOException;
16 import java.sql.Date;
17 import java.text.SimpleDateFormat;
18
19 import javax.swing.JComponent;
20 import javax.swing.JScrollPane;
21
22 import jpen.PKind;
23 import jpen.PKind.Type;
24 import jpen.PLevel;
25 import jpen.PLevelEvent;
26 import jpen.PenManager;
27 import jpen.event.PenAdapter;
28 import jpen.event.PenListener;
29
30 import com.augustoazevedo.signrec.assinador.vo.Assinatura;
31 import com.augustoazevedo.signrec.assinador.vo.Ponto;
32
33
34 /**
35  *
36  * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.↩
        com>
37  *
38  * 14/09/2010
39  *
40  */
41 class PenCanvas extends JComponent {
42
43     public static final long        serialVersionUID = 1l;
44
45     private static PenCanvas instance = null;
46
47     //private static final Logger   L = Logger.getLogger(PenCanvas.class.↩
        getName());
48     private static final Dimension  SIZE = new Dimension(2000, 1300);
49     private static final Color      BACKGROUND_COLOR = new Color(247, 217,↩
         186); // laranja claro
50     private static final float      STROKE_RAD = 25f;
51     private static final Dimension  PREF_SCROLLPANE_SIZE = new Dimension↩
        (800, 600);
52
53     final PenManager penManager;
54     final JScrollPane scrollPane;
55     private final BufferedImage image = GraphicsEnvironment.↩
        getLocalGraphicsEnvironment().getDefaultScreenDevice().↩
        getDefaultConfiguration().createCompatibleImage(SIZE.width, SIZE.↩
        height);
```

```
56      private final Graphics2D g = (Graphics2D) image.getGraphics();
57      private final Point2D.Float loc = new Point2D.Float();
58      private final Point2D.Float prevLoc = new Point2D.Float();
59      private float prevPressure = 0;
60      private BasicStroke stroke;
61      private final Rectangle dirtyArea = new Rectangle();
62      private boolean isDirty;
63      private final Rectangle rectangle = new Rectangle();
64      public Assinatura dados = new Assinatura();
65      private PenListener listener;
66
67
68      private PenCanvas() {
69
70          this.penManager = new PenManager(this);
71          penManager.pen.setFirePenTockOnSwing(true);
72          penManager.pen.setFrequencyLater(40);
73
74          //HABILITA SCROLLBARS
75          // Utils.freezeSize(this, SIZE);
76
77          setDoubleBuffered(false);
78          setOpaque(true);
79          scrollPane = new JScrollPane(this);
80          scrollPane.setPreferredSize(PREF_SCROLLPANE_SIZE);
81          scrollPane.setCursor(Cursor.getPredefinedCursor(Cursor.↩
                CROSSHAIR_CURSOR));
82
83          image.setAccelerationPriority(1);
84
85          g.setColor(BACKGROUND_COLOR);
86          rectangle.setFrame(0, 0, SIZE.width, SIZE.height);
87          g.fill(rectangle);
88          setupRenderingHints(g);
89
90          paintInstructions();
91
92          listener = (new PenAdapter() {
93
94              @Override
95              public void penLevelEvent(PLevelEvent ev) {
96
97                  for (PLevel.Type levelType : PLevel.Type.MOVEMENT_TYPES) {
98                      float value = penManager.pen.getLevelValue(levelType);
99                      switch (levelType) {
100                         case X:
101                             loc.x = value;
102                             break;
103                         case Y:
104                             loc.y = value;
105                             break;
106                     }
107                 }
108
109                 PKind.Type kindType = penManager.pen.getKind().getType();
110
```

```
111                 // TODO: Retirar CURSOR para que sejam aceitos somente os ↩
                        dados da STYLUS.
112
113                 if (kindType == Type.STYLUS || kindType == Type.CURSOR) {
114                     Ponto ponto = new Ponto(loc.x, loc.y, penManager.pen.↩
                            getLevelValue(PLevel.Type.PRESSURE), ev.↩
                            getDeviceTime());
115                     dados.add(ponto);
116                 }
117
118                 paintStroke();
119             }
120
121             @Override
122             public void penTock(long availableTime) {
123                 if (availableTime < 0) {
124                     //L.warning("no time to repaint... but this is a test,↩
                            so I continue.");
125                 }
126                 if (isDirty) {
127                     repaint(dirtyArea.x, dirtyArea.y, dirtyArea.width,
128                             dirtyArea.height);
129                 }
130                 isDirty = false;
131
132             }
133         });
134         penManager.pen.addListener(listener);
135
136     }
137
138
139     /**
140      * Singleton implementation.
141      * @return
142      */
143     public static PenCanvas getInstance(Assinatura dados){
144         if(instance == null){
145             instance = new PenCanvas();
146             if(dados != null)
147                 instance.setDados(dados);
148         }
149         else {
150             if(dados != null)
151                 instance.setDados(dados);
152             else
153                 instance.setDados(new Assinatura());
154             instance.clean();
155         }
156
157         return instance;
158     }
159
160
161     public void clean(){
162         cleanPanel();
```

```
163         penManager.pen.removeListener(listener);
164         penManager.pen.addListener(listener);
165     }
166 
167     public void setDados(Assinatura dados){
168         if (dados != null) {
169             this.dados = dados;
170         }
171     }
172 
173     private void paintInstructions() {
174         g.setColor(Color.BLACK);
175         g.drawString("ESC: Close", 10, 15);
176         g.drawString("SHIFT: Erase all", 10, 30);
177         g.drawString("CRTL: Hide/Show status bar", 10, 45);
178         g.drawString("ENTER: Save", 10, 60);
179     }
180 
181     private static void setupRenderingHints(Graphics2D g) {
182         g.setRenderingHint(RenderingHints.KEY_RENDERING,
183                 RenderingHints.VALUE_RENDER_QUALITY);
184         g.setRenderingHint(RenderingHints.KEY_STROKE_CONTROL,
185                 RenderingHints.VALUE_STROKE_PURE);
186         g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
187                 RenderingHints.VALUE_ANTIALIAS_ON);
188     }
189 
190     private synchronized void paintStroke() {
191         float pressure = penManager.pen.getLevelValue(PLevel.Type.PRESSURE↩
                );
192         if (pressure == 0) {
193             prevLoc.x = loc.x;
194             prevLoc.y = loc.y;
195             //L.fine("no pressure");
196             return;
197         }
198 
199         //CONTORNA BUG NA CAPTURA DE PRESSAO
200         if(pressure == 1 && prevPressure < 0.2)
201             pressure = prevPressure;
202         prevPressure = pressure;
203 
204         pressure *= pressure; // parabolic sensitivity
205         float r = pressure * STROKE_RAD;
206         PKind.Type kindType = penManager.pen.getKind().getType();
207         switch (kindType) {
208             case CUSTOM:
209                 g.setColor(Color.PINK);
210                 break;
211             case STYLUS:
212                 g.setColor(Color.BLACK);
213                 break;
214             case ERASER:
215                 g.setColor(BACKGROUND_COLOR);
216                 r = r * 2;
217                 break;
```

```
218             case CURSOR:
219                 g.setColor(Color.BLUE);
220         }
221         stroke = new BasicStroke(r, BasicStroke.CAP_ROUND,
222                 BasicStroke.JOIN_MITER);
223 
224         g.setStroke(stroke);
225         g.draw(new Line2D.Float(prevLoc, loc));
226 
227         prevLoc.x = loc.x;
228         prevLoc.y = loc.y;
229 
230         //L.fine("stoke painted");
231         addCursorAreaToDirtyArea();
232     }
233 
234     private void addCursorAreaToDirtyArea() {
235         evalCursorArea(rectangle);
236         if (isDirty) {
237             dirtyArea.add(rectangle);
238         } else {
239             dirtyArea.setRect(rectangle);
240         }
241         isDirty = true;
242     }
243 
244     private void evalCursorArea(Rectangle r) {
245         float max = 2 * STROKE_RAD + 2;
246         r.x = (int) (loc.x - max);
247         r.y = (int) (loc.y - max);
248         r.width = r.height = (int) (2 * max);
249     }
250 
251     @Override
252     protected synchronized void paintComponent(Graphics g) {
253         Graphics2D g2 = (Graphics2D) g;
254         g2.drawImage(image, null, null);
255     }
256 
257     protected synchronized void cleanPanel() {
258         dados.clear();
259         g.setBackground(BACKGROUND_COLOR);
260         g.clearRect(0, 0, SIZE.width, SIZE.height);
261         paintInstructions();
262         repaint();
263     }
264 
265     protected void writeFile() {
266         SimpleDateFormat formatador = new SimpleDateFormat("↩
                yyyyMMdd_HHmmss");
267         Date data = new Date(System.currentTimeMillis());
268 
269         penManager.pen.removeListener(listener);
270         try {
271             dados.saveFile(new String("data/teste_" + formatador.format(↩
                    data) + ".sig"), new String[]{"0"}," ");
```

```
272             //JOptionPane.showMessageDialog(this, "Arquivo salvo com ↩
                    sucesso. Diret rio 'data'.");
273         } catch (IOException e) {
274             //JOptionPane.showMessageDialog(this, "Erro: " + e.getMessage↩
                    ());
275             //e.printStackTrace();
276         }
277     }
278 
279     public void removerListener(){
280         penManager.pen.removeListener(listener);
281     }
282 }
```

Listing A.6: Classe: SampleRatePanel.java

```
1  /* [{
2  Copyright 2007, 2008 Nicolas Carranza <nicarran at gmail.com>
3  
4  This file is part of jpen.
5  
6  jpen is free software: you can redistribute it and/or modify
7  it under the terms of the GNU Lesser General Public License as published ↩
       by
8  the Free Software Foundation, either version 3 of the License,
9  or (at your option) any later version.
10 
11 jpen is distributed in the hope that it will be useful,
12 but WITHOUT ANY WARRANTY; without even the implied warranty of
13 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14 GNU Lesser General Public License for more details.
15 
16 You should have received a copy of the GNU Lesser General Public License
17 along with jpen.  If not, see <http://www.gnu.org/licenses/>.
18 }] */
19 package com.augustoazevedo.signrec.assinador;
20 
21 import java.util.LinkedList;
22 
23 import javax.swing.JTextField;
24 
25 import jpen.PLevelEvent;
26 import jpen.Pen;
27 import jpen.event.PenAdapter;
28 
29 class SampleRatePanel{
30     public static final int SAMPLE_COUNT=10;
31     private final LinkedList<Integer> periods=new LinkedList<Integer>();
32     private int average;
33 
34     private final JTextField textField=new JTextField(7);
35     {
36         textField.setEditable(false);
37         textField.setHorizontalAlignment(JTextField.RIGHT);
38     }
```

```
39
40      final JTextField panel=textField;
41
42      SampleRatePanel(final Pen pen){
43          pen.addListener(new PenAdapter(){
44                                  private long lastDeviceTime=-1;
45                                  @Override
46                                  public void penLevelEvent(PLevelEvent ev){
47                                      long evDeviceTime=ev.getDeviceTime();
48                                      if(lastDeviceTime==-1){
49                                          lastDeviceTime=evDeviceTime;
50                                          return;
51                                      }
52                                      long period=evDeviceTime-lastDeviceTime;
53                                      addPeriod((int)period);
54                                      lastDeviceTime=evDeviceTime;
55                                  }
56                                  @Override
57                                  public void penTock(long availableMillis){
58                                      updateTextField();
59                                  }
60                              });
61      }
62
63
64      void addPeriod(int period){
65          if(periods.size()==SAMPLE_COUNT)
66              periods.removeFirst();
67          periods.add(period);
68      }
69
70      void updateAverage(){
71          average=0;
72          for(int period: periods)
73              average+=period;
74          if(periods.size()>0)
75              average/=periods.size();
76      }
77
78      void updateTextField(){
79          updateAverage();
80          textField.setText(average+" ms");
81      }
82  }
```

Listing A.7: Classe: StatusPanel.java

```
1  /*
2   *
3   * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.↩
        com>
4   *
5   * 14/09/2010
6   *
7   */
```

```
8  package com.augustoazevedo.signrec.assinador;
9
10 import java.awt.FlowLayout;
11
12 import javax.swing.JPanel;
13
14
15 import jpen.Pen;
16 import jpen.PenManager;
17
18 /**
19  *
20  * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.↩
       com>
21  *
22  * 14/09/2010
23  *
24  */
25 class StatusPanel {
26     final JPanel panel = new JPanel();
27
28     StatusPanel(PenManager penManager) {
29         panel.setLayout(new FlowLayout(FlowLayout.CENTER, 0, 0));
30
31         Pen pen = penManager.pen;
32
33         LevelsPanel levelsPanel = new LevelsPanel(pen);
34         SampleRatePanel sampleRatePanel = new SampleRatePanel(pen);
35
36         panel.add(levelsPanel.panel);
37         levelsPanel.panel.setLayout(new FlowLayout(FlowLayout.CENTER, 0, ↩
               0));
38
39         panel.add(Utils.labelComponent("Per odo:",
40                 sampleRatePanel.panel));
41         panel.add(Utils.createHorizontalStrut());
42
43     }
44 }
```

Listing A.8: Classe: Utils.java

```
1  /* [{
2  Copyright 2008 Nicolas Carranza <nicarran at gmail.com>
3
4  This file is part of jpen.
5
6  jpen is free software: you can redistribute it and/or modify
7  it under the terms of the GNU Lesser General Public License as published ↩
       by
8  the Free Software Foundation, either version 3 of the License,
9  or (at your option) any later version.
10
11 jpen is distributed in the hope that it will be useful,
12 but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
13 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14 GNU Lesser General Public License for more details.
15
16 You should have received a copy of the GNU Lesser General Public License
17 along with jpen.  If not, see <http://www.gnu.org/licenses/>.
18 }] */
19 package com.augustoazevedo.signrec.assinador;
20
21 import java.awt.Component;
22 import java.awt.Dimension;
23 import javax.swing.border.Border;
24 import javax.swing.BorderFactory;
25 import javax.swing.Box;
26 import javax.swing.JComponent;
27 import javax.swing.JLabel;
28
29 final class Utils{
30     private static final Border LABELED_COMPONENT_BORDER=BorderFactory.↩
        createEmptyBorder(3,3,2,2);
31
32     static JComponent labelComponent(String label, Component c){
33         Box box=Box.createHorizontalBox();
34         if(!label.trim().endsWith(":"))
35             label=label.trim()+": ";
36         box.add(new JLabel(label));
37         box.add(Box.createHorizontalGlue());
38         freezeSizeToPreferred(c);
39         box.add(c);
40         box.setBorder(LABELED_COMPONENT_BORDER);
41         return alignTopLeft(box);
42     }
43
44     static void freezeSizeToPreferred(Component c){
45         freezeSize(c, c.getPreferredSize());
46     }
47
48     static void freezeSize(Component c, Dimension s){
49         c.validate();
50         c.setMinimumSize(s);
51         c.setMaximumSize(s);
52         c.setPreferredSize(s);
53     }
54
55     static JComponent alignTopLeft(JComponent c){
56         c.setAlignmentX(Component.LEFT_ALIGNMENT);
57         c.setAlignmentY(Component.TOP_ALIGNMENT);
58         return c;
59     }
60
61     static Component createHorizontalStrut(){
62         return new Box.Filler(new Dimension(3,0), new Dimension(3,0),
63                 new Dimension(3, 0));
64     }
65
66     static Component createVerticalStrut(){
67         return new Box.Filler(new Dimension(0,3), new Dimension(0,3),
```

```
68                  new Dimension(0, 3));
69      }
70  }
```

A.1.2 Pacote: com.augustoazevedo.signrec.assinador.vo

- Assinatura.java
- Ponto.java

Listing A.9: Classe: Assinatura.java

```
1   package com.augustoazevedo.signrec.assinador.vo;
2
3   import java.io.BufferedReader;
4   import java.io.BufferedWriter;
5   import java.io.FileNotFoundException;
6   import java.io.FileReader;
7   import java.io.FileWriter;
8   import java.io.IOException;
9   import java.io.Serializable;
10  import java.sql.Date;
11  import java.text.SimpleDateFormat;
12  import java.util.ArrayList;
13  import java.util.Iterator;
14  import java.util.StringTokenizer;
15
16  import com.augustoazevedo.signrec.dtw.TimeSeries;
17
18
19  public class Assinatura extends ArrayList<Ponto> implements Serializable {
20
21      private static final long serialVersionUID = 1684968093759061755L;
22
23      private float   max_x = 0,
24                      max_y = 0;
25
26
27      public Assinatura(){
28          super();
29      }
30
31      public Assinatura(String inputFile, boolean isLabeled, boolean ↩
            hasTotalPoints, String[] layout, String delimiter){
32          super();
33          try
34          {
35              // Record the Label names (fropm the top row.of the input file)↩
                .
36              BufferedReader br = new BufferedReader (new FileReader (↩
                    inputFile));  // open the input file
37
38              if (!isLabeled)
```

```
39                  br.readLine();
40
41              if (!hasTotalPoints)
42                  br.readLine();
43
44              String line = br.readLine();  // the top row that contains ↩
                    attribiute names.
45              StringTokenizer st = new StringTokenizer (line, String.valueOf(↩
                    delimiter));
46
47              // Read in all of the values in the data file.
48              while ((line = br.readLine()) != null)   // read lines until ↩
                    end of file
49              {
50                 if (line.length() > 0)  // ignore empty lines
51                 {
52                    st = new StringTokenizer(line,delimiter);
53
54                    String x = null,y = null,pressure = null,timestamp = null↩
                          ;
55
56                    int currentCol = 0;
57                    while (st.hasMoreTokens())
58                    {
59                       final String currentToken = st.nextToken();
60                       if (Ponto.X.equals(layout[currentCol])){
61                          x = currentToken;
62                       }
63                       else if (Ponto.Y.equals(layout[currentCol])){
64                           y = currentToken;
65                       }
66                       else if (Ponto.PRESSURE.equals(layout[currentCol])){
67                           pressure = currentToken;
68                       }
69                       else if (Ponto.TIMESTAMP.equals(layout[currentCol])){
70                           timestamp = currentToken;
71                       }
72
73                       currentCol++;
74                    }  // end while loop
75
76                    try {
77                        this.add(new Ponto(
78                                (x!=null?Float.valueOf(x).floatValue():0f),
79                                (y!=null?Float.valueOf(y).floatValue():0f),
80                                (pressure!=null?Float.valueOf(pressure).↩
                                       floatValue():0f),
81                                (timestamp!=null?Long.valueOf(timestamp).↩
                                       longValue():0l))
82                        );
83                    }catch(Exception e){
84
85                    }
86
87                 }  // end if
88              }  // end while loop
```

```
89          }
90          catch (FileNotFoundException e){
91             throw new InternalError("ERROR:  The file '" + inputFile + "' ↩
                was not found.");
92          }
93          catch (IOException e){
94             throw new InternalError("ERROR:  Problem reading the file '" + ↩
                inputFile + "'.");
95          }  // end try block
96
97          this.processData();
98      }
99
100     public void saveFile(String outputFile, String[] layout, String ↩
            delimiter) throws IOException{
101         SimpleDateFormat formatador = new SimpleDateFormat("↩
                yyyyMMdd_HHmmss");
102         Date data = new Date(System.currentTimeMillis());
103
104         this.processData();
105
106         try {
107             FileWriter file;
108             if(outputFile == null)
109                 file = new FileWriter("data/teste_" + formatador.format(↩
                        data) + ".txt");
110             else
111                 file = new FileWriter(outputFile);
112
113             BufferedWriter out = new BufferedWriter(file);
114
115             Iterator<Ponto> i = (Iterator<Ponto>) this.iterator();
116             if("0".equals(layout[0])){
117                 while (i.hasNext()) {
118                     Ponto a = i.next();
119                     out.write(a.x + delimiter + a.y + delimiter + a.↩
                            pressure + delimiter + a.timestamp + delimiter + "↩
                            \n");
120                 }
121             }
122             else if("1".equals(layout[0])){
123                 while (i.hasNext()) {
124                     Ponto a = i.next();
125                     out.write(a.timestamp + delimiter + a.deslocamento + "↩
                            \n");
126                 }
127             }
128             else if("2".equals(layout[0])){
129                 while (i.hasNext()) {
130                     Ponto a = i.next();
131                     out.write(a.timestamp + delimiter + a.deslocamento_x +↩
                             delimiter + a.deslocamento_y + "\n");
132                 }
133             }
134             else if("3".equals(layout[0])){
135                 while (i.hasNext()) {
```

```
136                     Ponto a = i.next();
137                     out.write(a.timestamp + delimiter + a.deslocamento_x +↩
                            delimiter + a.deslocamento_y + delimiter + a.↩
                            pressure + "\n");
138                 }
139             }
140             else if("4".equals(layout[0])){
141                 while (i.hasNext()) {
142                     Ponto a = i.next();
143                     out.write(a.timestamp + delimiter + a.deslocamento + ↩
                            delimiter + a.pressure + "\n");
144                 }
145             }
146 
147             out.close();
148         } catch (IOException e) {
149             // e.printStackTrace();
150             throw e;
151         }
152     }
153 
154 
155 
156     public void processData() {
157         if (this.size() > 0) {
158             retiraLixoInicioFim();
159             removePontosTempoIgual();
160             calculaDiferencaEntrePontos();
161             timestampRelativo();
162         }
163     }
164 
165     /*
166      * Remove movimentos aereos do inicio e final da assinatura.
167      */
168     private void retiraLixoInicioFim() {
169         if (this.size() > 0) {
170             /* Retira pontos aereos do inicio e do final das assinaturas.
171              */
172             while (get(0).pressure == 0) {
173                 remove(0);
174             }
175             while (get(size() - 1).pressure == 0) {
176                 remove(size() - 1);
177             }
178 
179             /* Corrige o tempo em rela   o     primeira amostra.
180              */
181             Long size = get(0).timestamp;
182             Iterator<Ponto> i = iterator();
183             while (i.hasNext()) {
184                 Ponto a = i.next();
185                 a.timestamp -= size;
186             }
187         }
188     }
```

```
189
190     /*
191      * Calcula o deslocamento do ponto em relacao ao anterior.
192      */
193     private void calculaDiferencaEntrePontos() {
194         Iterator<Ponto> i = iterator();
195         Ponto a = i.next();
196         a.deslocamento = 0f;
197         a.deslocamento_x = 0f;
198         a.deslocamento_y = 0f;
199         this.max_x = a.x;
200         this.max_y = a.y;
201
202         while (i.hasNext()) {
203             Ponto b = i.next();
204             // C lculo de hipotenusa.
205             double x2 = Math.pow((a.x - b.x), 2);
206             double y2 = Math.pow((a.y - b.y), 2);
207             b.deslocamento = (float) Math.sqrt(x2 + y2);
208             b.deslocamento_x = (a.x - b.x);
209             b.deslocamento_y = (a.y - b.y);
210
211             if(this.max_x < b.x) this.max_x = b.x;
212             if(this.max_y < b.y) this.max_y = b.y;
213
214             a = b;
215         }
216     }
217
218     /*
219      * Remove amostragens registradas com mesmo valor para tempo.
220      * Para capturas com mouse....
221      */
222     private void removePontosTempoIgual() {
223         ArrayList<Ponto> removerLista = new ArrayList<Ponto>();
224         Iterator<Ponto> i = iterator();
225
226         Ponto a = i.next();
227         while (i.hasNext()) {
228             Ponto b = i.next();
229
230             //CONTORNA BUG NA CAPTURA DE PRESSAO
231             if(b.pressure == 1 && a.pressure == 0)
232                 b.pressure = 0;
233
234             if (b.timestamp == a.timestamp) {
235                 removerLista.add(b);
236             } else {
237                 a = b;
238             }
239         }
240
241         i = removerLista.iterator();
242         while (i.hasNext()) {
243             remove(i.next());
244         }
```

```
245     }
246
247
248     /**
249      *  Geralmente os dispositivos para digitaliza   o retornam o ↩
            timestamp de um ponto
250      *  que corresponde    hora em que esse ponto foi capturado.
251      *  Neste m todo os timestamps de todos os pontos s o relativizados ↩
            ao primeiro,
252      *  passando o valor deste a ser 0.
253      */
254     private void timestampRelativo(){
255         if(size() > 0 || get(0).timestamp != 0){
256             long deslocamento = get(0).timestamp;
257             Iterator<Ponto> i = iterator();
258             while(i.hasNext()){
259                 Ponto p = i.next();
260                 p.timestamp = p.timestamp-deslocamento;
261             }
262         }
263     }
264
265
266     public void redimensionar(){
267         redimensionar(.5f);
268     }
269
270     public void redimensionar(float proporcao) {
271         Iterator<Ponto> i = iterator();
272         while (i.hasNext()) {
273             Ponto p = i.next();
274             p.x = p.x * proporcao;
275             p.y = p.y * proporcao;
276         }
277     }
278
279
280     public float getMax_x(){
281         return max_x;
282     }
283     public float getMax_y(){
284         return max_y;
285     }
286
287     /**
288      *
289      * @param layout
290      * @return
291      */
292     public TimeSeries getTimeSeries(ArrayList<String> layout){
293         return new TimeSeries(this, layout);
294     }
295 }
```

Listing A.10: Classe: Ponto.java

```
1  package com.augustoazevedo.signrec.assinador.vo;
2
3  import java.io.Serializable;
4
5  public class Ponto implements Serializable {
6
7      /**
8       *
9       */
10     private static final long serialVersionUID = 1L;
11
12     public float x = 0;
13     public float y = 0;
14     public float pressure = 0;
15     public long timestamp = 0;
16     public float deslocamento = 0;
17     public float deslocamento_x = 0;
18     public float deslocamento_y = 0;
19
20     public static final String X = "X";
21     public static final String Y = "Y";
22     public static final String PRESSURE = "PRESSURE";
23     public static final String TIMESTAMP = "TIMESTAMP";
24     public static final String DESLOCAMENTO = "DESLOCAMENTO";
25     public static final String DESLOCAMENTO_X = "DESLOCAMENTO_X";
26     public static final String DESLOCAMENTO_Y = "DESLOCAMENTO_Y";
27
28     public Ponto(float x, float y, float pressure, long time) {
29         this.x = x;
30         this.y = y;
31         this.pressure = pressure;
32         this.timestamp = time;
33     }
34
35     public void setDeslocamento(float deslocamento){
36         this.deslocamento = deslocamento;
37     }
38 }
```

A.1.3 Pacote: com.augustoazevedo.signrec.cadastro

- ConjuntoReferencia.java
- ProcessadorCadastramento.java
- VetorCadastro.java
- VetorCaracteristicas.java

Listing A.11: Classe: ConjuntoReferencia.java

```
1  /**
```

```
 * @author Augusto Cesar Gonçalves de Azevedo <augusto.azevedo at gmail.com>
 *
 * 12/09/2010
 */
package com.augustoazevedo.signrec.cadastro;

import java.util.ArrayList;
import java.util.Iterator;

import com.augustoazevedo.signrec.assinador.vo.Assinatura;
import com.augustoazevedo.signrec.dtw.TimeSeries;


/**
 * @author Augusto Cesar Gonçalves de Azevedo <augusto.azevedo at gmail.com>
 *
 * 12/09/2010
 *
 */
public class ConjuntoReferencia extends java.util.ArrayList<Assinatura> {

    private static final long serialVersionUID = 1L;

    /**
     *
     * @param layout
     * @return
     */
    public ArrayList<TimeSeries> getTimeSeries(ArrayList<String> layout){
        ArrayList<TimeSeries> array_TS = new ArrayList<TimeSeries>();

        Iterator<Assinatura> i = iterator();
        while(i.hasNext()){
            Assinatura a = i.next();
            array_TS.add(a.getTimeSeries(layout));
        }

        return array_TS;
    }
}
```

Listing A.12: Classe: ProcessadorCadastramento.java

```
package com.augustoazevedo.signrec.cadastro;
import java.util.ArrayList;
import java.util.Iterator;

import com.augustoazevedo.signrec.assinador.vo.Assinatura;
import com.augustoazevedo.signrec.assinador.vo.Ponto;
import com.augustoazevedo.signrec.dtw.TimeSeries;
import com.fastdtw.dtw.TimeWarpInfo;
import com.fastdtw.dtw.WarpPath;
import com.fastdtw.matrix.ColMajorCell;
```

```
11
12 /**
13  *
14  * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.↩
        com>
15  *
16  * 12/09/2010
17  *
18  */
19 public class ProcessadorCadastramento {
20
21     private static final int NUMERO_PARTES_ENERGIA = 32;
22
23     public static final ArrayList<String> LAYOUT_TIME_SERIES = new ↩
           ArrayList<String>();
24     public static final String[] LAYOUT_STRING = {Ponto.X, Ponto.Y, Ponto.↩
           TIMESTAMP, Ponto.PRESSURE, null};
25
26     static {
27         LAYOUT_TIME_SERIES.add(Ponto.TIMESTAMP);
28         LAYOUT_TIME_SERIES.add(Ponto.DESLOCAMENTO_X);
29         LAYOUT_TIME_SERIES.add(Ponto.DESLOCAMENTO_Y);
30         LAYOUT_TIME_SERIES.add(Ponto.PRESSURE);
31     }
32
33     /**
34      * Realiza a extra   o de par metros do conjunto de refer ncia.
35      *
36      * @param conjReferencia Conjunto de assinaturas de refer ncia do ↩
            usu rio.
37      * @param usarFastDTW Se <b>true</b> utiliza FastDTW. Caso contr rio ↩
            utiliza DTW.
38      * @param raio Raio de avalia   o para o caso do uso do FastDTW. Caso↩
            n o seja informado o valor padr o    10.
39      * @return Retorna o vetor de caracter sticas do Conjunto de ↩
            Refer ncia das assinaturas do usu rio.
40      */
41     public VetorCadastro processaCadastro(ConjuntoReferencia ↩
           conjReferencia, boolean usarFastDTW, int raio){
42         int qtdAmostras = conjReferencia.size();
43         double[][][] distancias = new double[qtdAmostras][qtdAmostras];
44         double[] energiaTotal = new double[NUMERO_PARTES_ENERGIA];
45
46         ArrayList<TimeSeries> conjReferencia_TS = conjReferencia.↩
               getTimeSeries(LAYOUT_TIME_SERIES);
47
48         Iterator<Assinatura> iterator = conjReferencia.iterator();
49         while(iterator.hasNext()){
50             Assinatura assinatura = iterator.next();
51             int i = conjReferencia.indexOf(assinatura);
52
53             double[] energia = calcularEnergiaInterna(assinatura,↩
                   NUMERO_PARTES_ENERGIA);
54
55             for(int j = 0; j < energia.length; j++){
56                 energiaTotal[j] += energia[j];
```

```
57              }
58
59              for(int n = i+1; n< (conjReferencia.size()); n++){
60                  final TimeSeries tsI = conjReferencia_TS.get(i);
61                  final TimeSeries tsJ = conjReferencia_TS.get(n);
62                  raio = (raio<0)?10:raio;
63                  final TimeWarpInfo info = (usarFastDTW) ? (com.fastdtw.dtw↩
                        .FastDTW.getWarpInfoBetween(tsI, tsJ, raio)) : (com.↩
                       fastdtw.dtw.DTW.getWarpInfoBetween(tsI, tsJ));
64
65                  distancias[n][i] = info.getDistance() * penalidade(info);
66                  distancias[i][n] = distancias[n][i];
67              }
68          }
69
70          return calculaMedias(distancias, energiaTotal, conjReferencia);
71      }
72
73      /**
74       *
75       * @param assinatura
76       * @param cadastro
77       * @return
78       */
79      public static VetorCaracteristicas comparaAssinatura(Assinatura ↩
            assinatura, VetorCadastro cadastro){
80
81          int raio = cadastro.getRaio();
82          boolean usarFastDTW = cadastro.isUsarFastDTW();
83
84          ConjuntoReferencia referencia = cadastro.getConjReferencia();
85          ArrayList<TimeSeries> referencia_TS = referencia.getTimeSeries(↩
                LAYOUT_TIME_SERIES);
86          int qtdAmostas = referencia.size();
87
88          double[] energia = ProcessadorCadastramento.calcularEnergiaInterna↩
                (assinatura,NUMERO_PARTES_ENERGIA);
89
90          TimeSeries assinatura_TS = assinatura.getTimeSeries(↩
                LAYOUT_TIME_SERIES);
91          double[] distancias = new double[referencia.size()];
92
93          for(int i = 0; i < qtdAmostas; i++){
94              final TimeSeries tsI = assinatura_TS;
95              final TimeSeries tsJ = referencia_TS.get(i);
96              raio = (raio<0)?10:raio;
97              final TimeWarpInfo info = (usarFastDTW) ? (com.fastdtw.dtw.↩
                    FastDTW.getWarpInfoBetween(tsI, tsJ, raio)) : (com.fastdtw↩
                    .dtw.DTW.getWarpInfoBetween(tsI, tsJ));
98
99              distancias[i] = info.getDistance() * penalidade(info);
100         }
101
102         double  min = distancias[0],
103                 max = distancias[0],
104                 template = distancias[cadastro.getTemplateIndex()];
```

```
105
106         for(int y=1; y<distancias.length; y++){
107             min = distancias[y]<min?distancias[y]:min;
108             max = distancias[y]>max?distancias[y]:max;
109         }
110         return new VetorCaracteristicas(min,max,template,energia);
111     }
112
113
114     /**
115      *
116      * @param valores
117      * @param energia
118      * @param conjReferencia
119      * @return
120      */
121     private static VetorCadastro calculaMedias(double[][] valores, double↩
        [] energia, ConjuntoReferencia conjReferencia){
122         int qtdAmostras = valores.length;
123
124         double  minMedio = 0,
125                 maxMedio = 0,
126                 templateMedio = 0;
127
128         int templateIndex = 0;
129
130         double[] medias = new double[qtdAmostras];
131
132         for(int i = 0; i<qtdAmostras; i++){
133             int inicial = i==0?1:0;
134
135             double  min = valores[i][inicial],
136                     max = valores[i][inicial],
137                     media = valores[i][inicial];
138
139             for(int n = 1; n<qtdAmostras; n++){
140                 if(n!=i){
141                     min = valores[i][n]<min?valores[i][n]:min;
142                     max = valores[i][n]>max?valores[i][n]:max;
143                     media += valores[i][n];
144                 }
145             }
146
147             minMedio += min;
148             maxMedio += max;
149             medias[i] = (media/((double)qtdAmostras-1));
150         }
151
152         minMedio = (minMedio/((double)qtdAmostras));
153         maxMedio = (maxMedio/((double)qtdAmostras));
154
155         templateMedio = medias[0];
156         templateIndex = 1;
157         for(int y=0; y<medias.length; y++){
158             if(templateMedio > medias[y]){
159                 templateMedio = medias[y];
```

```
160                 templateIndex = y;
161             }
162         }
163 
164         //Tira a m dia da energia interna das amostras
165         for(int j = 0; j < energia.length; j++){
166             energia[j] = ((double) energia[j]/qtdAmostras);
167         }
168 
169         return new VetorCadastro(minMedio, maxMedio, templateMedio, ↩
                templateIndex, energia, conjReferencia);
170     }
171 
172 
173     /**
174      *
175      * @param info
176      * @return
177      */
178     public static double calcularPercentualDiagonais(TimeWarpInfo info){
179         WarpPath wp = info.getPath();
180         int size = wp.size();
181 
182         double diagonais = 0;
183         int prevA = 0,
184             prevB = 0;
185 
186         for(int i = 0; i<size; i++){
187             ColMajorCell passo = wp.get(i);
188             int a = passo.getCol(),
189                 b = passo.getRow();
190 
191             if(a == prevA+1 && b == prevB+1)
192                 diagonais++;
193 
194             prevA = a;
195             prevB = b;
196         }
197 
198         int maxI = wp.maxI(),
199             maxJ = wp.maxJ();
200 
201         if(maxI < maxJ)
202             return (double)diagonais/maxI;
203         else
204             return (double)diagonais/maxJ;
205     }
206 
207     /**
208      * Calcula um percentual de penalidade de acordo com o percentual de ↩
            diagonais
209      * do {@link WarpPath} resultante da execu  o do DTW ou FastDTW ↩
            entre duas assinaturas.
210      *
211      * @param info TimeWarpInfo retornado pela execu  o do DTW ou ↩
            FastDTW entre duas assinaturas.
```

```
212         * @return Valor da penalidade proporcional ao percentual de diagonais↩
                no WarpPath.
213         */
214        public static double penalidade(TimeWarpInfo info){
215            return Math.pow((1+(1-calcularPercentualDiagonais(info))),4);
216        }
217
218
219        /**
220         * Calcula a energia interna da assinatura ou partes dela de acordo ↩
                com <b>nroPartes</b>.
221         *
222         * @param assinatura Assinatura para a qual se deseja calcular a ↩
                energia interna.
223         * @param nroPartes N mero de partes em que a assinatura ser  ↩
                dividida para realiza  o do c lculo.
224         * @return Vetor de tamanho nroPartes contendo o resultado do c lculo↩
                para cada uma das partes.
225         */
226        public static double[] calcularEnergiaInterna(Assinatura assinatura, ↩
               int nroPartes){
227            int pontosPorParte = assinatura.size()/nroPartes;
228            double[] energia = new double[nroPartes];
229            Iterator<Ponto> iterator = assinatura.iterator();
230
231            for(int i = 0; i <nroPartes; i++){
232                int n = 0;
233                while(n < pontosPorParte && iterator.hasNext()){
234                    Ponto p = iterator.next();
235                    energia[i] += Math.pow(p.pressure, 2);
236                    n++;
237                }
238            }
239
240            return energia;
241        }
242
243
244        /**
245         * Teste do Visual subcorpus do banco de dados SUSIG.
246         *
247         * @param args
248         */
249
250        public static void main(String[] args) {
251        /*
252
253            final int NUMERO_USUARIOS = 115;
254            final int NUMERO_AMOSTRAS_REFERENCIA = 5;
255
256            ProcessadorCadastramento proc = new ProcessadorCadastramento();
257
258            System.out.println↩
                   ("**************************************************");
259            for(int u = 1; u<=NUMERO_USUARIOS ; u++){
260
```

```
261                 String zeros = (u<10)?"00":((u<100)?"0":"");
262
263                 try{
264                     ConjuntoReferencia conjunto = new ConjuntoReferencia();
265                     for(int n = 1; n<= NUMERO_AMOSTRAS_REFERENCIA; n++){
266                         conjunto.add(new Assinatura("data/"+zeros+u+"_1_"+n+".sig", true, true, LAYOUT_STRING, " "));
267                     }
268
269                     (proc.processaCadastro(conjunto, false, 0)).salvarArquivo("data_proc/USER"+zeros+u+".cad");
270                     System.out.println("USER"+zeros+u);
271
272                 }
273                 catch(Throwable e){
274                     System.out.println("### ERRO: USER"+zeros+u);
275                     //e.printStackTrace();
276                 }
277
278                 System.out.println("*****************************************************");
279
280             }
281
282         */
283         }
284 }
```

Listing A.13: Classe: VetorCadastro.java

```
1  /**
2   * @author Augusto Cesar Gonçalves de Azevedo <augusto.azevedo at gmail.com>
3   *
4   * 12/09/2010
5   */
6  package com.augustoazevedo.signrec.cadastro;
7
8  import java.io.BufferedWriter;
9  import java.io.FileWriter;
10 import java.io.IOException;
11
12
13
14 /**
15  * @author Augusto Cesar Gonçalves de Azevedo <augusto.azevedo at gmail.com>
16  *
17  * 12/09/2010
18  *
19  */
20 public class VetorCadastro extends VetorCaracteristicas {
21     private final ConjuntoReferencia conjReferencia;
22     private final int templateIndex;
23
```

```
24      private final boolean usarFastDTW;
25      private final int raio;
26
27
28      /**
29       *
30       * @param minimo
31       * @param maximo
32       * @param template
33       * @param templateIndex
34       * @param energiaInterna
35       * @param conjReferencia
36       */
37      public VetorCadastro(double minimo, double maximo, double template, ↩
            int templateIndex, double[] energiaInterna, ConjuntoReferencia ↩
            conjReferencia, boolean usarFastDtw, int raio){
38          super(minimo, maximo, template, energiaInterna);
39          this.conjReferencia = conjReferencia;
40          this.templateIndex = templateIndex;
41          this.usarFastDTW = usarFastDtw;
42          this.raio = raio;
43      }
44
45      /**
46       *
47       * @param minimo
48       * @param maximo
49       * @param template
50       * @param templateIndex
51       * @param energiaInterna
52       * @param conjReferencia
53       */
54      public VetorCadastro(double minimo, double maximo, double template, ↩
            int templateIndex, double[] energiaInterna, ConjuntoReferencia ↩
            conjReferencia){
55          super(minimo, maximo, template, energiaInterna);
56          this.conjReferencia = conjReferencia;
57          this.templateIndex = templateIndex;
58          this.usarFastDTW = false;
59          this.raio = 0;
60      }
61
62
63  //  public VetorCadastro(String inputFile){
64  //      //TODO implementar conjunto de referencia
65  //        super(minimo, maximo, template, templateIndex, energiaInterna);
66  //        try
67  //        {
68  //            BufferedReader br = new BufferedReader (new FileReader (↩
        inputFile));
69  //            int size = Integer.valueOf(br.readLine());
70  //
71  //            double[] energiaInterna = new double[size-4];
72  //
73  //            double minimo = Double.valueOf(br.readLine());
74  //            double maximo = Double.valueOf(br.readLine());
```

```
75 //             double template = Double.valueOf(br.readLine());
76 //             int templateIndex = Integer.valueOf(br.readLine());
77 //
78 //             for(int i = 0; i<size; i++){
79 //               energiaInterna[i] = Double.valueOf(br.readLine());
80 //             }
81 //
82 //
83 //          }
84 //          catch (FileNotFoundException e){
85 //             throw new InternalError("ERROR:  The file '" + inputFile + "'↩
       was not found.");
86 //          }
87 //          catch (IOException e){
88 //             throw new InternalError("ERROR:  Problem reading the file '" ↩
      + inputFile + "'.");
89 //          }  // end try block
90 //   }
91
92      /**
93       * @return the conjReferencia
94       */
95      public ConjuntoReferencia getConjReferencia() {
96          return conjReferencia;
97      }
98
99
100     /**
101      * @return the templateIndex
102      */
103     public int getTemplateIndex() {
104         return templateIndex;
105     }
106
107
108     /**
109      * @return the usarFastDTW
110      */
111     public boolean isUsarFastDTW() {
112         return usarFastDTW;
113     }
114
115     /**
116      * @return the raio
117      */
118     public int getRaio() {
119         return raio;
120     }
121
122     /**
123      *
124      * @param outputFile
125      */
126     public void salvarArquivo(String outputFile){
127         try {
128             FileWriter file;
```

```
129             if(outputFile != null)
130                 file = new FileWriter(outputFile);
131             else
132                 throw new IOException("Erro ao lavar Cadastramento de ↩
                        usu rio. Arquivo de destino n o informado.");
133 
134             BufferedWriter out = new BufferedWriter(file);
135 
136             double[] energia = this.getEnergiaInterna();
137 
138             out.write((energia.length+4) + "\n");
139             out.write(this.getMinimo() + "\n");
140             out.write(this.getMaximo() + "\n");
141             out.write(this.getTemplate() + "\n");
142             out.write((this.getTemplateIndex()+1) + "\n"); // TODO retirar↩
                    +1
143 
144             for(int i = 0; i< energia.length; i++){
145                 out.write(energia[i] + "\n");
146             }
147 
148             out.close();
149         } catch (IOException e) {
150             System.out.println(e.getMessage());
151         }
152     }
153 
154 
155 }
```

Listing A.14: Classe: VetorCaracteristicas.java

```
1  /**
2   * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.↩
        com>
3   *
4   * 11/09/2010
5   */
6  package com.augustoazevedo.signrec.cadastro;
7 
8  import java.io.BufferedWriter;
9  import java.io.FileWriter;
10 import java.io.IOException;
11 
12 /**
13  * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.↩
        com>
14  *
15  * 11/09/2010
16  *
17  */
18 public class VetorCaracteristicas {
19 
20     private final double    minimo,
21                             maximo,
```

```
22                              template;
23
24      private final double[] energiaInterna;
25
26
27      /**
28       *
29       * @param minimo
30       * @param maximo
31       * @param template
32       * @param templateIndex
33       * @param energiaInterna
34       */
35      public VetorCaracteristicas(double minimo, double maximo, double ↩
            template, double[] energiaInterna){
36          this.minimo = minimo;
37          this.maximo = maximo;
38          this.template = template;
39
40          this.energiaInterna = energiaInterna;
41      }
42
43      /**
44       *
45       * @param outputFile
46       */
47      public void salvarArquivo(String outputFile){
48          try {
49              FileWriter file;
50              if(outputFile != null)
51                  file = new FileWriter(outputFile);
52              else
53                  throw new IOException("Erro ao lavar Cadastramento de ↩
                        usu rio. Arquivo de destino n o informado.");
54
55              BufferedWriter out = new BufferedWriter(file);
56
57              double[] energia = this.getEnergiaInterna();
58
59              out.write((energia.length+3) + "\n");
60              out.write(this.getMinimo() + "\n");
61              out.write(this.getMaximo() + "\n");
62              out.write(this.getTemplate() + "\n");
63
64              for(int i = 0; i< energia.length; i++){
65                  out.write(energia[i] + "\n");
66              }
67
68              out.close();
69          } catch (IOException e) {
70              System.out.println(e.getMessage());
71          }
72      }
73
74
75      /**
```

```
76          * @return the minimo
77          */
78         public double getMinimo() {
79             return minimo;
80         }
81 
82 
83         /**
84          * @return the maximo
85          */
86         public double getMaximo() {
87             return maximo;
88         }
89 
90 
91         /**
92          * @return the template
93          */
94         public double getTemplate() {
95             return template;
96         }
97 
98 
99         /**
100          * @return the energiaInterna
101          */
102         public double[] getEnergiaInterna() {
103             return energiaInterna;
104         }
105 
106 
107 
108 
109 }
```

A.1.4 Pacote: com.augustoazevedo.signrec.classificadores

- Classificador.java
- ClassificadorLinear.java
- ClassificadorSVM.java

Listing A.15: Classe: Classificador.java

```
1  /**
2   * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.↩
       com>
3   *
4   * 11/09/2010
5   */
6  package com.augustoazevedo.signrec.classificadores;
7  
```

```
8  import com.augustoazevedo.signrec.assinador.vo.Assinatura;
9  import com.augustoazevedo.signrec.cadastro.VetorCadastro;
10
11 /**
12  * @author Augusto Cesar Gonçalves de Azevedo <augusto.azevedo at gmail.com>
13  *
14  *         11/09/2010
15  *
16  */
17 public interface Classificador {
18
19     /**
20      * Este método deve receber como entrada uma assinatura atribuida a
21      * determinado usuário juntamente com as assinaturas que formam o conjunto
22      * de referência deste usuário.
23      *
24      * @param candidata
25      *            Assinatura a ser verificada.
26      * @param cadastro
27      *            Objeto do tipo {@link VetorCadastro} que contém o conjunto de
28      *            referência e o vetor de características do usuário.
29      * @return O método deve retornar <b>true</b> caso a assinatura pertença ao
30      *         usuário e <b>false</b> caso contrário.
31      */
32     public boolean classificar(Assinatura candidata,
33             VetorCadastro cadastro);
34
35 }
```

Listing A.16: Classe: ClassificadorLinear.java

```
1  /**
2  * @author Augusto Cesar Gonçalves de Azevedo <augusto.azevedo at gmail.com>
3  *
4  * 11/09/2010
5  */
6  package com.augustoazevedo.signrec.classificadores;
7
8
9  import com.augustoazevedo.signrec.assinador.vo.Assinatura;
10 import com.augustoazevedo.signrec.cadastro.ProcessadorCadastramento;
11 import com.augustoazevedo.signrec.cadastro.VetorCadastro;
12 import com.augustoazevedo.signrec.cadastro.VetorCaracteristicas;
13
14 /**
15  * @author Augusto Cesar Gonçalves de Azevedo <augusto.azevedo at gmail.com>
16  *
17  *         11/09/2010
18  *
```

```
19  */
20 public class ClassificadorLinear implements Classificador {
21
22     private static ClassificadorLinear instance = null;
23
24     public static final double THRESHOLD_EER = 1.05;
25     public static final double THRESHOLD = 1;//1.15 1.13
26
27     private ClassificadorLinear(){
28
29     }
30
31     public static ClassificadorLinear getInstance(){
32         if(instance == null)
33                 return (instance = new ClassificadorLinear());
34         return instance;
35     }
36
37     /*
38      * (non-Javadoc)
39      *
40      * @see
41      * com.augustoazevedo.signrec.classificadores.Classificador#↩
            classificar(
42      * com.augustoazevedo.signrec.assinador.vo.Assinatura, java.util.↩
            ArrayList)
43      */
44     @Override
45     public boolean classificar(Assinatura candidata,
46             VetorCadastro cadastro){
47
48         VetorCaracteristicas featureVector = ProcessadorCadastramento.↩
               comparaAssinatura(candidata, cadastro);
49
50         double  min = featureVector.getMinimo(),
51                 max = featureVector.getMaximo(),
52                 template = featureVector.getTemplate(),
53                 variancia = 0;
54
55         min = min / cadastro.getMinimo();
56         max = max / cadastro.getMaximo();
57         template = template / cadastro.getTemplate();
58
59         double mediaVar = (min + max + template)/3;
60         variancia = ((min - mediaVar)*(min - mediaVar) + (max - mediaVar)↩
               *(max - mediaVar) + (template - mediaVar)*(template - mediaVar↩
               ))/3;
61
62         if((variancia) < THRESHOLD)
63             return true;
64         return false;
65     }
66
67 }
```

Listing A.17: Classe: ClassificadorSVM.java

```
1  /**
2   * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.↩
       com>
3   *
4   * 13/09/2010
5   */
6  package com.augustoazevedo.signrec.classificadores;
7
8  import java.io.FileNotFoundException;
9
10 import libsvm.svm;
11 import libsvm.svm_model;
12 import libsvm.svm_node;
13
14 import com.augustoazevedo.signrec.assinador.vo.Assinatura;
15 import com.augustoazevedo.signrec.cadastro.ProcessadorCadastramento;
16 import com.augustoazevedo.signrec.cadastro.VetorCadastro;
17 import com.augustoazevedo.signrec.cadastro.VetorCaracteristicas;
18
19 /**
20  * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.↩
       com>
21  *
22  * 13/09/2010
23  *
24  */
25 public class ClassificadorSVM implements Classificador{
26
27
28     private static ClassificadorSVM instance = null;
29     private svm_model modelo = null;
30
31 //  private static final String ARQUIVO_MODELO = "src/com/augustoazevedo/↩
    signrec/classificadores/susig_visual.model"; //TODO ### arrumar ↩
    endere o arquivo
32     private static final String ARQUIVO_MODELO = "susig_visual.model";
33
34
35     private ClassificadorSVM() throws Exception{
36         modelo = carregarModelo(ARQUIVO_MODELO);
37     }
38
39     public static ClassificadorSVM getInstance(){
40         if(instance == null)
41             try{
42                 return (instance = new ClassificadorSVM());
43             } catch(Exception e){
44                 e.printStackTrace();
45             }
46         return instance;
47     }
48
49     /* (non-Javadoc)
```

```
50      * @see com.augustoazevedo.signrec.classificadores.Classificador#↩
            classificar(com.augustoazevedo.signrec.assinador.vo.Assinatura, ↩
            com.augustoazevedo.signrec.cadastro.VetorCadastro)
51      */
52     @Override
53     public boolean classificar(Assinatura candidata, VetorCadastro ↩
           cadastro) {
54
55         double[] parametros = getParametrosSVM(candidata, cadastro);
56
57         svm_node[] x = new svm_node[parametros.length];
58         for(int j = 0; j < parametros.length ; j++)
59         {
60             x[j] = new svm_node();
61             x[j].index = j + 1;
62             x[j].value = parametros[j];
63         }
64
65         if(svm.svm_predict(modelo,x) == 1d)
66             return true;
67
68         return false;
69     }
70
71
72     public static double[] getParametrosSVM(Assinatura candidata, ↩
           VetorCadastro cadastro){
73
74         VetorCaracteristicas featureVector = ProcessadorCadastramento.↩
               comparaAssinatura(candidata, cadastro);
75
76
77         //NORMALIZA O DOS PARMETROS B SICOS
78         double  min = featureVector.getMinimo(),
79                 max = featureVector.getMaximo(),
80                 template = featureVector.getTemplate(),
81                 variancia = 0;
82
83         min = min / cadastro.getMinimo();
84         max = max / cadastro.getMaximo();
85         template = template / cadastro.getTemplate();
86
87         double mediaVar = (min + max + template)/3;
88         variancia = ((min - mediaVar)*(min - mediaVar) + (max - mediaVar)↩
               *(max - mediaVar) + (template - mediaVar)*(template - mediaVar↩
               ))/3;
89
90
91         //CALCULOS DE ENERGIA
92         double[] energiaMedia = cadastro.getEnergiaInterna();
93         double[] energia = featureVector.getEnergiaInterna();
94         double[] energiaNormalizada = new double[energia.length];
95
96         double energiaTotal = 0, energiaMediaTotal = 0;
97         for(int x = 0; x < energia.length; x++){
98             energiaTotal += energia[x];
```

```
99              energiaMediaTotal += energiaMedia[x];
100             energiaNormalizada[x] = energia[x] / energiaMedia[x];
101         }
102
103         energiaTotal = energiaTotal / energiaMediaTotal;
104
105
106         //CONCATENA O DOS DADOS
107         int qtdParametros = energia.length + 5;
108         double[] parametros = new double[qtdParametros];
109         parametros[0] = template;
110         parametros[1] = min;
111         parametros[2] = max;
112         parametros[3] = variancia;
113         parametros[4] = energiaTotal;
114
115         for(int k = 0 ; k < energiaNormalizada.length ; k++){
116             parametros[k + 5] = energiaNormalizada[k];
117         }
118
119         return parametros;
120     }
121
122     private svm_model carregarModelo(String arquivoModelo) throws ↩
            Exception{
123         try{
124             svm_model model = svm.svm_load_model(arquivoModelo);
125
126             if(svm.svm_check_probability_model(model)!=0){
127                 System.out.print("Model supports probability estimates, ↩
                        but disabled in prediction.\n");
128             }
129             return model;
130         }
131         catch(FileNotFoundException e){
132             throw new Exception("Falaha ao carregar modelo. Arquivo n o ↩
                    encontrado.");
133         }
134         catch(ArrayIndexOutOfBoundsException e){
135             throw new Exception("Falaha ao carregar modelo.");
136         }
137     }
138
139 }
```

A.1.5 Pacote: com.augustoazevedo.signrec.dtw

- TimeSeries.java

Listing A.18: Classe:

```
1 /**
```

```
2  * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.com>
3  *
4  * 12/09/2010
5  */
6  package com.augustoazevedo.signrec.dtw;
7
8  import java.util.ArrayList;
9  import java.util.Iterator;
10
11 import com.augustoazevedo.signrec.assinador.vo.Assinatura;
12 import com.augustoazevedo.signrec.assinador.vo.Ponto;
13 import com.fastdtw.timeseries.TimeSeriesPoint;
14
15 /**
16  * Expans o da classe {@link com.timeseries.TimeSeries} para instancia  o
17  * a partir de uma {@link Assinatura}.
18  *
19  * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.com>
20  *
21  *         12/09/2010
22  *
23  */
24 public class TimeSeries extends com.fastdtw.timeseries.TimeSeries {
25
26     public TimeSeries(Assinatura assinatura, ArrayList<String> layout) {
27         super();
28
29         getLabels().add("Time");
30         for (int c = 1; c <= (layout.size()); c++)
31             getLabels().add(new String("c" + c));
32
33         Iterator<Ponto> iterator = assinatura.iterator();
34         while (iterator.hasNext()) {
35             Ponto p = iterator.next();
36
37             final ArrayList<Double> currentLineValues = new ArrayList<Double>();
38
39             if (layout.contains(Ponto.TIMESTAMP))
40                 currentLineValues.add(Double.valueOf(p.timestamp));
41             if (layout.contains(Ponto.DESLOCAMENTO))
42                 currentLineValues.add(Double.valueOf(p.deslocamento));
43             if (layout.contains(Ponto.DESLOCAMENTO_X))
44                 currentLineValues.add(Double.valueOf(p.deslocamento_x));
45             if (layout.contains(Ponto.DESLOCAMENTO_Y))
46                 currentLineValues.add(Double.valueOf(p.deslocamento_y));
47             if (layout.contains(Ponto.PRESSURE))
48                 currentLineValues.add(Double.valueOf(p.pressure));
49             if (layout.contains(Ponto.X))
50                 currentLineValues.add(Double.valueOf(p.x));
51             if (layout.contains(Ponto.Y))
52                 currentLineValues.add(Double.valueOf(p.y));
53
```

```
54              getTimeReadings().add(new Double(getTimeReadings().size()));
55
56              final TimeSeriesPoint readings = new TimeSeriesPoint(↩
                    currentLineValues.subList(0,currentLineValues.size()));
57              getTsArray().add(readings);
58
59          }
60
61      }
62
63      /**
64       * @param numOfDimensions
65       */
66      public TimeSeries(int numOfDimensions) {
67          super(numOfDimensions);
68          // TODO Auto-generated constructor stub
69      }
70
71      /**
72       * @param origTS
73       */
74      public TimeSeries(com.fastdtw.timeseries.TimeSeries origTS) {
75          super(origTS);
76          // TODO Auto-generated constructor stub
77      }
78
79      /**
80       * @param inputFile
81       * @param isFirstColTime
82       * @throws Throwable
83       */
84      public TimeSeries(String inputFile, boolean isFirstColTime)
85              throws Throwable {
86          super(inputFile, isFirstColTime);
87          // TODO Auto-generated constructor stub
88      }
89
90      /**
91       * @param inputFile
92       * @param delimiter
93       * @throws Throwable
94       */
95      public TimeSeries(String inputFile, char delimiter) throws Throwable {
96          super(inputFile, delimiter);
97          // TODO Auto-generated constructor stub
98      }
99
100     /**
101      * @param inputFile
102      * @param isFirstColTime
103      * @param delimiter
104      * @throws Throwable
105      */
106     public TimeSeries(String inputFile, boolean isFirstColTime, char ↩
            delimiter)
107             throws Throwable {
```

```
108         super(inputFile, isFirstColTime, delimiter);
109         // TODO Auto-generated constructor stub
110     }
111 
112     /**
113      * @param inputFile
114      * @param colToInclude
115      * @param isFirstColTime
116      * @throws Throwable
117      */
118     public TimeSeries(String inputFile, int[] colToInclude,
119             boolean isFirstColTime) throws Throwable {
120         super(inputFile, colToInclude, isFirstColTime);
121         // TODO Auto-generated constructor stub
122     }
123 
124     /**
125      * @param inputFile
126      * @param isFirstColTime
127      * @param isLabeled
128      * @param delimiter
129      * @throws Throwable
130      */
131     public TimeSeries(String inputFile, boolean isFirstColTime,
132             boolean isLabeled, char delimiter) throws Throwable {
133         super(inputFile, isFirstColTime, isLabeled, delimiter);
134         // TODO Auto-generated constructor stub
135     }
136 
137     /**
138      * @param inputFile
139      * @param colToInclude
140      * @param isFirstColTime
141      * @param isLabeled
142      * @param delimiter
143      * @throws Throwable
144      */
145     public TimeSeries(String inputFile, int[] colToInclude,
146             boolean isFirstColTime, boolean isLabeled, char delimiter)
147             throws Throwable {
148         super(inputFile, colToInclude, isFirstColTime, isLabeled, ↩
                delimiter);
149         // TODO Auto-generated constructor stub
150     }
151 
152 }
```

A.1.6 Pacote: com.augustoazevedo.signrec.util

- AssinaturaRenderizadorSolo.java
- ErroResultanteSVM.java
- ExportadorDados.java

- GeradorArquivosSVM.java
- GerenciadorTeste.java
- VizualizadorAssinatura.java

Listing A.19: Classe: AssinaturaRenderizadorSolo.java

```
1  package com.augustoazevedo.signrec.util;
2  import javax.swing.JFrame;
3
4  import com.augustoazevedo.signrec.assinador.vo.Assinatura;
5  import com.augustoazevedo.signrec.frame.AssinaturaRenderizador;
6
7      public class AssinaturaRenderizadorSolo extends AssinaturaRenderizador↩
           {
8
9          /**
10          *
11          */
12         private static final long serialVersionUID = 1L;
13
14         public AssinaturaRenderizadorSolo(Assinatura assinatura, String ↩
               tituloJanela){
15             super(assinatura);
16             f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
17             f.setTitle(tituloJanela);
18             f.setSize(500,300);
19         }
20
21     }
```

Listing A.20: Classe: ErroResultanteSVM.java

```
1  package com.augustoazevedo.signrec.util;
2  import java.io.BufferedReader;
3  import java.io.FileNotFoundException;
4  import java.io.FileReader;
5  import java.io.IOException;
6
7
8  public class ErroResultanteSVM {
9
10     private static final int ASSINATURAS_VERDADEIRAS = 15,
11                              ASSINATURAS_FALSAS = 10,
12                              TOTAL_ASSINATURAS = 25;
13
14     private static int  contador = 0,
15                         falsosPositivos = 0,
16                         falsosNegativos = 0;
17
18
19     /**
```

```
20      * @param args
21      */
22     public static void main(String[] args) {
23          try {
24             BufferedReader br = new BufferedReader (new FileReader ("svm/↩
                   susig_visual.t.predict"));
25
26              String linha = null;
27              while((linha = br.readLine()) != null){
28                  contador++;
29
30                  int n = contador % (ASSINATURAS_VERDADEIRAS + ↩
                        ASSINATURAS_FALSAS);
31
32
33                  if(n > 0 && n <= ASSINATURAS_VERDADEIRAS){
34                      if(Double.valueOf(linha) == 0d)
35                          falsosNegativos++;
36                  }
37                  else
38                      if(Double.valueOf(linha) == 1d){
39                          falsosPositivos++;
40                          System.out.println(contador);
41                      }
42              }
43
44              System.out.println("↩
                    ------------------------------------------------------");
45              System.out.println("F.Pos      : " + falsosPositivos + " \tFAR↩
                    : " + ((double) falsosPositivos/contador));
46              System.out.println("F.Neg      : " + falsosNegativos + " \tFRR↩
                    : " + ((double) falsosNegativos/contador));
47              System.out.println("ERRO TOTAL: " + ((double) (↩
                    falsosNegativos+falsosPositivos)/contador));
48              System.out.println("             " + (1d-((double) (↩
                    falsosNegativos+falsosPositivos)/contador)));
49              System.out.println("TOTAL      : " + contador);
50              System.out.println("↩
                    ------------------------------------------------------");
51  //              System.out.println("% treino   : " + ((double) (↩
        TOTAL_ASSINATURAS-(ASSINATURAS_VERDADEIRAS+ASSINATURAS_FALSAS))/↩
        TOTAL_ASSINATURAS));
52  //              System.out.println("% teste    : " + ((double) ((↩
        ASSINATURAS_VERDADEIRAS+ASSINATURAS_FALSAS))/TOTAL_ASSINATURAS));
53              System.out.println("% treino   : " + ((double) (↩
                    TOTAL_ASSINATURAS*67d)/(TOTAL_ASSINATURAS*94)));
54              System.out.println("% teste    : " + ((double) (↩
                    TOTAL_ASSINATURAS*27d)/(TOTAL_ASSINATURAS*94)));
55
56          } catch (FileNotFoundException e) {
57              // TODO Auto-generated catch block
58              e.printStackTrace();
59          } catch (IOException e) {
60              // TODO Auto-generated catch block
61              e.printStackTrace();
62          }
```

```
63
64
65      }
66
67  }
```

Listing A.21: Classe: ExportadorDados.java

```
1   package com.augustoazevedo.signrec.util;
2   import java.io.BufferedWriter;
3   import java.io.FileWriter;
4   import java.io.IOException;
5
6   import com.augustoazevedo.signrec.assinador.vo.Assinatura;
7   import com.augustoazevedo.signrec.cadastro.ProcessadorCadastramento;
8   import com.augustoazevedo.signrec.cadastro.VetorCadastro;
9   import com.augustoazevedo.signrec.classificadores.ClassificadorSVM;
10
11  /**
12   *
13   * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.↩
         com>
14   *
15   * 17/09/2010
16   *
17   */
18  public class ExportadorDados {
19
20      private static StringBuffer autenticas = new StringBuffer(2048);
21      private static StringBuffer falsificacoes = new StringBuffer(2048);
22
23
24      /**
25       *
26       * @param args
27       */
28      public static void main(String[] args) {
29
30          int cont = 0;
31
32          System.out.println("↩
                ************************************************");
33          for(int u = 1; u <= GerenciadorTeste.NUMERO_USUARIOS; u++){
34              try {
35                  String zeros = (u<10)?"00":((u<100)?"0":"");
36                  String usuario = new String(zeros+""+u);
37
38                  VetorCadastro cadastro = GerenciadorTeste.montarCadastro(↩
                        usuario);
39
40                  for(int n = (GerenciadorTeste.NUMERO_AMOSTRAS_REFERENCIA +↩
                         1); n <= GerenciadorTeste.NUMERO_AMOSTRAS; n++){
41
42                      String grupo = "";
43                      int sigNro = n;
```

```
44                      if(n<11){
45                          grupo = "_1_";
46                      }
47                      else if(n<21){
48                          grupo = "_2_";
49                          sigNro -= 10;
50                      }
51                      else{
52                          grupo = "_f_";
53                          sigNro -= 20;
54                      }
55
56                      Assinatura assinatura = new Assinatura("data/"+usuario↩
                            +grupo+sigNro+".sig", true, true, ↩
                            ProcessadorCadastramento.LAYOUT_STRING, " ");
57
58                      double[] parametros = ClassificadorSVM.↩
                            getParametrosSVM(assinatura, cadastro);
59
60                      String parametrosStr = "";
61                      for(int i = 1; i <= parametros.length; i++){
62                          parametrosStr = parametrosStr.concat(parametros[i↩
                                -1] + " ");
63                      }
64
65                      String saida = new String(parametrosStr);
66
67                      System.out.println("U"+u+" A"+n+"  "+(u>67?cont++:"")+↩
                            " \t: " + saida);
68
69
70                      if(n < 21)
71                          autenticas.append(saida + "\n");
72                      else
73                          falsificacoes.append(saida + "\n");
74
75                  }
76
77
78              } catch (Throwable e) {
79                  //System.out.println("### ERRO User: " + u);
80                  //e.printStackTrace();
81              }
82          }
83
84          salvarArquivos();
85
86      }
87
88
89      private static void salvarArquivos(){
90
91          try {
92
93              FileWriter file = new FileWriter("autenticas.dat");
94              BufferedWriter out = new BufferedWriter(file);
```

```
95              out.write(autenticas.toString());
96              out.close();
97
98              file = new FileWriter("falsas.dat");
99              out = new BufferedWriter(file);
100             out.write(falsificacoes.toString());
101             out.close();
102
103
104         } catch (IOException e) {
105             System.out.println(e.getMessage());
106         }
107     }
108
109 }
```

Listing A.22: Classe: GeradorArquivosSVM.java

```
1  package com.augustoazevedo.signrec.util;
2  import java.io.BufferedWriter;
3  import java.io.FileWriter;
4  import java.io.IOException;
5
6  import com.augustoazevedo.signrec.assinador.vo.Assinatura;
7  import com.augustoazevedo.signrec.cadastro.ProcessadorCadastramento;
8  import com.augustoazevedo.signrec.cadastro.VetorCadastro;
9  import com.augustoazevedo.signrec.classificadores.ClassificadorSVM;
10
11 /**
12  * Classe que gera os arquivos para treinamento e avalia  o do ↩
       classificador SVM.
13  *
14  * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.↩
       com>
15  *
16  * 11/09/2010
17  *
18  */
19 public class GeradorArquivosSVM {
20
21     private static StringBuffer treinamento = new StringBuffer(2048);
22     private static StringBuffer validacao = new StringBuffer(2048);
23
24
25     /**
26      *
27      * @param args
28      */
29     public static void main(String[] args) {
30
31         int cont = 0;
32
33         System.out.println("↩
               ************************************************");
34         for(int u = 1; u <= GerenciadorTeste.NUMERO_USUARIOS; u++){
```

```
35              try {
36                  String zeros = (u<10)?"00":((u<100)?"0":"");
37                  String usuario = new String(zeros+""+u);
38
39                  VetorCadastro cadastro = GerenciadorTeste.montarCadastro(↩
                        usuario);
40
41                  for(int n = (GerenciadorTeste.NUMERO_AMOSTRAS_REFERENCIA +↩
                         1); n <= GerenciadorTeste.NUMERO_AMOSTRAS; n++){
42
43                      String grupo = "";
44                      int sigNro = n;
45                      if(n<11){
46                          grupo = "_1_";
47                      }
48                      else if(n<21){
49                          grupo = "_2_";
50                          sigNro -= 10;
51                      }
52                      else{
53                          grupo = "_f_";
54                          sigNro -= 20;
55                      }
56
57                      Assinatura assinatura = new Assinatura("data/"+usuario↩
                            +grupo+sigNro+".sig", true, true, ↩
                            ProcessadorCadastramento.LAYOUT_STRING, " ");
58
59                      double[] parametros = ClassificadorSVM.↩
                            getParametrosSVM(assinatura, cadastro);
60
61                      String parametrosStr = "";
62                      for(int i = 1; i <= parametros.length; i++){
63                          parametrosStr = parametrosStr.concat(" "+i+":"+↩
                                parametros[i-1]);
64                      }
65
66                      String saida = new String((n<21?"1":"0") + ↩
                            parametrosStr);
67
68                      System.out.println("U"+u+" A"+n+"  "+(u>67?cont++:"")+↩
                            " \t: " + saida);
69
70
71                      if(u <= 67)
72                          treinamento.append(saida + "\n");
73                      else
74                          validacao.append(saida + "\n");
75
76                  }
77
78
79              } catch (Throwable e) {
80                  //System.out.println("### ERRO User: " + u);
81                  //e.printStackTrace();
82              }
```

```
83          }
84
85          salvarArquivos();
86
87      }
88
89
90      private static void salvarArquivos(){
91
92          try {
93
94              FileWriter file = new FileWriter("svm/susig_visual");
95              BufferedWriter out = new BufferedWriter(file);
96              out.write(treinamento.toString());
97              out.close();
98
99              file = new FileWriter("svm/susig_visual.t");
100             out = new BufferedWriter(file);
101             out.write(validacao.toString());
102             out.close();
103
104
105         } catch (IOException e) {
106             System.out.println(e.getMessage());
107         }
108     }
109
110 }
```

Listing A.23: Classe: GerenciadorTeste.java

```
1  package com.augustoazevedo.signrec.util;
2  import java.io.BufferedWriter;
3  import java.io.FileWriter;
4  import java.io.IOException;
5  import java.sql.Date;
6  import java.text.SimpleDateFormat;
7
8  import com.augustoazevedo.signrec.assinador.vo.Assinatura;
9  import com.augustoazevedo.signrec.cadastro.ConjuntoReferencia;
10 import com.augustoazevedo.signrec.cadastro.ProcessadorCadastramento;
11 import com.augustoazevedo.signrec.cadastro.VetorCadastro;
12 import com.augustoazevedo.signrec.classificadores.Classificador;
13 import com.augustoazevedo.signrec.classificadores.ClassificadorSVM;
14
15 /**
16  *
17  * @author Augusto C sar Gon alves de Azevedo <augusto.azevedo at gmail.↩
        com>
18  *
19  * 14/09/2010
20  *
21  */
22 public class GerenciadorTeste {
23
```

```
24      public static final int NUMERO_USUARIOS = 115;
25      public static final int NUMERO_AMOSTRAS_REFERENCIA = 5;
26      public static final int NUMERO_AMOSTRAS = 30;
27
28      private static int  contador = 0,
29                          falsosPositivos = 0,
30                          falsosNegativos = 0;
31
32      //private static final Classificador classificador = ↩
            ClassificadorLinear.getInstance();
33      private static final Classificador classificador = ClassificadorSVM.↩
            getInstance();
34
35      private static final boolean USAR_FASTDTW = false;
36      private static final int RAIO_FASTDTW = 0;
37
38
39
40      /**
41       * @param args
42       */
43      public static void main(String[] args) {
44
45          System.out.println("↩
                **********************************************");
46          for(int u = 67; u<=NUMERO_USUARIOS ; u++){
47              try {
48                  String zeros = (u<10)?"00":((u<100)?"0":"");
49                  String usuario = new String(zeros+""+u);
50
51                  VetorCadastro cadastro = montarCadastro(usuario);
52
53                  for(int n = (NUMERO_AMOSTRAS_REFERENCIA + 1); n <= ↩
                        NUMERO_AMOSTRAS; n++){
54
55                      String grupo = "";
56                      int sigNro = n;
57                      if(n<11){
58                          grupo = "_1_";
59                      }
60                      else if(n<21){
61                          grupo = "_2_";
62                          sigNro -= 10;
63                      }
64                      else{
65                          grupo = "_f_";
66                          sigNro -= 20;
67                      }
68
69                      Assinatura assinatura = new Assinatura("data/"+usuario↩
                            +grupo+sigNro+".sig", true, true, ↩
                            ProcessadorCadastramento.LAYOUT_STRING, " ");
70
71                      boolean CLASSIFICACAO = classificador.classificar(↩
                            assinatura, cadastro);
72
```

```
73                      contador++;
74
75                      if(n < 21 && !CLASSIFICACAO)
76                          falsosNegativos++;
77
78                      else if(n > 20 && CLASSIFICACAO)
79                          falsosPositivos++;
80
81                      System.out.println("U"+u+" A"+n+";" + (n<21?true:false↩
                            ) + ";" + CLASSIFICACAO);
82                  }
83
84              } catch (Throwable e) {
85                  System.out.println("### ERRO User: " + u);
86                  //e.printStackTrace();
87              }
88          }
89
90          System.out.println("↩
                ************************************************");
91          System.out.println("CLASSIFICADOR\t: " + classificador.getClass())↩
                ;
92
93          System.out.println("TOTAL \t\t: " + (contador));
94          System.out.println("FA    \t\t: " + (falsosPositivos));
95          System.out.println("FR    \t\t: " + (falsosNegativos));
96
97          System.out.println("FAR   \t\t: " + ((double)falsosPositivos / ↩
                contador));
98          System.out.println("FRR   \t\t: " + ((double)falsosNegativos / ↩
                contador));
99          System.out.println("↩
                ************************************************");
100
101         //salvarLog(classificador);
102
103     }
104
105     /**
106      *
107      * @param usuario
108      * @return
109      */
110     public static VetorCadastro montarCadastro(String usuario) {
111         ProcessadorCadastramento proc = new ProcessadorCadastramento();
112         ConjuntoReferencia conjunto = new ConjuntoReferencia();
113         for(int n = 1; n<= NUMERO_AMOSTRAS_REFERENCIA; n++){
114             conjunto.add(new Assinatura("data/"+usuario+"_1_"+n+".sig", ↩
                    true, true, ProcessadorCadastramento.LAYOUT_STRING, " "));
115         }
116         return proc.processaCadastro(conjunto, USAR_FASTDTW, RAIO_FASTDTW)↩
                ;
117     }
118
119     private static void salvarLog(Classificador classificador){
120
```

```
121         SimpleDateFormat formatador = new SimpleDateFormat("yyyyMMdd_HHmmss");
122         Date data = new Date(System.currentTimeMillis());
123 
124         try {
125             String  far = String.valueOf((double)falsosPositivos / contador),
126                     frr = String.valueOf((double)falsosNegativos / contador);
127 
128             far = far.length()>5?far.substring(0,5):far;
129             frr = frr.length()>5?frr.substring(0,5):frr;
130 
131             FileWriter file = new FileWriter("log/teste_"+classificador.getClass()+"_[FAR "+ far +"][FRR "+ frr +"][" + formatador.format(data) + "].txt");
132 
133             BufferedWriter out = new BufferedWriter(file);
134 
135 
136             out.write("*************************************************" + "\n");
137             out.write("CLASSIFICADOR \t: " + classificador.getClass() + "\n");
138 
139             out.write("DATA  \t\t: " + formatador.format(data) + "\n");
140             out.write("TOTAL \t\t: " + (contador) + "\n");
141             out.write("FA    \t\t: " + (falsosPositivos) + "\n");
142             out.write("FR    \t\t: " + (falsosNegativos) + "\n");
143             out.write("FAR   \t\t: " + ((double)falsosPositivos / contador) + "\n");
144             out.write("FRR   \t\t: " + ((double)falsosNegativos / contador) + "\n");
145             out.write("*************************************************" + "\n");
146 
147             out.close();
148         } catch (IOException e) {
149             System.out.println(e.getMessage());
150         }
151     }
152 
153 }
```

Listing A.24: Classe: VizualizadorAssinatura.java

```
1  package com.augustoazevedo.signrec.util;
2 
3  import com.augustoazevedo.signrec.assinador.vo.Assinatura;
4  import com.augustoazevedo.signrec.assinador.vo.Ponto;
5 
6 
7  public class VizualizadorAssinatura {
8 
```

```
9       //private static String[] layout = {Ponto.X, Ponto.Y, Ponto.TIMESTAMP,↩
            null, null, null, Ponto.PRESSURE};
10      private static String[] layout = {Ponto.X, Ponto.Y, Ponto.TIMESTAMP, ↩
            Ponto.PRESSURE, null};
11
12
13      /**
14       * @param args
15       */
16      public static void main(String[] args) {
17
18  //      new AssinaturaRenderizadorSolo(new Assinatura("data/U35S35.TXT", ↩
        false, true, layout, " "),"35");
19  //      new AssinaturaRenderizadorSolo(new Assinatura("data/U35S28.TXT", ↩
        false, true, layout, " "),"28");
20  //      new AssinaturaRenderizadorSolo(new Assinatura("data/U35S27.TXT", ↩
        false, true, layout, " "),"27");
21  //      new AssinaturaRenderizadorSolo(new Assinatura("data/U35S12.TXT", ↩
        false, true, layout, " "),"12");
22  //      new AssinaturaRenderizadorSolo(new Assinatura("data/U35S11.TXT", ↩
        false, true, layout, " "),"11");
23  //      new AssinaturaRenderizadorSolo(new Assinatura("data/U35S10.TXT", ↩
        false, true, layout, " "),"10");
24  //
25  //      new AssinaturaRenderizadorSolo(new Assinatura("data/U35S1.TXT", ↩
        false, true, layout, " "),"1");
26  //      new AssinaturaRenderizadorSolo(new Assinatura("data/U35S2.TXT", ↩
        false, true, layout, " "),"2");
27  //      new AssinaturaRenderizadorSolo(new Assinatura("data/U35S3.TXT", ↩
        false, true, layout, " "),"3");
28  //      new AssinaturaRenderizadorSolo(new Assinatura("data/U35S4.TXT", ↩
        false, true, layout, " "),"4");
29  //      new AssinaturaRenderizadorSolo(new Assinatura("data/U35S5.TXT", ↩
        false, true, layout, " "),"5");
30
31
32          new AssinaturaRenderizadorSolo(new Assinatura("data/019_2_8.sig", ↩
                true, true, layout, " "),"18");
33          new AssinaturaRenderizadorSolo(new Assinatura("data/019_2_9.sig", ↩
                true, true, layout, " "),"19");
34          new AssinaturaRenderizadorSolo(new Assinatura("data/019_2_10.sig",↩
                 true, true, layout, " "),"20");
35          new AssinaturaRenderizadorSolo(new Assinatura("data/019_f_1.sig", ↩
                true, true, layout, " "),"21");
36          new AssinaturaRenderizadorSolo(new Assinatura("data/019_f_2.sig", ↩
                true, true, layout, " "),"22");
37          new AssinaturaRenderizadorSolo(new Assinatura("data/019_f_3.sig", ↩
                true, true, layout, " "),"23");
38          new AssinaturaRenderizadorSolo(new Assinatura("data/019_f_4.sig", ↩
                true, true, layout, " "),"24");
39
40
41
42      }
43  }
```

Referências

[1] R. Bellman and Rand Corporation. *Dynamic programming*. Princeton University Press, Princeton, 1957. 26

[2] S. Chu, E. Keogh, D. Hart, and M. Pazzani. Iterative deepening dynamic time warping for time series. In *In Proc. 2nd SIAM International Conference on Data Mining*, Arlington, Virginia, USA, 2002. 13, 16

[3] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. 7

[4] J G A Dolfing, E H L Aarts, and J J G M van Oosterhout. On-line signature verification with hidden markov models. *Pattern Recognition, International Conference on*, 2:1309, 1998. 6

[5] A. Eije. *Dominando o OpenSwing*. Ciência Moderna, 1 edition, 2010. 37

[6] K. Huang and H. Yan. Stability and style-variation modeling for on-line signature verification. *Pattern Recognition*, 36(10):2253–2270, 2003. 5

[7] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):67–72, 1975. 16

[8] A. K. Jain, F. D. Griess, and S. D. Connell. On-line signature verification. *International Journal of Pattern Recognition and Artificial Intelligence*, 11(4):549–569, 2002. 3, 6

[9] E. Keogh and M. Pazzani. Scaling up dynamic time warping for datamining applications. In *In Proc. 6th Internation Conference on Knowledge Discovery and Data Mining*, pages 285–289, Boston, Massachusetts, USA, 2000. 13, 16

[10] A. Kholmatov and B. Yanikoglu. Identity authentication using improved online signature verification method. *Pattern Recognition Letters*, 26(15):2400–2408, 2005. vii, 5, 7, 21, 23, 24, 25, 26, 27, 28

[11] A. Kholmatov and B. Yanikoglu. Susig: an on-line signature database, associated protocols and benchmark results. *Pattern Analysis and Applications*, 12(3):227–236, 2009. 22, 60

[12] H. Lei and V. Govindaraju. A comparative study on the consistency of features in on-line signature verification. *Pattern Recognition Letters*, 26(15):2483–2489, 2005. 6

[13] J. C. Martínez-Romo, F. J. Luna-Rosas, and M. Mora-González. On-line signature verification based on genetic optimization and neural-network-driven fuzzy reasoning. In *Proceedings of the 8th Mexican International Conference on Artificial Intelligence*, MICAI '09, pages 246–257, Berlin, Heidelberg, 2009. Springer-Verlag. 5

[14] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, March 1970. 12

[15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 3 edition, 2007. vii, 9, 10, 11

[16] N. K. Ratha, A. Senior, and R. M. Bolle. Automated biometrics. In *2000 Biometrics Consortium Workshop*, pages 445–453. Springer-Verlag, 2001. 2

[17] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26:43–49, 1978. 16

[18] S. Salvador and P. Chan. Fastdtw: Toward accurate dynamic time warping in linear time and space. *Intell. Data Anal.*, 11(5):561–580, 2007. vii, ix, 9, 14, 15, 16, 17, 19, 20, 27

[19] X. Xiao and G. Leedham. Signature verification using a modified bayesian network. *Pattern Recognition*, 35(5):983–995, 2002. 5, 6

[20] L. Yang, B. Widjaja, and R. Prasad. Application of hidden markov models for signature verification. *Pattern Recognition*, 28(2):161–170, 1995. 6

[21] L. Yang, B. K. Widjaja, and R. Prasad. Application of hidden markov models for signature verification. *Pattern Recognition*, 28(2):161–170, 1995. 5