UNIVERSIDADE FEDERAL DE ALAGOAS

INSTITUTO DE MATEMÁTICA

PROGRAMA DE PÓS GRADUAÇÃO EM MATEMÁTICA

SAMUEL ROCHA SILVA

RECONHECIMENTO DE GESTOS CUSTOMIZADOS DA MÃO EM TEMPO REAL USANDO APRENDIZADO DE MÉTRICAS E GRAFOS DE AÇÃO

MACEIÓ - AL

SETEMBRO DE 2017

SAMUEL ROCHA SILVA

RECONHECIMENTO DE GESTOS CUSTOMIZADOS DA MÃO EM TEMPO REAL USANDO APRENDIZADO DE MÉTRICAS E GRAFOS DE AÇÃO

Dissertação de Mestrado submetida em Setembro de 2017 à Banca Examinadora, designada pelo Colegiado do Programa de Pós-Graduação em Matemática da Universidade Federal de Alagoas, como parte dos requisitos necessários à obtenção do grau de Mestre em Matemática.

Orientador: Prof. Dr. Thales Miranda de Almeida Vieira

MACEIÓ - AL

SETEMBRO DE 2017

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária Responsável: Helena Cristina Pimentel do Vale

S586r Silva, Samuel Rocha.
Reconhecimento de gestos customizados da mão em tempo real usando aprendizado de métrica e grafos de ação / Samuel Rocha Silva. – 2017.
111 f.

Orientador: Thales Miranda de Almeida Vieira.
Dissertação (mestrado em Matemática) – Universidade Federal de Alagoas. Instituto de Matemática. Maceió, 2017.

Bibliografia: f. 108-111.

1. Língua brasileira de sinais. 2. Leap motion. 3. Aprendizagem de máquina. 4. Reconhecimento de gestos. 5. Aprendizagem de métrica. I. Título.

CDU: 51:004.9

SAMUEL ROCHA SILVA

RECONHECIMENTO DE GESTOS CUSTOMIZADOS DA MÃO EM TEMPO REAL USANDO APRENDIZADO DE MÉTRICAS E GRAFOS DE AÇÃO

Dissertação de Mestrado submetida em Setembro de 2017 à Banca Examinadora, designada pelo Colegiado do Programa de Pós-Graduação em Matemática da Universidade Federal de Alagoas, como parte dos requisitos necessários à obtenção do grau de Mestre em Matemática.

Banca examinadora:

Prof. Dr. Adelailson Peixoto da Silva
UFAL

Prof. Dr. Dimas Martinez Morera
UFAM

Prof. Dr. Tiago Figueiredo Vieira
UFAL

Prof. Dr. Thales Miranda de Almeida Vieira
Orientador
UFAL

À minha mãe, irmãos e a toda minha família que, com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa de minha vida.

Agradecimentos

Primeiramente à Deus, por me conceder saúde e disposição para vencer mais um desafio em minha vida.

À minha mãe, que através de sua simplicidade e imensa bondade me ensinou os verdadeiros valores da vida. Aos meus irmãos, cunhados e demais familiares, que sempre me apoiaram durante essa jornada. À minha namorada, por seu amor, carinho e incentivo, que sempre renovavam minhas energias nos momentos de maiores difilculdades.

Ao meu orientador, Prof. Dr. Thales Vieira, pelo seu apoio, comprometimento e pelas várias reuniões que foram de suma importância para o desenvolvimento deste trabalho.

Aos Profs. Dr. Adelailson Peixoto, Dr. Dimas Martinez e Dr. Tiago Vieira por aceitarem compor a banca examinadora deste trabalho. Em especial ao Prof. Dr. Tiago pelas diversas correções e sugestões.

Ao Prof. Adriano Valeriano pelo constante apoio e incentivo ao meu crescimento científico e intelectual durante nossos mais de 10 anos de convivência.

À todos que fizeram e fazem parte do Laboratório de Computação Gráfica, pelos momentos de descontração e aprendizado.

Aos amigos do Instituto de Matemática e CPMAT, em especial ao Diego, Matheus, Robson e Vinícius, que através de nossos diversos momentos de descontração tornaram esta jornada mais leve.

Aos docentes do Instituto de Matemática que tanto contribuíram na minha formação acadêmica e à todos que fazem a coordenação do programa, que me ajudaram com esclarecimentos e com o acompanhamento de minha vida acadêmica junto ao departamento.

Por fim, agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo suporte financeiro durante estes 2 anos que estive com bolsa de mestrado PICME.

Resumo

Devido à recente popularização dos sensores de profundidade e de movimento, o reconhecimento eficiente de gestos humanos tem se apresentado como uma alternativa para interação homem-máquina. Nesta dissertação tratamos, em particular, do reconhecimento de gestos da mão em tempo real utilizando o sensor Leap Motion. Consideramos que um gesto dinâmico, em geral, pode ser representado por uma pequena sequência finita de poses estáticas, as quais são treinadas e reconhecidas por um novo classificador baseado em aprendizagem de métricas denominado *Classificador de Distâncias Médias Mínimas* (DMM). Para aprender métricas no espaço de poses, experimentamos o algoritmo *Large Margin Nearest Neighbor* (LMNN) nas versões linear e não-linear. Para reconhecer gestos dinâmicos, apresentamos um classificador baseado nos *Modelos ocultos de Markov* (HMM) e em *grafos de ação*. Com esta abordagem probabilística, mostramos que gestos dinâmicos da mão podem ser reconhecidos em tempo real e online, ou seja, sem a necessidade de pausas ou segmentações extras por parte do usuário. Finalmente, experimentamos o método proposto em quatro aplicações, onde destacamos o reconhecimento de sinais da *Língua Brasileira de Sinais* (LIBRAS). Os resultados apresentados mostram a robustez de todas as variações do método proposto.

Palavras-chaves: língua brasileira de sinais, leap motion, aprendizagem de máquina, reconhecimento de gestos, aprendizagem de métrica.

Abstract

The recent popularization of depth and motion sensors, human gestures recognition has become an alternative for human-computer interaction. In this Masters dissertation we focus on hand gesture recognition in real time using the Leap Motion sensor. We consider that a dynamic gesture can be represented by a small finite sequence of static poses, which are trained and recognized by a novel classifier based on metric learning called *Minimal Mean Distances Classifier* (MMD). To learn metrics in the pose space, we experimented the *Large Margin Nearest Neighbor* (LMNN) algorithm in both linear and non-linear variants. To recognize dynamic gestures, we present a classifier based on the *Hidden Markov Model* (HMM) and action graphs. Using this probabilistic approach, we show that dynamic hand gestures can be recognized in real-time and online (not requiring pauses or segmentation information to the user). Finally, we experimented the proposed method in four applications, highlighting the recognition of signals of the *Brazilian Sign Language* (LIBRAS). The presented results show the robustness of all variations of the proposed method.

Keywords: brazilian sign language, leap motion, machine learning, hand gesture, metric learning.

Lista de ilustrações

Lista de tabelas

Sumário

INTRODUÇÃO

O reconhecimento de gestos vem se apresentando como uma nova altenativa para a interação homem-máquina, possibilitando o desenvolvimento de novas aplicações em realidade virtual, interfaces naturais de usuário, jogos imersivos e reconhecimento de línguas de sinais. Graças à recente popularização de sensores, tais como Microsoft Kinect [1], Intel RealSense 3D [2] ou Leap Motion [3] (Figura 1), por exemplo, ocorreram muitos avanços nesta área. Estes sensores são capazes de detectar e rastrear diferentes partes do corpo humano. No entanto, o reconhecimento de gestos humanos ainda é uma tarefa difícil e subjetiva, pois um mesmo gesto pode ser realizado por diferentes usuários com diversas variações.

Embora muitos avanços tenham sido alcançados no que diz respeito ao reconhecimento de gestos corporais utilizando sensores ópticos [4], o reconhecimento robusto de gestos da mão utilizando estes sensores ainda é um problema pouco explorado. Em geral, sensores ópticos tendem a ter dificuldade em rastrear a mão devido à oclusões de dedos. Estas oclusões ocorrem devido à complexidade das juntas da mão, dependendo da pose realizada. Por estes motivos, algoritmos de reconhecimento de gestos da mão utilizando sensores ópticos ainda não apresentam alto grau de acurácia.

Figura 1 – Ilustração de uso do sensor Leap Motion para rastreamento de movimentos das mãos.

Fonte – Community, 2016.

1.1 Objetivos

Neste trabalho abordamos o problema de reconhecimento de gestos dinâmicos da mão utilizando o sensor Leap Motion. Consideramos que um gesto dinâmico pode ser codificado como uma sequência de poses chave (*key poses* [6]), e buscamos desenvolver classificadores para poses estáticas da mão, os quais são incorporados em um classificador de gestos dinâmicos da mão.

Para o problema de reconhecimento de poses estáticas da mão, utilizamos aprendizado de máquina. Mais especificamente, nosso método procurará aprender métricas que melhorem o desempenho do classificador de *Distâncias Médias Mínimas* (DMM) proposto. Para isto, criamos conjuntos de treinamento de poses estáticas e passamos estes conjuntos como entrada para algoritmos de aprendizagem de métricas linear e não linear. As métricas calculadas são usadas no lugar da distância euclidiana padrão.

Já para o reconhecimento de gestos dinâmicos da mão, treinamos gestos dinâmicos usando uma abordagem probabilística baseada nos *Modelos Ocultos de Markov* (HMM), onde supomos que as transições entre poses estáticas de um gesto dependem unicamente da pose estática corrente do gesto, ou seja, não depende de um passado distante. Na etapa de treinamento de gestos dinâmicos, são realizados diversos exemplos de gestos de cada classe, a fim de generalizar todas as possíveis transições entre poses estáticas que podem ocorrer na classe. As informações de probabilidades de transições entre poses estáticas de cada classe de gestos são então codificadas em estruturas denominadas grafos de ação. Por fim, estes grafos são utilizados para possibilitar o reconhecimento de gestos dinâmicos da mão em tempo real e possivelmente *online*, ou seja, em tempo real sem a necessidade de pausas ou quaisquer tipos de segmentações extras, deixando o usuário livre para realizar os gestos da forma como bem entender.

1.2 Trabalhos Relacionados

De modo geral os trabalhos relacionados ao reconhecimento de gestos podem ter até duas etapas: reconhecimento de poses estáticas e reconhecimento de gestos dinâmicos. Sendo a primeira etapa geralmente um passo intermediário necessário para o reconhecimento de gestos dinâmicos realizado na segunda etapa. Isto ocorre pois muitos dos métodos de reconhecimento de gestos dinâmicos trabalham com o modelo de gestos sequencial

temporal de poses estáticas. Logo, o desenvolvimento de algoritmos de reconhecimento de poses estáticos torna-se indispensável.

A literatura de reconhecimento de gestos utilizando aprendizagem de métrica é escassa. Destacamos o trabalho de Berlemont[7], uma tese de doutorado desenvolvida no *Instituto Nacional de Ciências Aplicadas* (INSA) de Lyon, França. Uma das aplicações foi o desenvolvimento de um método para reconhecimento de gestos inerciais 3D relizados com smartphones. Para isto foi utilizada uma abordagem de aprendizagem de métrica não linear obtida através de um tipo de rede neural denominada *Rede Neural Siamesa* (SNN) [8], que é uma classe de rede neural popular entre tarefas que envolvem encontrar semelhança entre dois objetos comparáveis.

A literatura em reconhecimento de gestos é bastante diversa, abrangendo diferentes técnicas e com inúmeras aplicações. Focamos aqui nos mais relacionados ao tema de pesquisa deste trabalho, no entanto, recomendamos ao leitor [9, 10, 11, 12] para uma apresentação mais abrangente dos trabalhos relacionados.

Mais relacionado com nossa abordagem é o trabalho de Miranda et al.[6], que propõe um modelo de reconhecimento de gestos corporais utilizando o sensor Kinect como dispositivo de extração de recursos. Cada pose estática foi descrita com uma representação angular adaptada das articulações do esqueleto fornecido pelo sensor Kinect. Para detecção e classificações de poses estáticas, foi utilizado um classificador multi-classe derivado das *Máquinas de Vetores de Suporte* (SVM). Enquanto para o reconhecimento de gestos dinâmicos foi proposto um esquema de florestas de decisão, onde cada raíz da floresta é uma *key pose* do gesto, as folhas representam classes de gestos e caminhos da raíz às folhas representam sequências de *key poses* dos respectivos gestos das folhas. Com esta abordagem foi possível efetuar o reconhecimento de gestos dinâmicos corporais de forma *online* (em tempo real). No entanto, este método possui o problema de latência alta, uma vez que a identificação dos gestos só ocorre após o término da execução dos mesmos.

Para corrigir o problema de latência observado em [6], Vieira et al.[13] propôs a utilização de um método de reconhecimento de gestos mais adequado que possibilita o reconhecimento de ações humanas com baixa latência. Para isto, substituiu o método de florestas de decisão pela abordagem probabilística de grafos de ação [14]. Além disso, foram desenvolvidas outras melhorias, como por exemplo: a possibilidade de reconhecimento de

gestos que diferem apenas pela velocidade de execução.

1.2.1 Reconhecimento de Gestos da Mão

No que diz respeito ao reconhecimento de gestos da mão, diferentes abordagens surgiram fazendo uso de diferentes tecnologias, tais como: webcams [15], luvas coloridas [16], luvas de dados [17] e, mais recentemente, os sensores de profundidade (como o Kinect) [18] e de movimento (como o Leap Motion). Abaixo destacamos alguns trabalhos que utilizaram estas duas últimas tecnologias.

Em Marin et al.[19] é proposto um esquema de reconhecimento de gestos da mão que utiliza uma combinação de recursos extraídos de ambos os sensores Leap Motion e Kinect. Do Leap Motion foram utilizados recursos como informações de distâncias das pontas dos dedos ao centro da palma, ângulos de aberturas das pontas dos dedos com o centro da palma e elevação das pontas dos dedos. Enquanto do Kinect foram utilizados recursos como curvatura e correlação.

Em Hu et al.[20], o reconhecimento de gestos dinâmicos é feito através da análise de imagens estáticas de trajetória do gesto utilizando *Redes Neurais Convolucionais* (CNN). Cada imagem de trajetória utiliza como recursos as posições 3D das pontas dos dedos obtidos através do sensor Leap Motion. Com esta abordagem foram reconhecidos os dígitos de 0 a 9 “desenhados no ar” de forma segmentada, com acurácia de 98.8%.

Já Schmidt et al.[21] propõe uma abordagem de reconhecimento de gestos estáticos da mão com o sensor Leap Motion onde foi utilizado como recursos: os ângulos entre os dedos, ângulos entre os vetores direcionais dos dedos com o vetor normal da palma e a distância das pontas dos dedos para o centro da palma. Foram utilizados dois classificadores, o primeiro utilizando SVM e o segundo baseado em florestas aleatórias [22]. Em sequência foram comparados os desempenhos de ambos os métodos na tarefa de reconhecimento de poses de um conjunto de testes com classes de poses estáticos relativamente simples, isto é, classes de poses estáticas que não possuem grandes oclusões de juntas e dedos da mão.

Alguns outros trabalhos recentes que utilizam técnicas híbridas, bem como outras não abordadas nesta dissertação, são Lu et al.[23], Guo et al.[24] e John et al.[25].

Como podemos ver, a literatura em reconhecimento de gestos da mão com sensores ópticos ainda é um campo pouco explorado, com abordagens por vezes simplistas que

são insatisfatórias para o desenvolvimento de aplicações reais robustas, em tempo real e *online*. Em Cheng et al.[4] são discutidas algumas destas limitações, como por exemplo: os desafios encontrados para desenvolver aplicações robustas que realizem reconhecimento *online* de gestos dinâmicos da mão sem a necessidade de quaisquer tipos de pausas ou segmentações manuais por parte do usuário.

Como contribuição para a literatura, nosso método de reconhecimento *online* de gestos dinâmicos da mão, com o sensor Leap Motion, introduz no campo de reconhecimento de gestos manuais um método, inspirado no trabalho de Vieira et al.[13], capaz de reconhecer gestos dinâmicos da mão em tempo real com baixa latência (ou seja, de forma antecipada), ao mesmo tempo que deixa o usuário livre para executar os gestos da forma que bem entender. Com este trabalho contribuímos também com o desenvolvimento de uma literatura básica em língua portuguesa a respeito do problema de aprendizagem de métricas.

2 VISÃO GERAL

Tratamos neste trabalho do problema de reconhecimento de gestos estáticos (poses) e dinâmicos da mão usando o sensor Leap Motion. No entanto, para o reconhecimento de gestos dinâmicos, ou seja, movimentos da mão, também será necessário reconhecer poses.

Abordamos o problema de classificação e reconhecimento de poses da mão usando aprendizado de máquina. Mais especificamente, procuramos aprender métricas que melhorem o desempenho de um classificador simples de poses da mão. Essa classificação é então utilizada para possibilitar o reconhecimento *online* e *offline* de gestos da mão.

Para melhor esclarecimento, dividimos o método de reconhecimento de poses e gestos em quatro etapas: treinamento de poses; reconhecimento de poses; treinamento de gestos dinâmicos; e reconhecimento de gestos dinâmicos.

2.1 Treinamento de Poses da Mão com Aprendizagem de Métrica

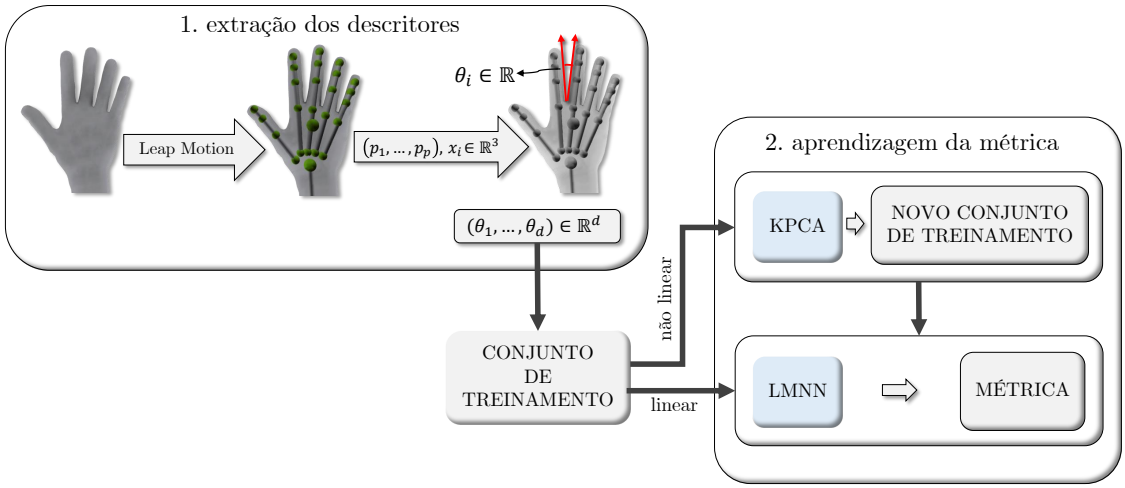
Aqui é abordada a aprendizagem de métricas que melhor se adaptem aos tipos de dados trabalhados. Inicialmente, é necessário criar um conjunto de treinamento para distintas classes de poses da mão, onde o usuário deve fornecer exemplos de cada uma das classes que deseje reconhecer em sua aplicação.

De acordo com a Figura 2, para cada exemplo dado, o sensor Leap Motion é utilizado para extrair informações relevantes da pose da mão, tais como a posição das juntas em relação ao sensor e o vetor normal da palma. Com estas informações, extraímos inicialmente características da mão que descrevem poses estáticas de forma invariante a rotação e translação. Posteriormente são consideradas outras características, como os vetores normal e direcional da mão (Figura 17), e o centro da palma (Figura 18), a fim de obter as variações de rotação e translação da mão que muitos gestos dinâmicos possuem.

Após a aquisição dos exemplos, o conjunto de treinamento obtido com informações supervisionadas (rótulos das classes) será usado para aprender uma métrica. Caso a métrica desejada seja linear, o algoritmo *Large Margin Nearest Neighbor* (LMNN) recebe como entrada o conjunto de treinamento e calcula uma matriz que representa uma métrica de Mahalanobis, que aproxima exemplos de uma mesma classe e distancia exemplos de classes distintas (Figura 2, quadro 2, abaixo). Caso a métrica desejada seja não linear, o conjunto

de treinamento é passado como entrada para o algoritmo *Kernel Principal Component Analysis* (KPCA), que mapeia os dados de maneira não linear e retorna um novo conjunto de treinamento em um espaço de alta dimensão (Figura 2, quadro 2, acima). Este novo conjunto de treinamento é então passado como entrada para o algoritmo LMNN, onde uma métrica pode ser aprendida mais eficientemente.

Figura 2 – Criação do conjunto de treinamento de poses da mão e aprendizagem da métrica linear (quadro 2, abaixo) ou não linear (quadro 2, acima).

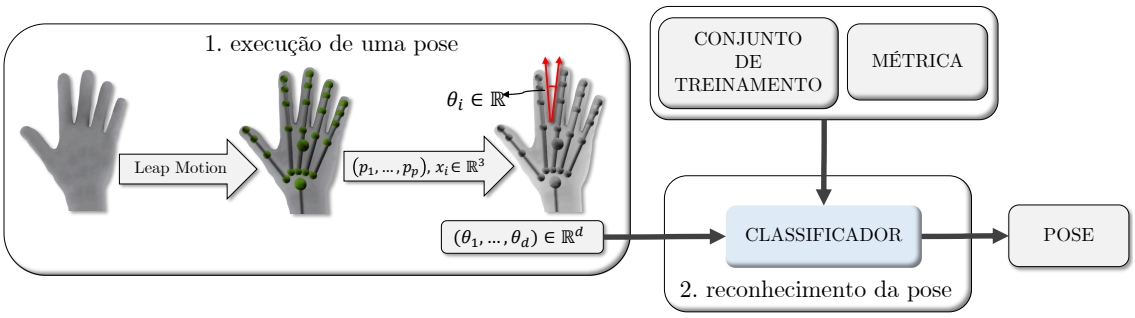


Fonte – Autor, 2017.

2.2 Reconhecimento de poses

Uma vez obtida a melhor métrica associada a um determinado conjunto de treinamento, vem a tarefa de reconhecimento de novas poses reali zadas pelo usuário em tempo real. Para tanto, utilizamos o sensor Leap Motion e realizamos a pose a ser reconhecida. Como na etapa anterior, o sensor extrai as características da pose executada e essas informações são passadas para o classificador de poses. O classificador leva em conta as distâncias entre a pose executada e representantes das classes de poses do conjunto de treinamento, segundo a métrica aprendida anteriormente, para decidir a que classe pertence a amostra. Um exemplo de um classificador deste tipo é o kNN. A Figura 3 ilustra este processo.

Figura 3 – Etapas do método usado no reconhecimento de poses.



Fonte – Autor, 2017.

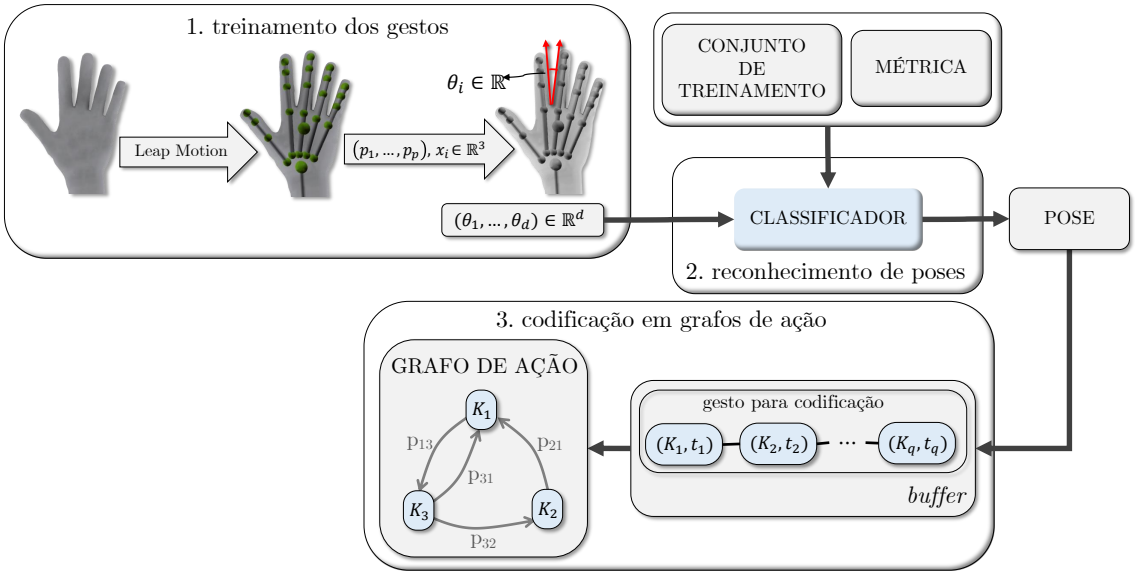
2.3 Treinamento de Gestos Dinâmicos da Mão usando Grafos de Ação

Nesta etapa é feito o treinamento de gestos dinâmicos. O usuário deve fornecer exemplos de cada classe de gestos dinâmicos que deseja que a máquina reconheça. Estes exemplos serão compostos por sequências de poses da mão que serão representadas pelos descritores de poses da mão. Utilizando o classificador de poses decodificamos essas sequências em sequências de *key poses*. Finalmente, utilizando uma abordagem chamada grafos de ação, é feita uma codificação das sequências de *key poses* dos gestos em matrizes de transições de probabilidades de *key poses*, baseadas em modelos de Markov. Desse modo, cada classe de gesto dinâmico corresponde a um modelo codificado como um grafo de ação. A Figura 4 ilustra este processo.

2.4 Reconhecimento de Gestos Dinâmicos

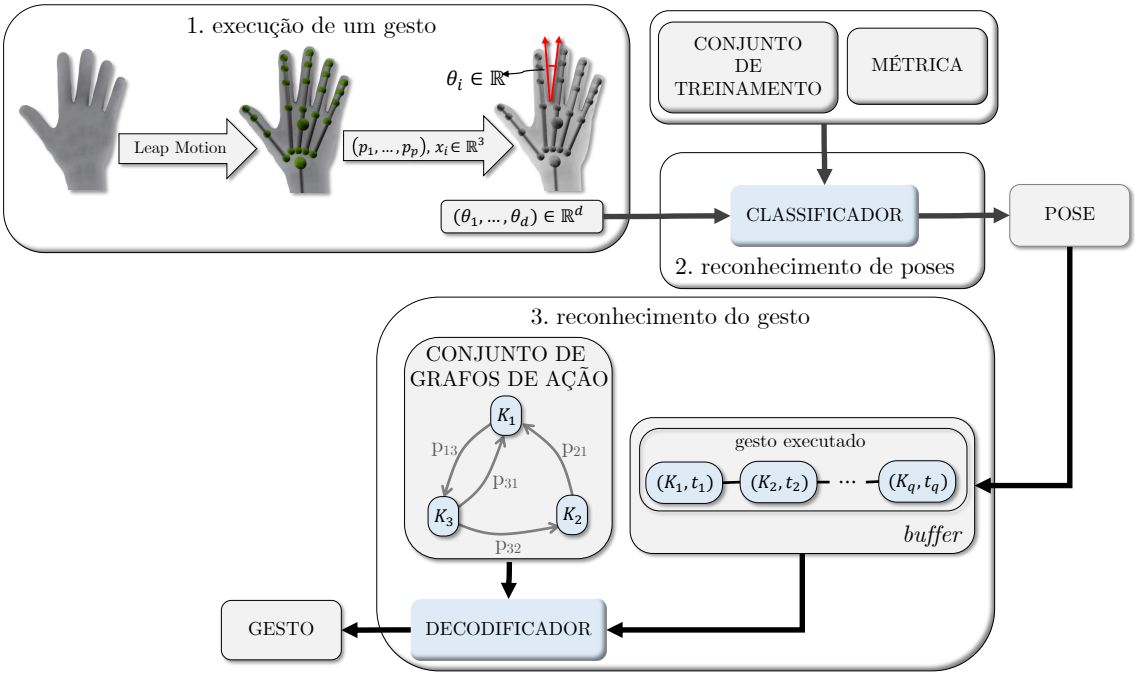
Nesta etapa o usuário poderá realizar gestos dinâmicos treinados previamente e a máquina deve ser capaz de reconhecer o gesto realizado, de forma *offline* ou *online* (em tempo real). Para decodificar estes gestos, será utilizado um decodificador baseado nos *Modelo Oculto de Markov* (HMM) para calcular a probabilidade de cada gesto realizado ter sido gerado a partir de cada modelo de classe de gesto treinado previamente. Estas probabilidades serão utilizadas para inferir o gesto realizado. No reconhecimento *online* este decodificador de gestos funciona em paralelo ao classificador de poses. A Figura 5 ilustra este processo.

Figura 4 – Etapas do treinamento dos gestos com a codificação em grafos de ação.



Fonte – Autor, 2017.

Figura 5 – Etapas do método de reconhecimento de gestos.



Fonte – Autor, 2017.

3 PRELIMINARES

3.1 Sensores de Profundidade

Sensores de profundidade são dispositivos capazes de capturar, em tempo real, nuvens de pontos 3D representando a geometria de superfícies que estão em seu campo de visão. Dessa forma, estes sensores podem ser utilizados em diversas aplicações, incluindo: reconhecimento de gestos para o controle de dispositivos de entretenimento e utilitários domésticos, tais como televisores e consoles de jogos; realidade virtual; captura de movimento para geração de modelos em animações, e; reconstrução de geometria 3D.

Mas para entendermos como esse mapeamento é feito é preciso dar uma descrição geral do funcionamento de tais sensores. Em geral, existem dois componentes que sempre estão presentes nos sensores de profundidade: projetores de luz infravermelha e câmeras infravermelhas. O projetor serve para lançar luz infravermelha (com ou sem um padrão definido) sobre as superfícies em seu campo de visão. Vale ressaltar que o ser humano não é capaz de ver a luz projetada porque ela tem comprimentos de onda do intervalo de luz não visível pelo olho humano, porém as câmeras destes sensores são capazes de capturar luz com estas características. Uma câmera infravermelha é essencialmente a mesma que uma câmera RGB regular, exceto que os pixels das imagens capturadas representam comprimentos de onda no intervalo de luz infravermelha. Quando a luz é projetada (com ou sem padrão), a câmera envia essas imagens para o processador do sensor, que as decodifica, retornando uma imagem de profundidade do campo de visão do sensor naquele dado instante. Matematicamente, uma imagem de profundidade é uma função $f : I \times J \to \mathbb{R}$, com $I, J \subset \mathbb{N}$, tal que para cada par de pontos (x, y) do retângulo $I \times J$, $f(x, y) = d$ representa a distância do sensor ao ponto correspondente na superfície mapeada.

Normalmente os fabricantes de sensores de profundidade, como o Kinect por exemplo [26], não disponibilizam qual padrão de luz o projetor de seus sensores utilizam, mas vale destacar que a qualidade desse padrão implica diretamente na qualidade da geometria capturada pelo sensor.

Outra classe de sensores, projeta luz sem utilizar padrões, se baseando em estereoscopia para reconstruir a geometria. Nestes sensores, é necessário capturar simultaneamente

duas imagens de posições distintas. Esta é a tecnologia utilizada, por exemplo, no sensor Leap Motion [27]. Na próxima seção descrevemos um pouco sobre seu o funcionamento.

3.2 Leap Motion

O sensor Leap Motion, criado pela empresa homônima e apresentado ao mercado em 2013, é uma tecnologia que consiste em um pequeno dispositivo dotado de sensores ópticos e luz infravermelha, que faz o reconhecimento e rastreamento de mãos e dedos do usuário em tempo real [27].

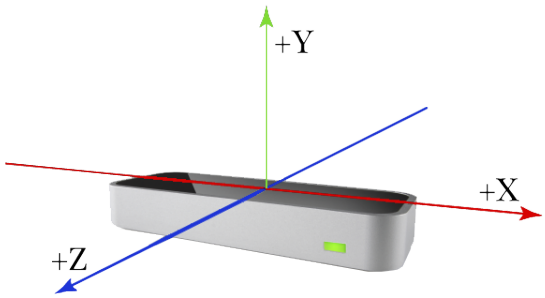
Este pequeno sensor pode ser posicionado em cima de uma mesa e, segundo o fabricante, os sensores ópticos incorporados no dispositivo têm um campo de visão de 150 graus com um alcance que varia de 25 a 600 milímetros acima do dispositivo.

No estudo de Weichert et al.[28] foi mostrado que a precisão global média do dispositivo é de 0,7 milímetros. Entretanto, assim como todos os sensores de profundidade disponíveis no mercado, o Leap Motion também sofre com problemas de oclusão do alvo a ser rastreado. Dessa forma, poderão ocorrer alguns erros grosseiros na hora do rastreamento de alguns dedos da mão que não estejam visíveis ao sensor. Para tentar contornar esse problema, o software Leap Motion combina os dados de seus sensores com um modelo interno da mão humana, para assim tentar prever posições de dedos em situações de rastreamento difíceis.

O Leap Motion utiliza um sistema de coordenadas interno, onde a origem é centrada no topo do dispositivo. O eixo Y está na posição vertical, com valores positivos aumentando para cima. O eixo X é paralelo a borda mais longa do dispositivo, com valores positivos crescentes para a direita. Por fim, o eixo Z tem valores positivos na direção do usuário, como mostra a Figura 6.

A API (*Application Programming Interface*) do Leap Motion fornece uma vasta quantidade de informações de quaisquer mãos detectadas pelo dispositivo, como por exemplo, vetor normal e direcional da palma, velocidade de deslocamento, posição das juntas dos dedos, etc. Todas essas informações são calculadas em tempo real e fornecidas para o usuário a cada quadro. Para isto, o Leap Motion adquire duas imagens infravermelhas (uma de cada câmera), e usando uma tecnologia não divulgada pela empresa, extrai o esquelo 3D da mão comparando as imagens 2D geradas por cada câmera. A taxa de

Figura 6 – Sistema de coordenadas Leap Motion.



Fonte – Motion, 2015.

atualização de quadros é bastande elevada, chegando a cerca de 200 quadros por segundo [3].

A API está estruturada hierarquicamente nas classes: *Frame, Hands, Arms, Fingers* e *Bones*. A classe *Frame* é essencialmente a raíz do modelo de dados do Leap Motion.

A classe *Hands* fornece informações sobre, por exemplo, a posição do braço a que a mão está ligada e as listas de dedos das mãos. Métodos como *Hand::palmNormal()* e *Hands::direction()* retornam vetores que definem a orientação da mão detectada.

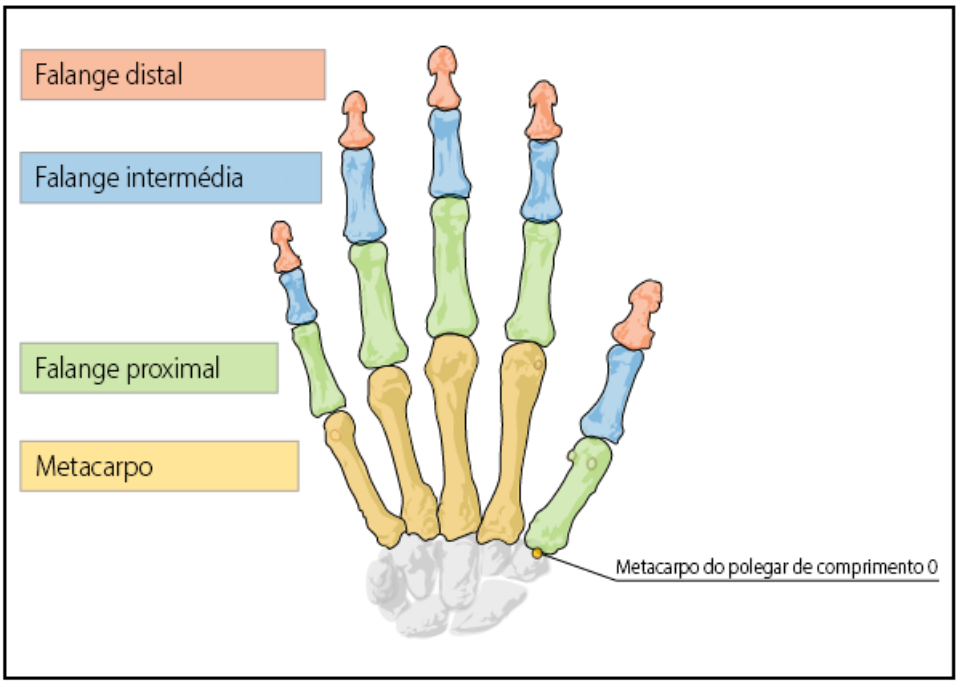
A classe *Arms* serve para fornecer a orientação, comprimento, largura e extremidades do braço. Quando o cotovelo não está em vista o sistema faz uma previsão baseada nas proporções médias do comprimento do braço humano.

Já a classe *Fingers* fornece informações sobre cada dedo. Dedos são identificados pelos tipos, ou seja, por polegar, indicador, médio, anular ou mindinho. Um objeto *Finger* fornece um objeto *Bone* que descreve a posição e orientação de cada osso do dedo. Para o Leap Motion, todos os dedos contêm 4 ossos ordenados da base a ponta, como mostra a Figura 7.

Os ossos são identificados como:

- Metacarpo - o osso interno da mão que liga o dedo ao pulso (exceto para o polegar).
- Falange proximal - o osso da base do dedo, ligado à palma.
- Falange intermédia - o osso do meio do dedo, entre a ponta e a base.
- Falange distal - o osso da extremidade do dedo.

Figura 7 – Modelo da mão para o sistema Leap Motion.



Fonte – adaptado de Motion, 2015.

Uma observação importante é que o modelo para o polegar não está totalmente de acordo com o sistema de nomenclatura anatômica padrão. Um polegar real tem um osso a menos que os outros dedos, mas para facilidades de programação, o sistema foi implementado com um osso metacárpio de comprimento zero, para que o polegar tenha o mesmo número de ossos nos mesmos índices que os outros dedos. Dessa forma, no sistema do Leap Motion, o osso metacarpo foi rotulado como falange proximal e o falange proximal real como falange intermédia.

Como mencionado anteriormente, o software Leap Motion utiliza um modelo interno da mão humana para fornecer um rastreamento previsível, mesmo quando os dedos de uma mão não estão totalmente visíveis. Dessa forma, movimentos sutis de dedos, que estão fora do campo de visão dos sensores do dipositivo, podem não ser rastreados corretamente.

A API do dispositivo é capaz de fornecer uma medida de confiabilidade do modelo da mão rastreado, assumindo valores entre 0 e 1, que indicam o quão bem os dados rastreados se encaixam no modelo interno da mão, sendo 0 quando os dados de rastreamento diferem totalmente do modelo interno; 1 quando os dados rastreados se encaixam totalmente no modelo; e valores intermediários para indicar oclusões parciais de partes da mão.

3.3 Aprendizagem de Máquina

Segundo Mohri et al.[29] a aprendizagem de máquina pode ser definida de forma geral como o ramo da Ciência da Computação que estuda métodos computacionais que usam experiência para aprender e fazer previsões sobre dados. A experiência nesse sentindo refere-se a informações passadas para a máquina, tais informações constituem o chamado conjunto de treinamento do problema.

A aprendizagem de máquina se preocupa em projetar algoritmos de previsão eficientes e precisos. Tais algoritmos operam construindo um modelo a partir de amostras de dados (conjunto de treinamento) a fim de fazer previsões ou decisões guiadas pelos dados ao invés de seguir instruções inflexíveis e estáticas pré-programadas. Desse modo, o tamanho e a qualidade dessa amostra de dados são cruciais para o sucesso das previsões e decisões feitas pela máquina.

Uma vez que a qualidade de um algoritmo de aprendizagem depende dos dados utilizados, a aprendizagem de máquina está intimamente relacionada com a análise de dados. De uma forma geral, as técnicas de aprendizagem de máquina são métodos de manipulação de dados que combinam conceitos fundamentais da informática com ideias da estatística, da probabilidade e da otimização matemática.

A aprendizagem de máquina é utilizada em uma variedade de tarefas computacionais onde programar algoritmos explícitos é inviável. Exemplos de aplicações incluem a classificação de documentos, detecção de spams, reconhecimento de voz, reconhecimento ótico de caracteres (OCR) e reconhecimento de gestos (objetivo de pesquisa deste trabalho); além de aplicações em motores de busca, como o do Google [30], e em sistemas de recomendação inteligentes, como o sistema de recomendação de filmes e séries da Netflix [31].

Os problemas de aprendizagem de máquina podem diferir de acordo com diferentes critérios, como por exemplo: o tipo de problema, a abordagem da solução e o cenário de aprendizagem. Nesse contexto, o tipo de problema refere-se, em geral, a saída do algoritmo de aprendizagem utilizado. A abordagem refere-se aos tipos de técnicas e algoritmos utilizados. Enquanto o cenário de aprendizagem refere-se, em geral, ao tipo de conjunto de treinamento do problema.

Entre os principais problemas de aprendizagem de máquina, podemos citar: classi-

ficação, ordenação, regressão e clusterização.

Nos problemas de classificação o objetivo é atribuir uma categoria a cada item, como por exemplo na classificação de documentos, onde podem ser atribuídas categorias como negócios, esportes, política ou tempo. O número de categorias nesse problema é muitas vezes relativamente pequeno, mas pode ser exremamente grande em problemas difíceis como no reconhecimento ótico de caracteres (OCR) ou reconhecimento de voz. Neste trabalho, tratamos do problema de classificação de poses e gestos da mão.

Por outro lado em ordenação o objetivo é ordenar itens de acordo com algum critério. A busca de sítios na internet, por exemplo, é um problema clássico de ordenação.

Já nos problemas de regressão o objetivo é prever um valor real para cada item. Exemplos de regressão incluem a previsão de valores de estoque ou variações de variáveis econômicas. Neste problema é dada uma penalidade para uma previsão incorreta baseada na diferença de magnitude dos valores verdadeiro e previsto, ao contrário do problema de classificação que não há uma noção de proximidade entre categorias.

Em clusterização o objetivo é particionar itens em regiões homogêneas. Esse tipo de problema aparece geralmente quando é preciso analisar conjuntos muito grandes de dados. Por exemplo, no contexto da análise de redes sociais, os algoritmos de clusterização tentam encontrar grupos de usuários específicos dentre grandes grupos de pessoas.

Quanto a abordagem da solução dos problemas de aprendizagem, podemos destacar: aprendizagem por similaridade e métrica, e aprendizagem por representação.

Na aprendizagem por similaridade e métrica são analisados pares de amostras consideradas similares e pares de amostras consideradas menos similares. A máquina precisa então aprender uma função de similaridade (ou uma função de distância métrica) que possa predizer se novas amostras são similares ou não. Neste trabalho, realizamos aprendizagem de métricas para avaliar distâncias entre poses da mão.

Já a aprendizagem por representação é utilizada quando se quer preservar a informação das amostras, mas transformando-a de forma que a torne útil. Frenquentemente aplicada como um passo de pré-processamento antes da máquina aprender de fato a fazer classificações ou previsões.

Podemos ainda classificar a aprendizagem de máquina quanto aos cenários de

aprendizagem. Esses cenários diferem nos tipos de dados de treinamento disponíveis, na ordem e no método pelo qual os dados de treinamento são recebidos pela máquina. Os tipos mais comuns de aprendizagem incluídos nessa classificação são: aprendizagem supervisionada, aprendizagem não supervisionada e aprendizagem semi-supervisionada.

Na aprendizagem supervisionada a máquina recebe um conjunto de amostras rotuladas com categorias e o objetivo é fazer a previsão de a qual categoria novas amostras pertencem. Este é o cenário de aprendizagem mais comum associado aos problemas de classificação, regressão e ordenação.

Já na aprendizagem não supervisionada a máquina recebe um conjunto de amostras sem rótulos de categorias e o objetivo é predizer a qual categoria novas amostras pertencem. Uma vez que, em geral, não há nenhuma amostra disponível rotulada nessa configuração, pode ser difícil avaliar quantitativamente o desempenho de uma máquina de aprendizagem não supervisionada. Exemplos de problemas que utilizam este cenário são clusterização e redução de dimensionalidade.

Por outro lado, na aprendizagem semi-supervisionada a máquina recebe um conjunto de amostras composto por dados rotulados e não rotulados, e assim como nos cenários anteriores, o objetivo é fazer a previsão de qual categoria novas amostras se encaixam. Esse tipo de aprendizado é comum em ambientes onde os dados sem rótulos são facilmente acessíveis, mas os rótulos são difíceis de serem obtidos. A análise das condições sob as quais isso pode realmente ser realizado é o tema de muitas pesquisas teóricas sobre aprendizagem de máquina. Exemplos de problemas que se enquadram nesse cenário de aprendizagem incluem problemas de classificação, regressão e ordenação.

3.3.1 Terminologia

Agora descreveremos a terminologia utilizada em problemas de aprendizagem de máquina. Para isto, utilizaremos o problema de reconhecimento de sinais em LIBRAS para ilustrar melhor cada conceito. O reconhecimento de sinais em LIBRAS consiste no problema de classificar a que classe de sinais treinadas uma nova pose pertence. Dito isto, vamos a terminologia básica:

- **Exemplos**: são as amostras de dados utilizados para treinamento e avaliação. No problema de reconhecimento de sinais, esses exemplos correspondem à coleção de po-

ses da mão, representando os sinais de LIBRAS, que será usada para a aprendizagem e teste.

- **Descritores**: é o conjunto de atributos mais relevantes, normalmente representado como um vetor, associado a um exemplo. No caso de poses da mão, alguns descritores relevantes podem incluir os ângulos de abertura entre os dedos, os ângulos dos vetores das articulações com o vetor normal da palma, a posição de algumas articulações, o centro da palma, e assim por diante.

- **Rótulos**: são os valores ou classes atribuídos a exemplos. Nos problemas de classificação, os exemplos recebem rótulos representando uma classe de exemplos específica, tais rótulos são normalmente representados com números naturais. No problema de reconhecimento de poses da mão, as poses poderiam receber rótulos de diferentes classes de acordo com as poses que o usuário deseje que a máquina reconheça. Por exemplo, no problema de reconhecimento de sinais em LIBRAS, os rótulos poderiam ser 0 para a letra A, 1 para a B, 2 para a C, e assim por diante. Já nos problemas de regressão, os exemplos são normalmente rotulados com valores reais.

- **Conjunto de treinamento**: são os exemplos usados para treinar um algoritmo de aprendizagem. No problema de reconhecimento de sinais, o conjunto de treinamento consiste em exemplos de sinais rotulados como $0, 1, 2, 3, \ldots, 25$, indicando respectivamente as classes de sinais $A, B, C, D, \ldots, Z$. No entanto, o conjunto de treinamento pode variar bastante para os diferentes cenários de aprendizagem vistos anteriormente.

- **Conjunto de validação**: são os exemplos usados para ajustar os parâmetros de um algoritmo de aprendizagem. Pois tais algoritmos normalmente têm um ou mais parâmetros livres, e o conjunto de validação é usado para selecionar valores apropriados para esses parâmetros do modelo. Normalmente necessário ao se trabalhar no cenário de aprendizagem supervisionado.

- **Conjunto de teste**: são os exemplos utilizados para avaliar o desempenho de um algoritmo de aprendizagem. Esse conjunto é separado dos conjuntos de treinamento e validação e não é disponibilizado à máquina na faze de aprendizagem. No problema de reconhecimento de sinais, o conjunto de teste consiste em uma coleção de exemplos de poses da mão representando sinais feitos pelo usuário, para os quais o algoritmo

de aprendizagem deve prever o rótulo com base nos seus descritores. Estas previsões são então comparadas com os rótulos verdadeiros afim de medir o desempenho do algoritmo.

- **Validação cruzada**: é uma técnica usada para calibrar parâmetros em algoritmos de aprendizagem de máquina, e assim melhorar o desempenho de previsão da máquina. Normalmente busca-se tais parâmetros ótimos pelo cruzamento de dados de exemplos do conjunto de validação e observa-se qual destes fornece o menor erro segundo alguma função de custo pré-estabelecida.

- **Espaço de características**: é o espaço onde estão os descritores do conjunto de treinamento. Normalmente um espaço vetorial $\mathbb{R}^d$ de dimensão d, onde d é quantidade de descritores dos exemplos do conjunto de treinamento.

Definidos estes conceitos, podemos agora descrever com maior precisão o modelo genérico de treinamento de uma máquina de aprendizagem. Pois em geral, algoritmos de aprendizagem seguem um roteiro de treinamento pré-estabelecido, sendo dividido em algumas fases.

Na primeira fase fazemos uma análise de quais descritores são mais relevantes para os exemplos do problema, isto é, quais atributos caracterizam melhor os exemplos. Essa fase inicial é de grande importância no desenvolvimento da solução do problema de aprendizagem, pois descritores úteis podem efetivamente guiar algoritmos de aprendizagem para serem robustos e precisos, enquanto que descritores mais pobres podem ser enganosos, podendo levar a efeitos desastrosos sobre o desempenho de previsão da máquina.

Na segunda fase são coletados os exemplos, que são particionados aleatoriamente nos conjuntos de treinamento, validação e teste. O tamanho de cada um desses conjuntos depende de várias considerações diferentes. Por exemplo, a quantidade de dados reservados para validação depende do número de parâmetros livres do algoritmo. Além disso, quando a quantidade de exemplos rotulados é relativamente pequena, o conjunto de treinamento é frequentemente escolhido para ser maior do que o conjunto de teste, uma vez que o desempenho de aprendizagem depende diretamente do conjunto de treinamento.

Na terceira fase são usados os conjuntos de treinamento e validação pra treinar o algoritmo de aprendizagem através da fixação de diferentes valores de seus parâmetros

livres. Para cada valor desses parâmetros, o algoritmo seleciona uma hipótese diferente, escolhendo dentre elas a hipótese que resulta no melhor desempenho no conjunto de validação.

Por fim, temos a fase de teste, que tem como objetivo fazer a análise do desempenho do algoritmo aprendido. Aqui a máquina tenta prever os rótulos dos exemplos do conjunto de teste. O desempenho do algoritmo é analisado usando a função de perda associada ao problema. Por exemplo, no problema de detecção de spams, a função de perda *zero-um* pode ser utilizada para comparar os rótulos previstos pelo algoritmo e os verdadeiros.

Na próxima seção, descreveremos o funcionamento geral do algoritmo de aprendizagem de métricas LMNN, que será a base do método de aprendizagem proposto para resolver o problema de reconhecimento de poses e gestos da mão estudado neste trabalho.

3.4 Aprendizagem de Métricas

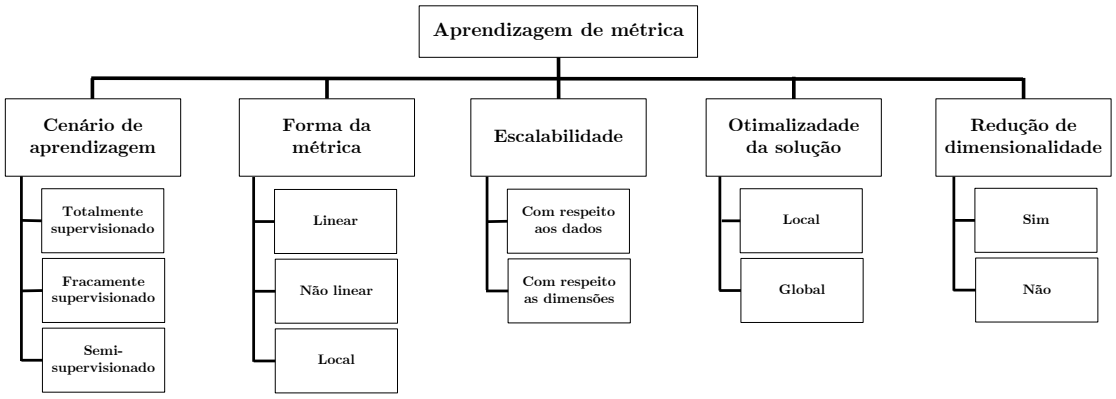
Nesta seção falaremos de um tipo especial de aprendizagem de máquina: a aprendizagem de métricas. Este problema de aprendizagem está relacionado com a aprendizagem de uma função de distância ótimo para um problema específico; isto torna-se necessário pelo fato de muitas vezes o problema de aprendizagem tratado exigir que certos atributos tenham mais relevância que outros para serem corretamente classificados, ou para resolver problemas de ordenação de instâncias por similaridade.

Os métodos e algoritmos de aprendizagem de métricas podem diferir de acordo com muitas características, como por exemplo: cenário de aprendizagem, tipo de métrica, escalabilidade, otimalidade da solução e redução de dimensionalidade. O diagrama da Figura 8 mostra as principais características de algoritmos de aprendizagem de métricas.

Na característica de cenário de aprendizagem daremos enfoque a formulação totalmente supervisionada. No que diz respeito a forma da métrica, detalharemos nas próximas subseções os casos: linear e não linear. Sobre a otimalidade da solução, focaremos no modelo de aprendizagem de métrica global. Quanto as demais características, os detalhes podem ser encontrados no estudo de Bellet et al.[32].

Porém, antes de prosseguirmos com a teoria básica de aprendizagem de métrica, precisamos definir precisamente alguns conceitos matemáticos que serão utilizados com frequência de agora em diante.

Figura 8 – Cinco principais características de algoritmos de aprendizagem de métricas.



Fonte – adaptado de Bellet et al., 2013.

3.4.1 Definições Matemáticas

Definição 3.1. Uma métrica num conjunto X é uma função $d : X \times X \to \mathbb{R}$, que associa a cada par ordenado de elementos $x, y \in X$, um número real $d(x, y)$, chamado distância de x a y, de modo que sejam satisfeitas as seguintes condições para quaisquer $x, y \in X$:

(a) $d(x, x) = 0$;

(b) Se $x \neq y$ então $d(x, x) > 0$;

(c) $d(x, y) = d(y, x)$;

(d) $d(x, z) \leq d(x, y) + d(y, z)$;

As condições (a) e (b) dizem que $d(x, y) \geq 0$ para todo $x, y \in X$, e que $d(x, y) = 0$ se, e somente se, $x = y$. A condição (c) afirma que a distância $d(x, y)$ é uma função simétrica das variáveis x e y. Já (d) diz que uma métrica satisfaz a desigualdade triangular. Quando uma função satisfaz todas as condições de uma métrica, exceto a condição (a), chamamos essa função de *pseudométrica*. O par (X, d), onde X é um conjunto e d é uma métrica em X, é chamado de *espaço métrico*.

Definição 3.2. Um conjunto $\mathcal{K}$ é chamado de cone se para todo $x \in \mathcal{K}$ tem-se ainda que $\alpha x \in \mathcal{K}$ para todo $\alpha \geq 0$.

Definição 3.3. Um cone $\mathcal{K}$ é dito convexo se para todos $x, y \in \mathcal{K}$ tem-se que $\alpha_1 x + \alpha_2 y \in \mathcal{K}$ para todos $\alpha_1, \alpha_2 \geq 0$; isto é, se combinações lineares de elementos de $\mathcal{K}$ ainda estão em $\mathcal{K}$.

Definição 3.4. Uma matriz simétrica $M \in S\mathbb{R}^{n\times n}$ é chamada de positiva semidefinida se $x^T M x \geq 0$ para todo $x \in \mathbb{R}^n$ não nulo.

O conjunto das matrizes positivas semidefinidas será representado como $\mathcal{S}_+^n$ e usaremos a notação $M \succeq 0$ para dizer que $M \in \mathcal{S}_+^n$, isto é, para afirmar que M é positiva semidefinida.

Teorema 3.4.1 (**Caracterização das matrizes positivas semidefinidas**). Seja $M \succeq 0$. Então, valem as seguintes afirmações:

(a) Todos os autovalores de M são não negativos.

(b) Existe uma matriz G tal que $M = G^T G$.

Proposição 3.4.1. O conjunto das matrizes positivas semidefinidas $\mathcal{S}_+^n$ é um cone convexo.

Demonstração. Primeiramente, temos que provar que $\mathcal{S}_+^n$ é de fato um cone. Mas isto é simples, basta observar que se $M \in \mathcal{S}_+^n$, então por definição $x^T M x \geq 0$, logo para todo $\alpha \geq 0$ temos que $x^T(\alpha M)x = \alpha(x^T M x) \geq 0$, o que implica que $\alpha M \in \mathcal{S}_+^n$. Para provar a segunda parte, isto é, que $\mathcal{S}_+^n$ é convexo, basta observar que se $M, N \in \mathcal{S}_+^n$, então para todos $\alpha, \beta \geq 0$ temos que $x^T(\alpha M + \beta N)x = \alpha(x^T M x) + \beta(x^T N x) \geq 0$, o que implica que $\alpha M + \beta N \in \mathcal{S}_+^n$ e portanto $\mathcal{S}_+^n$ é um cone convexo. □

Definição 3.5. Um *kernel* positivo semi-definido $k(\cdot, \cdot)$, em um conjunto não vazio $\mathcal{X}$, é uma função $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ tal que para todo $n \in \mathbb{N}$ e todo $x_1, \ldots, x_n \in \mathcal{X}$ a matriz $K = (k_{ij})_{n\times n}$, $k_{ij} = k(x_i, x_j)$, é positiva semidefinida.

3.4.2 Formulação Inicial

Descreveremos agora uma formulação inicial que servirá de base para os modelos de aprendizagem de métrica, tanto linear quanto não linear estudados neste trabalho.

Segundo Kulis[33] é proposta a seguinte formulação: dada uma função de distância de entrada $d(x, y)$ entre elementos x e y (a distância euclidiana, por exemplo), juntamente

com informações supervisionadas, isto é, com dados de treinamento rotulados, construa uma nova função de distância $\tilde{d}(x, y)$ que é "melhor" do que a função de distância original. Essa função é geralmente do tipo $\tilde{d}(x, y) = d(f(x), f(y))$ para alguma função f, ou seja, a máquina deve aprender algum mapeamento f e utilizar a função de distância original sobre os dados mapeados por f.

Esta formulação pode ser classificada como global, uma vez que é aprendido um único mapeamento f para ser aplicado a todos os dados. Outras formulações com métodos de aprendizagem de métricas locais podem ser encontradas nos trabalhos de Wang et al.[34] e Noh et al.[35].

Dividimos a formulação de aprendizagem de métrica global, com respeito ao mapeamento f, em duas subclasses: linear e não linear. Em ambos os casos, tomamos como distância de entrada a distância Euclidiana, isto é, $d(x, y) = \|x - y\|$.

No caso linear, a função f será uma transformação linear, logo poderá ser representada como uma matriz G, donde a distância aprendida será da forma $\tilde{d}(x, y) = \|Gx - Gy\|$. Mostraremos detalhadamente este caso na próxima seção.

No caso não linear, uma das técnicas mais eficazes para aprender tais mapeamentos é transformar o problema em linear utilizando uma técnica chamada *truque do kernel*. A ideia básica para este caso é aprender um mapeamento linear em um novo espaço de características induzido por uma função não linear ϕ, donde a distância aprendida terá a forma $\tilde{d}(x, y) = \|G\phi(x) - G\phi(y)\|$. Explicaremos detalhadamente este procedimento na Seção 3.4.5.

3.4.3 Método Linear

Introduzida pelo matemático indiano Mahalanobis[36] em 1936, a distância de Mahalanobis refere-se originalmente a uma medida que se baseia nas correlações entre variáveis com as quais diferentes padrões podem ser identificados e analisados. Formulada originalmente como

$$d_{maha}(x, x') = \sqrt{(x - x')^T \Omega^{-1} (x - x')}, \tag{3.1}$$

onde x e x' são vetores aleatórios da mesma distrituição de probabilidade com matriz de covariância Ω.

No entanto, por um abuso de terminologia comum na literatura de aprendizagem

de métrica, vamos o usar o termo *distância de Mahalanobis* entre vetores x e y para se referir a distância quadrática

$$d_M(x,y) = \sqrt{(x-y)^T M(x-y)}, \tag{3.2}$$

onde M é uma matriz positiva semidefinida. Observe que quando M é a matriz identidade, a equação acima se torna a distância Euclidiana.

Como M é positiva semidefinida, pelo teorema 3.4.1 podemos escrevê-la como $M = G^T G$ para alguma matriz G. Substituindo em 3.2 teremos

$$\begin{aligned} d_M(x,y) &= \sqrt{(x-y)^T G^T G(x-y)} \\ &= \sqrt{(G(x-y))^T G(x-y)} \\ &= \sqrt{(Gx-Gy)^T (Gx-Gy)}, \end{aligned}$$

ou seja, $d_M(x,y) = \|Gx - Gy\|$.

Assim, uma distância de Mahalanobis como formulada acima, corresponde implicitamente ao cálculo da distância Eulidiana após a projeção linear dos dados pela transformação linear definida pela matriz G, o que coincide exatamene com a nossa formulação inicial de aprender uma transformação linear global para ser aplicada aos dados.

Vale lembrar que, computacionalmente, é mais vantajoso usar o quadrado da distância do que apenas a distância Euclidiana ou de Mahalanobis comum. Dessa forma, a partir de agora, quando nos referirmos a distância Euclidiana ou de Mahalanobis entre dois vetores x e y estaremos nos referindo ao quadrado das suas distâncias, isto é, à

$$d(x,y) = (x-y)^T (x-y) \tag{3.3}$$

e

$$d_M(x,y) = (x-y)^T M(x-y) \tag{3.4}$$

De modo geral, as técnicas de aprendizagem de métrica de Mahalanobis são formuladas como um problema de otimização convexa [37]; onde o objetivos é minimizar uma função de custo sujeita a restrições.

A seguir, descreveremos um método que tenta encontrar uma distância de Mahalanobis ótima a partir de dados de treinamento chamado *Large Margin Nearest Neighbor* (LMNN), desenvolvido por Weinberger e Saul[38].

3.4.4 Large Margin Nearest Neighbor (LMNN)

O algoritmo LMNN foi originalmente proposto para melhorar o desempenho do algoritmo de classificação kNN através da aprendizagem de uma distância melhor que a distância Euclidiana [38]. No entanto, nos limitaremos por enquanto ao uso do LMNN apenas para a aprendizagem da distância de Mahalanobis, deixando o método de classificação para ser discutido na Seção 4.3.

O objetivo principal por trás do LMNN é aprender uma pseudométrica sob a qual todos os exemplos no conjunto de treinamento estejam mais próximos de pelo menos k exemplos que compartilham o mesmo rótulo de classe, ao mesmo tempo que exemplos que compartilham rótulos diferentes sejam afastados por uma distância superior.

Primeiramente, denotemos nosso conjunto de treinamento supervisionado como $\mathcal{T} = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subset \mathbb{R}^d \times \mathcal{C}$, onde $\mathbb{R}^d$ é o espaço de características de dimensão d e $\mathcal{C} = \{0, 1, \ldots, c\}$ é o conjunto de rótulos de classes. Com essa notação cada par ordenado (x_i, y_i) representa o i-ésimo exemplo do conjunto de treinamento, onde x_i é vetor de descritores e y_i é o rótulo de classe exemplo.

Consideremos agora os conjuntos de pares ordenados $\mathcal{S} = \{(i, j);\ y_i = y_j\}$ e $\mathcal{D} = \{(i, j); y_i \neq y_j\}$ com $i, j = 1, \ldots, n$, onde n é a cardinalidade do cojunto de treinamento $\mathcal{T}$. Note que $\mathcal{S}$ contém todos os pares de índices de exemplos que compartilham o mesmo rótulo de classe, já $\mathcal{D}$ contém todos os pares de índices de exemplos que têm rótulos diferentes.

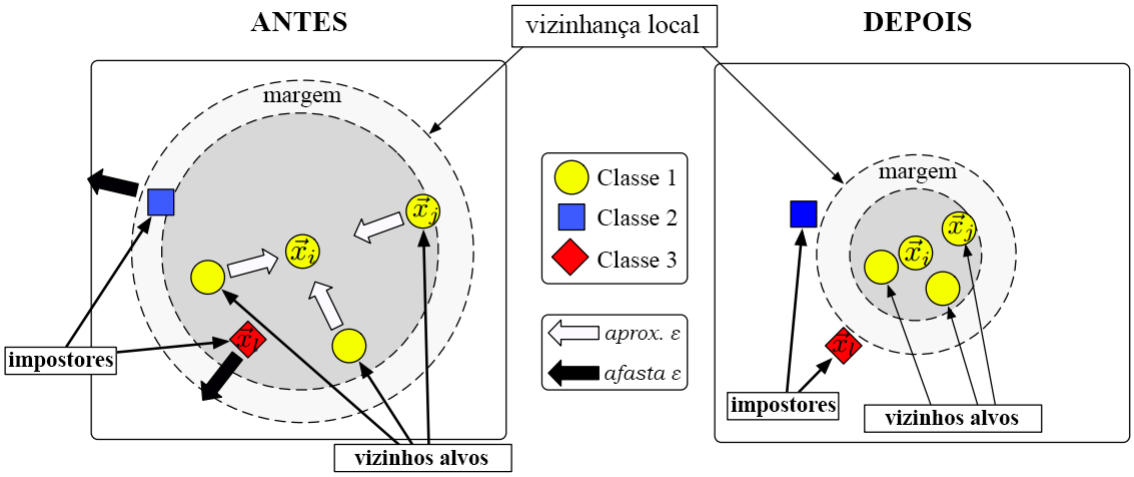
Considere ainda o conjunto de triplas ordenadas $\mathcal{R} = \{(i, j, k);\ y_i = y_j \neq y_k,\ d_M(x_i, x_j) < d_M(x_i, x_k)\}$ com $i, j, k = 1, \ldots, n$. Observe que $\mathcal{R}$ é definido para conter triplas de índices de dois exemplos da mesma classe e um terceiro de outra classe, que possuem a característica de distância relativa satisfeita pela desigualdade $d_M(x_i, x_j) < d_M(x_i, x_k)$, onde d_M é a distância de Mahalanobis definida por 3.4.

Denotemos ainda por *vizinho alvo* de um exemplo x_i os exemplos pertencentes a mesma classe de x_i e que desejamos que virem vizinhos mais próximos de x_i segundo a distância aprendida d_M. E por *impostor* aqueles exemplos que são vizinhos mais próximos de x_i mas que pertencem a uma classe diferente da de x_i.

O algoritmo LMNN tentará aprender uma métrica de Mahalanobis d_M que faz com que pelo menos k vizinhos alvos de um exemplo x_i se tornem de fato vizinhos mais

próximos, ao mesmo tempo que exemplos impostores sejam afastados de x_i por uma margem de distância relativamente grande. A Figura 9 ilustra o efeito provocado pela distância de Mahalanobis buscada pelo algoritmo LMNN.

Figura 9 – Ilustração esquemática da vizinhança de um exemplo antes da otimização da métrica d_M (à esquerda) e após a otimização (à direita). A métrica é otimizada de modo que os $k = 3$ vizinhos alvos de x_i fiquem dentro de uma bola de raio menor após o treinamento; ao mesmo tempo que exemplos de classes diferentes (impostores) fiquem fora desta bola com uma margem relativamente grande.



Fonte – adaptado de Weinberger e Saul, 2009.

Para isto, o algoritmo LMNN busca minimizar a função de custo

$$\mathcal{L}(M) = \sum_{(i,j)\in\mathcal{S}} d_M(x_i, y_j) + \lambda \sum_{(i,j,k)\in\mathcal{R}} [1 + d_M(x_i, x_j) - d_M(x_i, x_k)]_+ \tag{3.5}$$

sujeita a condição de a matriz M ser positiva semidefinida ($M \succeq 0$).

A minimização dessa função implica na minimização dos dois somatórios. No primeiro somatório busca-se minimizar as distâncias entre exemplos da mesma classe, o que faz com que vizinhos alvos de um exemplo se tornem de fato vizinhos mais próximos. Já no segundo somatório busca-se minimizar cada termo $[1 + d_M(x_i, x_j) - d_M(x_i, x_k)]_+ = max(1 + d_M(x_i, x_j) - d_M(x_i, x_k), 0)$, o que faz com que exemplos impostores x_k sejam distanciados da classe de exempos de rótulo y_i por pelo menos 1 unidade de distância. A constante $\lambda \in \mathbb{R}$ serve para alterar o peso do segundo somatório, isto é, para dizermos ao algoritmo se queremos forçar mais ou menos o distanciamento de exemplos impostores.

Como o LMNN busca minimizar a função $\mathcal{L}(M)$ sujeita a condição $M \succeq 0$, temos que o LMNN pode ser reformulado como um problema de programação semidefinida, uma classe particular de problemas otimização convexa, da seguinte forma:

$$\begin{aligned} \underset{M}{\text{minimizar}} \quad & \sum_{(i,j)\in\mathcal{S}} d_M(x_i, y_j) + \sum_{(i,j,k)\in\mathcal{R}} \xi_{ijk} \\ \text{sujeito a} \quad & d_M(x_i, x_j) + 1 \leq d_M(x_i, x_k) + \xi_{ijk} \\ & \xi_{ijk} \geq 0 \\ & M \succeq 0, \end{aligned} \tag{3.6}$$

onde a variável de folga ξ_{jk} absorve a restrição colocada sobre os exemplos impostores e a soma total de tais variáveis deve ser minimizada. A última condição assegura que M é positiva semidefinida.

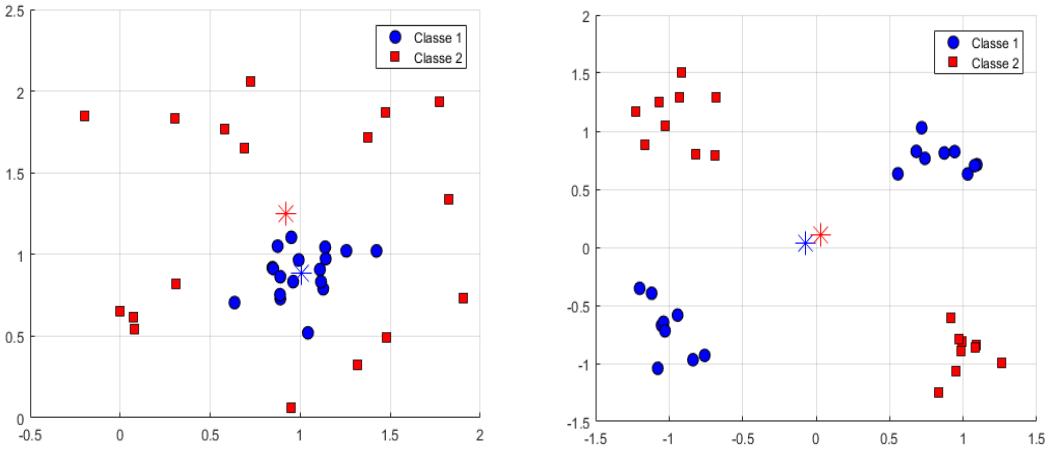
Embora problemas de programação semidefinida tendam a sofrer de alta complexidade computacional, este problema particular pode ser resolvido muito eficientemente porque, na maioria dos casos, as restrições já são satisfeitas naturalmente e não precisam ser aplicadas em tempo de execução. Para mais detalhes sobre o método de otimização, ver [38]. Vale a pena mencionar que o próprio autor do método disponibiliza uma implementação eficiente em Matlab, disponível em [39].

3.4.5 Método Não Linear

Como vimos na seção anterior, a aprendizagem de uma distância de Mahalanobis ótima pode ser encontrada na maioria das vezes através da formulação do problema como um problema de otimização convexa. No entanto, visto que esta distância é uma transformação linear, nem sempre será possível satisfazer as restrições do problema de minimização dado pela Equação 3.6 para um conjunto de treinamento qualquer. Pois, uma vez que transformações lineares se limitam a escalas, rotações e cisalhamentos dos dados, muitas vezes tais operações não são capazes de lidar com certos conjuntos de dados não separáveis linearmente. Para ilustrar melhor este fato, observe a Figura 10.

Para trabalhar com casos desse tipo é melhor aprender uma transformação não linear. No entanto, aprender tais transformações é uma tarefa difícil, pois ao contrário das transformações lineares que podem ser expressas como matrizes de parâmetros, o conjunto de transformações não lineares não é prontamente parametrizado. Para aprender

Figura 10 – Em (a) é ilustrado um caso onde os exemplos da classe 1 cercam os exemplos da classe 2, impossibilitando a aprendizagem de uma métrica linear que diferencie bem as duas classes. Já em (b) temos um caso onde os exemplos das classe 1 e 2 são compostos por dois aglomerados diametralmente opostos, impossibilitando a aprendizagem de uma métrica linear que aproxime exemplos da mesma classe e afaste de classes distintas.



(a) (b)

Fonte – Autor, 2017.

uma transformação do tipo, retringiremos a forma do mapeamento não linear a uma classe particular de transformações não lineares de tal forma que os parâmetros possam ser aprendidos eficientemente. A esta classe daremos o nome de transformações lineares *kernelizadas*.

Abaixo mostraremos dois métodos de *kernelização*, o método padrão e um método que utiliza *Análise de Componentes Principais* (PCA).

Truque do *Kernel* Padrão

Dado um *kernel* positivo semi-definido $k(x, y) = \phi(x)^T\phi(x)$, onde $\phi : \mathbb{R}^d \to \mathcal{H}$ é não linear, com $\mathcal{H}$ um espaço de Hilbert de dimensão possivelmente infinito; sejam $\phi = \phi(x)$, $\phi' = \phi(x)'$ e $\phi_i = \phi(x_i)$ exemplos mapeados em um novo espaço de características $\mathcal{H}$. A distância de Mahalanobis sobre $M \succeq 0$ entre dois exemplos ϕ_i e ϕ_j é

$$d_M(\phi_i, \phi_j) = (\phi_i - \phi_j)^T M(\phi_i - \phi_j) = (\phi_i - \phi_j)^T G^T G(\phi_i - \phi_j). \tag{3.7}$$

Para fixar as ideias, suponha que $G^T = (a_1, \ldots, a_d)$, onde cada a_i é um vetor coluna, e seja $\Phi = (\phi_1, \ldots, \phi_n)$ a matriz dos exemplos do conjunto de treinamento mapeados no

espaço $\mathcal{H}$. A ideia principal do truque do *kernel* é utilizar a seguinte parametrização

$$G^T = \Phi U^T \tag{3.8}$$

onde $U^T = (u_1, \ldots, u_d)$, com cada u_i sendo uma vetor coluna de dimensão n. Substituindo $G^T = \Phi U^T$ e $G = U\Phi^T$ na Equação 3.7 temos que

$$\begin{aligned} d_M(\phi_i, \phi_j) &= (\phi_i - \phi_j)^T \Phi U^T U \Phi^T (\phi_i - \phi_j) \\ &= (\Phi^T(\phi_i - \phi_j))^T U^T U \Phi^T (\phi_i - \phi_j) \\ &= (\Phi^T\phi_i - \Phi^T\phi_j)^T U^T U (\Phi^T\phi_i - \Phi^T\phi_j), \end{aligned}$$

ou seja, $d_M(\phi_i, \phi_j) = (k_i - k_j)U^T U(k_i - k_j)$, onde $k_i = \Phi^T\phi_i = (\langle\phi_1, \phi_i\rangle, \ldots, \langle\phi_n, \phi_i\rangle)^T$. Nossa fórmula agora depende apenas de produtos internos $\langle\phi_i, \phi_j\rangle$, e podemos, portanto, fazer a substituição $k_{ij} = \langle\phi_i, \phi_j\rangle$, exigindo que a matriz de *kernels* $K = (k_{ij})$ seja positiva semidefinida.

Portanto, o problema de encontrar a melhor distância de Mahalanobis no espaço de características inicial é agora reduzido a encontrar a melhor transformação linear U. No entanto, muitas vezes encontar U é muito mais problemático do que encontrar G no espaço inicial como é mostrado no estudo de Chatpatanasiri et al.[40].

Uma vez encontrada a matriz U, a distância de Mahalanobis entre um novo exemplo de teste x' e um exemplo x_i do conjunto de treinamento no novo espaço $\mathcal{H}$ pode ser calculado como

$$d_M(\phi_i, \phi_j) = (k' - k_i)^T U^T U (k' - k_i), \tag{3.9}$$

onde $k' = (k(x', x_1), \ldots, k(x', x_n))^T$.

Kernel Principal Component Analysis (KPCA)

Como enfatizamos acima, embora o truque do *kernel* padrão possa ser aplicado para resolver o problema da aprendizagem de métrica não linear, muitas vezes encontrar U torna-se muito mais difícil do que encontrar G no espaço de características inicial. Para resolver este problema, desenvolveu-se um método chamado *Kernel Principal Component Analysis* (KPCA) que pode ser aplicado eficientemente para a aprendizagem de uma métrica não linear.

Porém, antes de explicarmos a formulação do KPCA, vale a pena comentar a formulação linear da *Análise de Componentes Principais* (PCA) [41]. Dado um conjunto de

n pontos $x_k \in \mathbb{R}^d$ centrados na origem ($\sum_{i=1}^{n} x_i = 0$), o objetivo do PCA é a diagonalização da matriz de covariância

$$C = \frac{1}{n} \sum_{k=1}^{n} x_k x_k^T. \tag{3.10}$$

Para isto, é preciso resolver a equação de autovalores

$$\lambda v = Cv \tag{3.11}$$

para autovalores $\lambda \geq 0$ e autovetores $v \in \mathbb{R}^d \setminus \{0\}$. É possível mostrar que a matriz de covariância sempre é positiva semidefinida, portanto a Equação 3.11 sempre tem solução com $\lambda \geq 0$. Como

$$\begin{aligned} \lambda v &= Cv \\ &= (\tfrac{1}{n} \sum_{k=1}^{n} x_k x_k^T) v \\ &= \tfrac{1}{n} \sum_{k=1}^{n} x_k (x_k^T v) \\ &= \tfrac{1}{n} \sum_{k=1}^{n} x_k \langle v, x_k \rangle \end{aligned}$$

então

$$v = \sum_{k=1}^{n} \frac{1}{\lambda n} \langle x_k, v \rangle x_k. \tag{3.12}$$

Portanto, os autovetores v pertecem ao subespaço dos pontos x_k, $k = 1, \ldots, n$.

Ordenando os autovalores em ordem crescente é possível mostrar que as direções dos autovetores, correspondentes aos autovalores de maior valor, correspondem às direções ortogonais nas quais o conjunto $\{x_k\}_{k=1}^{n}$ tem maiores variâncias nos dados. A Figura 11 ilustra as direções principais de um conjunto de pontos no plano.

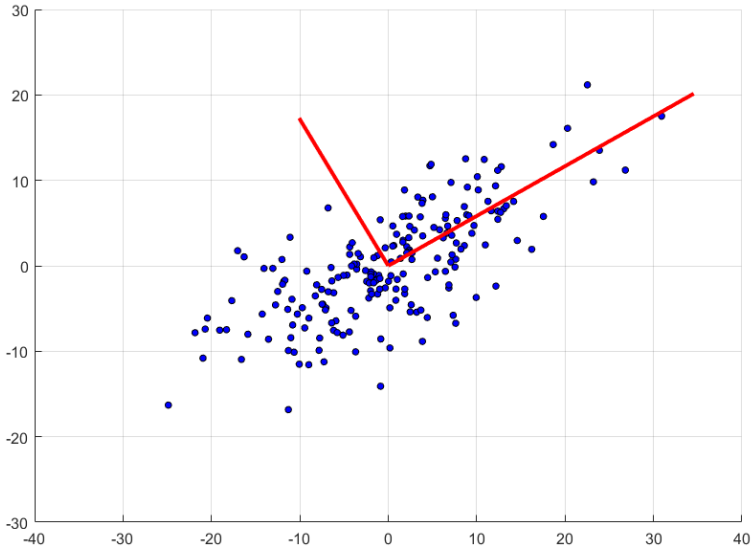
O KPCA nada mais é do que uma formulação não linear do PCA tradicional. A motivação inicial surge pela busca de componentes principais em um espaço de característica de dimensão mais alta.

Considere então um mapeamento não linear $\phi : \mathbb{R}^d \to \mathcal{H}$ do conjunto de características inicial $\mathbb{R}^d$ em um espaço de características $\mathcal{H}$ de dimensão mais alta, e sejam $k(\cdot, \cdot)$, $\phi = \phi(x)$, $\phi' = \phi(x)'$ e $\phi_i = \phi(x_i)$ como definidos na seção anterior.

Motivados pela Equação 3.10, definimos a matriz de covariância dos exemplos ϕ_k, mapeados por ϕ, como

$$\widetilde{C} = \frac{1}{n} \sum_{k=1}^{n} \phi_k \phi_k^T, \tag{3.13}$$

Figura 11 – Direções principais (em vermelho) de um conjunto de pontos no plano obtidas através do PCA.



Fonte – Autor, 2017.

e procuramos por autovalores $\lambda \geq 0$ e autovetores $\psi \in \mathcal{H} \setminus \{0\}$ resolvendo a equação de autovalores

$$\lambda\psi = \widetilde{C}\psi. \tag{3.14}$$

Assim como anteriormente, podemos mostrar que cada autovetor ψ pertence ao subespaço dos pontos ϕ_j, $j = 1, \ldots, n$, e que existem coeficientes α_j, $j = 1, \ldots, n$, tais que

$$\psi = \sum_{j=1}^{n} \alpha_j \phi_j \tag{3.15}$$

Dessa forma, faz sentindo tomar produtos internos por ϕ_i em ambos os lados da Equação 3.11. Fazendo isto, obtemos o sistema de equações

$$\lambda\langle\phi_i, \psi\rangle = \langle\phi_i, \widetilde{C}\psi\rangle, \ i = 1, \ldots, n. \tag{3.16}$$

Substituindo 3.13 e 3.15 no sistema acima, e considerando as entradas $k_{ij} := \langle\phi_i, \phi_j\rangle$ da matriz de *kernels* K, temos que

$$\begin{aligned}
\lambda\langle\phi_i, \sum_{j=1}^{n} \alpha_j\phi_j\rangle &= \langle\phi_i, (\tfrac{1}{n}\sum_{p=1}^{n} \phi_p\phi_p^T)(\sum_{j=1}^{n} \alpha_j\phi_j)\rangle \\
\lambda n \sum_{j=1}^{n} \langle\phi_i, \phi_j\rangle\alpha_j &= \langle\phi_i, \sum_{j=1}^{n}\sum_{p=1}^{n} \phi_p\phi_p^T\alpha_j\phi_j\rangle \\
&= \sum_{j=1}^{n} \alpha_j \sum_{p=1}^{n} \langle\phi_i, \phi_p\rangle\langle\phi_p, \phi_j\rangle \\
\lambda n \sum_{j=1}^{n} k_{ij}\alpha_j &= \sum_{j=1}^{n} \alpha_j \sum_{p=1}^{n} k_{ip}k_{pj}.
\end{aligned}$$

Quando i varia de 1 até n podemos escrever o sistema de n equações do tipo acima como a equação matricial

$$\lambda n K \alpha = K^2 \alpha \tag{3.17}$$

onde α denota o vetor coluna com entradas $\alpha_1, \ldots, \alpha_n$. Como K é simétrica, para encontrar as soluções de 3.17 basta resolver a equação de autovalores

$$\lambda n \alpha = K \alpha \tag{3.18}$$

pois, obviamente, todas as soluções de 3.18 satisfazem 3.17, e vice-versa.

Além disso, é possível mostrar que K é positiva semidefinida, portanto 3.18 possui autovalores não-negativos como solução. Assim como no caso do PCA linear, ordenamos os autovalores em ordem crescente. No entanto, o que nos interessa são as projeções dos pontos ϕ_i nas componentes principais (autovetores ψ da matriz de covariância $\widetilde{C}$) do conjunto de pontos $\{\phi_i\}_{i=1}^n$ contido no espaço de característica $\mathcal{H}$. Para isto, encontraremos uma representação implícita dessas projeções em função apenas dos autovetores α da matriz de *kernels* K.

Sejam $\lambda^1 \leq \lambda^2 \leq \ldots \leq \lambda^n$ os autovalores soluções de 3.18 ordenados em ordem crescente e $\alpha^1, \alpha^2, \ldots, \alpha^n$ os autovetores correspondentes, e seja λ^p o primeiro autovalor não nulo. Para calcular as projeções de ϕ_i em $\{\psi^r\}_{r=p}^n$, o conjunto de autovetores $\{\psi^r\}_{r=p}^n$ deve formar uma base ortonormal do subespaço gerado pelos pontos $\{\phi_i\}_{i=1}^n$. Como cada autovetor ψ^r pode ser escrito como em 3.15 e o conjunto de autovetores $\{\alpha^r\}_{r=p}^n$ é linearmente independente, então é possível mostrar que o conjunto $\{\psi^r\}_{r=p}^n$ também é linearmente independente, portanto é uma base do subespaço gerado pelo conjunto $\{\phi_i\}_{i=1}^n$.

Além disso, como

$$\begin{aligned}
\langle \psi^r, \psi^s \rangle &= \langle \sum_{i=1}^n \alpha_i^r \phi_i, \sum_{j=1}^n \alpha_j^s \phi_j \rangle \\
&= \sum_{i=1}^n \sum_{j=1}^n \alpha_i^r \alpha_j^s \langle \phi_i, \phi_j \rangle \\
&= \sum_{i=1}^n (\alpha_j^r \sum_{j=1}^n \alpha_j^s k_{ij}) \\
&= \langle \alpha^r, K \alpha^s \rangle,
\end{aligned}$$

então $\langle \psi^r, \psi^s \rangle = n\lambda^s \langle \alpha^r, \alpha^s \rangle$. Logo, $\langle \psi^r, \psi^s \rangle = 0$, para $r \neq s$, pois os autovetores α^r e α^s são ortogonais. E para que tenhamos $\langle \psi^r, \psi^s \rangle = 1$ quando $r = s$, basta normalizar os autovetores α^r de tal forma que $n\lambda^r \langle \alpha^r, \alpha^r \rangle = 1$.

Feito isso, teremos uma base ortonormal $\{\psi^r\}_{r=p}^{n}$ do subespaço dos pontos $\{\phi_i\}_{i=1}^{n}$. Donde, cada ponto ϕ_i é representado por sua projeção

$$\varphi_i = (\langle\phi_i, \psi^p\rangle, \ldots, \langle\phi_i, \psi^n\rangle),\ 1 \leq p \leq n \tag{3.19}$$

e cada coordenada pode ser calculada implicitamente por

$$\begin{aligned}\langle\phi_i, \psi^r\rangle &= \langle\phi_i, \sum_{j=1}^{n} \alpha_j^r \phi_j\rangle \\ &= \sum_{j=1}^{n} \alpha_j^r \langle\phi_i, \phi_j\rangle,\end{aligned}$$

ou seja,

$$\langle\phi_i, \psi^r\rangle = \sum_{j=1}^{n} \alpha_j^r k_{ij}. \tag{3.20}$$

Já as coordenadas de um novo exemplo $x \in \mathbb{R}^d$, do espaço de características inicial, podem ser calculadas como a projeção $\varphi = (\langle\phi(x), \psi^p\rangle, \ldots, \langle\phi(x), \psi^n\rangle)$ de $\phi(x)$ nos autovetores ψ^r, onde

$$\langle\phi(x), \psi^r\rangle = \sum_{j=1}^{n} \alpha_j^r k(x, x_j). \tag{3.21}$$

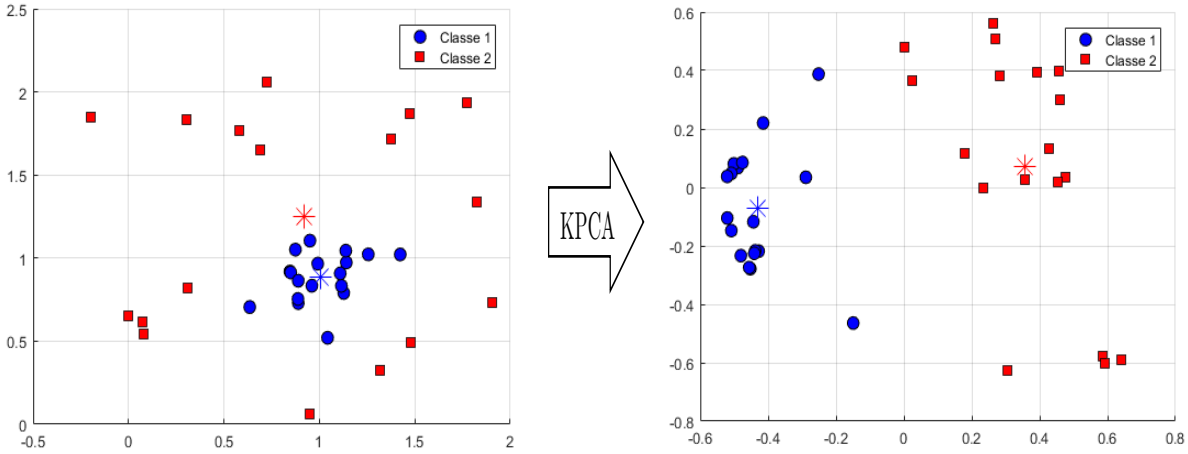
Note que na formulação acima, estamos projetando os pontos ϕ_i nos $n - p + 1$ autovetores ψ_r correspondentes aos $\lambda^p \leq \ldots \leq n$ não nulos. Porém, na maioria das vezes, muitos destes autovalores são próximos de zero, indicando que na direção do autovetor correspondente há pouca variância nos dados. Este fato pode ser levado em conta no momento de calcular as projeções segundo a Equação 3.19, pois pode reduzir drasticamente a dimensão dos vetores φ_i com pouca ou nenhuma perda de informação, em troca de um custo computacional menor.

A Figura 12 mostra o efeito do KPCA sobre um conjunto de treinamento não separável linearmente. Os pontos foram mapeados em um espaço de características de dimensão alta utilizando um *kernel gaussiano* e depois projetados em duas componentes principais.

Centralização dos Pontos em Espaço de Alta Dimensão

Para aplicar o método do PCA ao conjunto de pontos $\{\phi_i\}_{i=1}^{n}$ supomos que os dados estavam centrados na origem, isto é, que $\sum_{i=1}^{n} \phi_i = 0$. No entanto, como não conhecemos $\phi : \mathbb{R}^d \to \mathcal{H}$ explicitamente, então não é possível fazer essa operação diretamente sobre os pontos ϕ_i. Além disso, centralizar os pontos x_i no espaço de características inicial não

Figura 12 – Ilustração de um caso onde duas classes de exemplos não são separáveis linearmente, à esquerda; e o efeito produzido pelo KPCA, à direita. Os pontos em formato de asterisco ($*$) representam os centros das classes antes e após a aplicação do KPCA.



Fonte – Autor, 2017.

garante que as imagens ϕ_i serão centralizadas em um espaço de características de dimensão mais alta.

Para resolver este problema, observe que dado o conjunto $\{\phi_i\}_{i=1}^n$ de n pontos $\phi_i \in \mathcal{H}$, então o novo conjunto $\{\widetilde{\phi}_i\}_{i=1}^n$, onde

$$\widetilde{\phi}_i := \phi_i - \frac{1}{n}\sum_{i=1}^{n}\phi_i, \tag{3.22}$$

está centrado na origem, isto é, $\sum_{i=1}^n \widetilde{\phi}_i = 0$.

Como não temos os pontos $\widetilde{\phi_i}$ representados explicitamente, não podemos calcular a matriz $\widetilde{K}$ dos *kernels* $\widetilde{k}_{ij} = \langle\widetilde{\phi}_i, \widetilde{\phi}_j\rangle$ diretamente. No entanto

$$\begin{aligned}
\widetilde{k}_{ij} &= \langle\widetilde{\phi}_i, \widetilde{\phi}_j\rangle \\
&= \langle\phi_i - \tfrac{1}{n}\sum_{r=1}^{n}\phi_r, \phi_j - \tfrac{1}{n}\sum_{s=1}^{n}\phi_s\rangle \\
&= \langle\phi_i, \phi_j\rangle - \tfrac{1}{n}\sum_{s=1}^{n}\langle\phi_i, \phi_s\rangle - \tfrac{1}{n}\sum_{r=1}^{n}\langle\phi_r, \phi_j\rangle + \tfrac{1}{n^2}\sum_{r,s=1}^{n}\langle\phi_r, \phi_s\rangle \\
&= k_{ij} - \tfrac{1}{n}\sum_{s=1}^{n}k_{is} - \tfrac{1}{n}\sum_{r=1}^{n}k_{rj} + \tfrac{1}{n^2}\sum_{r,s=1}^{n}k_{rs} \\
&= k_{ij} - \sum_{j=1}^{n}k_{is}(1_n)_{sj} - \sum_{r=1}^{n}(1_n)_{ir}k_{rj} + \sum_{r,s=1}^{n}(1_n)_{ir}k_{rs}(1_n)_{sj},\ (1_n)_{ij} := 1/n.
\end{aligned}$$

Logo

$$\widetilde{K} = K - 1_n K - K 1_n + 1_n K 1_n, \tag{3.23}$$

onde 1_n é uma matriz com entradas $(1_n)_{ij} := 1/n$.

Assim, antes de resolver o problema de autovalores dado na Equação 3.18 da matriz de *kernels* K, deve-se executar um passo de pré-processamento recalculando as entradas de K como sendo iguais as entradas da matriz $\widetilde{K}$ definida pela Equação 3.23.

Vale a pena também mencionar que nesta subseção supomos de forma implícita a existência de uma função não linear $\phi : \mathbb{R}^d \to \mathcal{H}$ que mapeia cada $x_i \in \mathbb{R}^d$ para $\phi_i = \phi(x)$ em um espaço $\mathcal{H}$ de dimensão possivelmente infinito, onde o produto interno $\langle \phi(x), \phi(y) \rangle = \phi(y)^T \phi(x)$ é dado por uma função de *kernel* $k(\cdot, \cdot)$ no espaço de características inicial $\mathbb{R}^d$. No entanto, surge a pergunta: dada uma função de *kernel* positiva semidefinida $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, existe realmente uma função não linear $\phi : \mathbb{R}^d \to \mathcal{H}$ tal que $k(x, y) = \langle \phi(x), \phi(y) \rangle = \phi(y)^T \phi(x)$? A resposta para esta questão é positiva, e pode ser provada devido a um resultado conhecido como **Teorema de Mercer** [42].

3.4.6 Kernel Large Margin Nearest Neighbor (KLMNN)

Na Seção 3.4.4 explicamos o funcionamento do algoritmo LMNN passando como entrada o conjunto de treinamento supervisionado $\mathcal{T} = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subset \mathbb{R}^d \times \mathcal{C}$. No entanto, o algoritmo se tornava ineficiente ao se deparar com casos dos tipos mostrados na Figura 10. Isto motivou o uso do KPCA, que nada mais é do que uma forma de reorganizar os pontos em um espaço de dimensão alta de forma que os mesmos possam ser separados linearmente, como mostra a Figura 12.

A partir do conjunto de treinamento inicial $\mathcal{T}$ calculamos um novo conjunto de treinamento supervisionado $\widetilde{\mathcal{T}} = \{(\varphi_1, y_1), \ldots, (\varphi_n, y_n)\} \subset \mathbb{R}^m \times \mathcal{C}$, onde cada $\varphi_i \in \mathbb{R}^m$ é obtido através da Equação 3.19 e $\mathbb{R}^m$ é o subespaço dos m autovetores ψ^r mais relevantes do conjunto $\{\phi_i\}_{i=1}^n$, e $\mathcal{C} = \{0, 1, \ldots, c\}$ é o conjunto de rótulos de classes inicial.

Tirando proveito do efeito que o KPCA provoca nos pontos, aplicamos o algoritmo LMNN ao novo conjunto de treinamento supervisionado $\widetilde{\mathcal{T}}$ e procuramos por uma métrica ótima no espaço de características $\mathbb{R}^m$. A este processo de aplicar o LMNN com os pontos de saída do KPCA juntamente com as informações de supervisão iniciais chamaremos de *Kernel Large Margin Nearest Neighbor* (KLMNN). Para fins de classificação de um novo exemplo $x' \in \mathbb{R}^d$, devemos recalcular suas coordenadas no espaço $\mathbb{R}^m$ através da Equação 3.21. Isto é feito para calcular as distâncias entre x' e os exemplos x_i com a

métrica encontrada pelo KLMNN.

Um fato importante a saber, no contexto de aprendizagem de métrica, é a validade da representação de um ponto ϕ_i de dimensão infinita como um vetor φ_i de dimensão finita. Pois como o problema de aprendizagem de métrica é formulado como um problema de otimização, então dada uma função de custo $\mathcal{L}(\cdot)$ como definida pela Equação 3.5, é necessário saber se o valor ótimo de $\mathcal{L}(\cdot)$ baseado nos pontos ϕ_i é igual ao valor ótimo de $\mathcal{L}(\cdot)$ baseado nos pontos φ_i. Este fato é garantido pelo **Teorema de Representação**, enunciado e demonstrado com detalhes no estudo de Chatpatanasiri et al.[40].

3.5 Modelo Oculto de Markov

Nesta seção descrevemos de forma breve o Modelo Oculto de Markov, uma ferramenta que vem sendo amplamente utilizada pela comunidade de aprendizagem de máquina para modelagem de problemas nas quais os dados são representados como uma sequência de observações ao longo do tempo.

Começaremos definindo Cadeia de Markov, uma ferramenta mais simples que o Modelo Oculto de Markov generaliza, na qual as *observações* são os próprios estados do modelo.

3.5.1 Cadeia de Markov

Seja $S = \{s_1, s_2, ..., s_n\}$ um conjunto de estados e $X : \Omega \to S$ uma variável aleatória de $\Omega = \{t : \ t \geq 0\}$ a assumir valores do conjunto de estados S em diferentes instantes de tempo t.

Definimos Cadeias de Markov como uma sequência de estados $X_t, X_{t+1}, \ldots, X_{t+p}$ que satisfaça a propriedade de que o estado da cadeia no instante de tempo seguinte dependa apenas do estado da cadeia no tempo atual, ou seja:

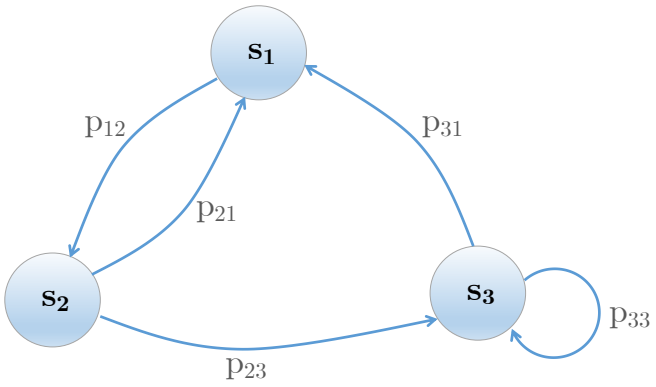
$$P(X_{t+1} = s \mid X_t, X_{t-1}, \ldots, X_1, X_0) = P(X_{t+1} = s \mid X_t). \tag{3.24}$$

Esta propriedade é conhecida como **propriedade de Markov** e é o que caracteriza as Cadeias de Markov.

A cadeia inicia em um dos estados com uma probabilidade preestabelecida e move-se aleatoriamente de um estado para outro de acordo com probabilidades de transições

entre estados. Denotaremos a probabilidade de transição do estado s_i para o estado s_j por p_{ij}. Vale a pena comentar que essa probabilidade é independe do instante de tempo t considerado, ou seja, $p_{ij} = P(X_{t+1} = s_j \mid X_t = s_i)$ para todo $t \geq 0$. A Figura 13 exibe um grafo orientado que ilustra um exemplo de Cadeia de Markov de três estados.

Figura 13 – Representação de uma Cadeia de Markov como um grafo orientado. Os vértices são os estados da cadeia e as arestas reprensentam a transição de um estado s_i para s_j com uma probabilidade p_{ij}.



Fonte – Autor, 2017.

Normalmente essas informações de probabilidades de transições entre estados são representadas como uma matriz $P = (p_{ij})_{n \times n}$, chamada de **matriz de transição de probabilidade**, onde a ij-ésima entrada representa a probabilidade p_{ij} de transição do estado s_i para o estado s_j. Quando não se sabe ao certo em qual estado uma Cadeia de Markov deve iniciar, costuma-se considerar um vetor de probabilidades inicial $p = (p_1, p_2, \ldots, p_n)$ para representar que a cadeia inicia no estado s_i com probabilidade p_i.

Como cada transição entre estados depende apenas do estado atual na qual a cadeia se encontra, então a probabilidade de ocorrer uma sequência de estados $X_t, X_{t+1}, \ldots, X_{t+p}$ é dada por

$$P(X_t, X_{t+1}, \ldots, X_{t+p}) = \prod_{k=0}^{p-1} P(X_{t+k+1} \mid X_{t+k}),\ t \geq 0. \tag{3.25}$$

Desse modo, Cadeias de Markov podem ser utilizadas para classificação de padrões que têm natureza sequencial. Considere que cada classe de um determinado problema de classificação é representada por uma cadeia de Markov. Dada uma sequência de estados a ser classificada, o modelo mais provável que gerou tal sequência pode ser encontrado calculando-se as probabilidades segundo a Equação 3.25.

Definiremos agora o Modelo Oculto de Markov, explicitando as semelhanças e diferenças existentes para uma Cadeia de Markov.

3.5.2 Modelo Oculto de Markov (HMM)

Um *Modelo Oculto de Markov* (HMM) é um modelo probabilístico composto de um conjunto $W = \{w_1, w_2, \ldots, w_m\}$ chamado de *conjunto de observações*; uma variável aleatória $Y : \Omega \to W$, a assumir *observações* do conjunto W ao longo do tempo; e, assim como no caso das Cadeias de Markov, também é composto por um conjunto de estados $S = \{s_1, s_2, \ldots, s_n\}$ e uma variável aleatória $X : \Omega \to S$, a assumir estados do conjunto S ao longo do tempo. A diferença é que em um HMM não temos certeza sobre os estados do modelo e nem sobre qual o estado no tempo t, ou seja, X_t é oculto. Além disso, em um HMM assume-se que a variável aleatória Y depende única e exclusivamente do estado na qual a variável aleatória X encontra-se. Dessa forma, se q_{ik} é a probabilidade de uma observação w_k ocorrer, dado que ocorreu o estado s_i no instante de tempo t, então

$$q_{ik} = P(Y_t = w_k \mid X_t = s_i),\ t \geq 0. \tag{3.26}$$

Assim como as Cadeias de Markov, um HMM também pode ser representado visualmente como um grafo orientado. A Figura 14 ilustra um HMM de três estados e quarto observações.

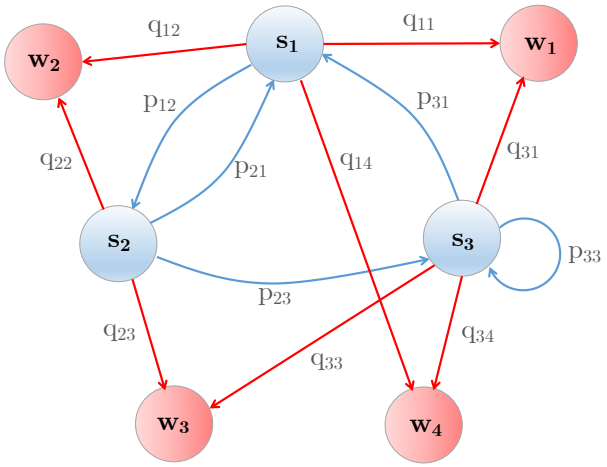
Normalmente essas informações de probabilidades q_{ik} são armazenadas como entradas de uma matriz de probabilidades Q, de ordem $n \times m$, onde n é a quantidade de estados e m a quantidade de observáveis de um HMM. E assim como no caso das Cadeias de Markov, as probabilidades de transições p_{ij} entre estados também são armazenadas como entradas de uma matriz de transição P, de ordem $n \times n$.

Como cada observação gerada Y_t depende apenas do estado atual X_t, então a probabilidade de ocorrer uma sequência de observações $\mathcal{Y} = \{Y_t, Y_{t+1}, \ldots, Y_{t+p}\}$ é dada por

$$P(\mathcal{Y}) = \sum_{\mathcal{X}} P(\mathcal{Y} \mid \mathcal{X}) \cdot P(\mathcal{X}),\ t \geq 0, \tag{3.27}$$

onde o somatório varia sobre todas as sequências de estados $\mathcal{X} = \{X_t, X_{t+1}, \ldots, X_{t+p}\}$ possíveis.

Figura 14 – Representação de um HMM como um grafo orientado. Os vértices azuis são os estados e os vermelhos as observações. Já as arestas azuis reprensentam as transições entre estados com probabilidades p_{ij} e as vermelhas indicam quais os possíveis observáveis gerados a partir de um estado s_i com probabilidade q_{ik}.



Fonte – Autor, 2017.

A grande explosão combinatória de possíveis sequências torna o cálculo desta probabilidade impraticável. Algoritmos eficientes para o cálculo de probabilidades em HMMs podem ser encontrados em Rabiner[43].

4 MÉTODO

Neste capítulo explicamos detalhadamente o método de reconhecimento de poses e gestos da mão proposto no trabalho.

4.1 Treinamento de Poses

Em qualquer sistema de aprendizagem de máquina temos uma fase inicial chamada de **fase de treinamento**, que é a fase de captura de dados que servirão de base para a máquina aprender a tomar decisões sobre novas instâncias.

Esta fase consiste na captura de exemplos de classes de poses que servirão de base para o algoritmo de reconhecimento classificar novas poses. Quando a aprendizagem é supervisionada, cria-se um conjunto de treinamento composto por vetores de descritores de poses de mão, associado com informações de qual classe de poses cada vetor de descritores representa (rótulos de classes). Explicamos detalhadamente este processo nas próximas seções.

4.1.1 Extração de Descritores

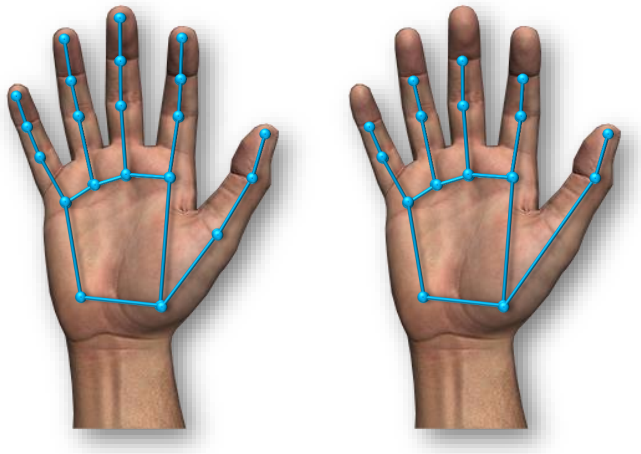
Como definimos na Seção 3.3.1, os descritores compõem um conjunto de atributos relevantes para descrever um objeto de entrada em um problema de aprendizagem de máquina. Dessa forma, a escolha dos descritores apropriados para determinada aplicação é uma tarefa que exige bastante atenção, pois uma escolha de descritores errada pode comprometer a capacidade de generalização do modelo de aprendizagem pretendido.

Para o problema de reconhecimento de gestos do qual se refere este trabalho, procuramos fazer a escolha de descritores de mão que mais se ajusta as necessidades de distinção de diferentes tipos de poses estásticas. Para isto, optamos por separar a escolha dos descritores de mão em quatro categorias de poses estáticas: **invariantes**, **variantes por rotação**, **variantes por translação** e **variantes por rotação e translação**.

Para a visualização de poses estáticas utilizamos um modelo do esqueleto da mão composto pela posição espacial de 21 juntas detectadas pelo sensor Leap Motion, ilustradas na Figura 15 (a). Todavia, para a extração dos descritores escolhidos nem todas essas juntas são relevantes. Assim, optamos por desconsiderar as posições das pontas dos dedos

(exceto do polegar), e a posição da junta do polegar que faz a ligação dos ossos falange proximal e falange intermédia, como ilustra a Figura 15 (b). Para relembrar a nomenclatura dos ossos da mão utilizada pela API do Leap Motion consultar a Figura 7. As posições das pontas dos dedos são irrelevantes para nosso modelo pelo fato de os ângulos entre os ossos falange intermédia e distal serem irrelevantes, visto que é inerente da própria anatomia da mão humana este ângulo depender do ângulo de abertura entre os ossos falange proximal e intermédia. Tente por exemplo dobrar a junta de ligação dos ossos falange proximal e intermédia sem dobrar a junta de ligação dos ossos intermédia e distal. Esta limitação, no entanto, não é válida para o dedo polegar, onde a limitação de movimento ocorre agora na junta de ligação dos ossos falange proximal e falange intermédia. Neste caso optamos por desconsiderar tal junta e considerar os ossos falange proximal e intermédia como se fossem um único osso, ver Figura 15 (b) abaixo.

Figura 15 – (a) Juntas da mão utilizadas para visualização da pose. (b) Juntas utilizadas para extração de descritores da mão.



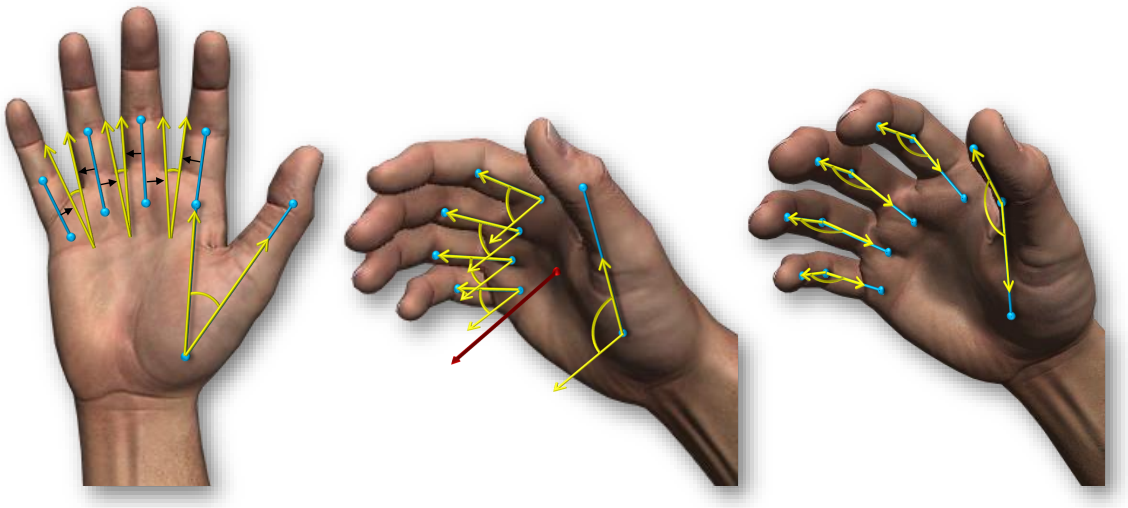
(a) (b)

Fonte – Autor, 2017.

Na categoria de **poses invariantes** todas as poses estáticas são representadas com um conjunto de descritores invariantes por rotação e translação. Para isto, foram escolhidos como descritores um conjunto de 14 ângulos: os 4 ângulos entre as falanges proximais, Figura 16 (a); os 5 ângulos entre o vetor normal da palma e as falanges proximais, Figura 16 (b); e os 5 ângulos entre os ossos falanges intermédias e falanges proximais de cada um dos cinco dedos da mão, Figura 16 (c). Como cada exemplo de pose estática da mão nesta categoria é representada com 14 atributos de ângulos (todos medidos em radianos),

então passamos a representar uma pose estática invariante como um vetor de descritores $x \in \mathbb{R}^{14}$.

Figura 16 – Ilustração dos ângulos, em amarelo, utilizados como descritores de poses invariantes a rotação e translação. Na Figura (a), os vetores, que indicam a direção das falanges proximais, estão deslocados para obter uma melhor visualização dos ângulos entre os mesmos.



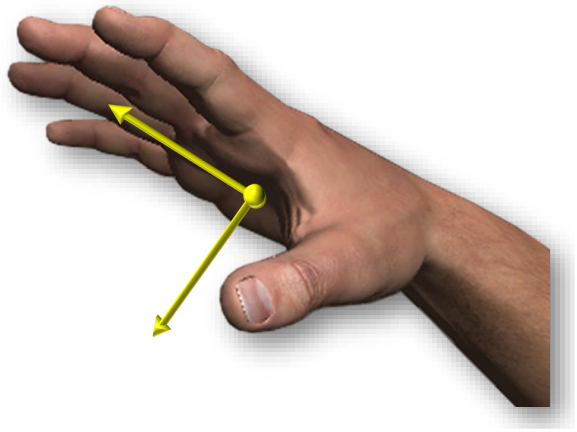
(a) (b) (c)

Fonte – Autor, 2017.

Já na categoria de **poses variantes por rotação** todas as poses estáticas são representadas com um conjunto de descritores variantes por rotação, porém, que ainda sejam invariantes por translação. Ou seja, nesta categoria diferentes exemplos de poses estáticas que na categoria de poses invariantes pertenceriam a mesma classe, aqui podem pertencer a classes de poses distintas dependendo da rotação da mão definida pelos vetores normal e direcional da palma destacados na Figura 17. Para isto, foram escolhidos todos os 14 descritores de ângulos da categoria de poses invariantes, acrescentada das 3 coordenadas do vetor normal da palma e também das 3 coordenadas do vetor direcional da palma da mão, totalizando assim 20 descritores. Logo, cada exemplo de pose estática nesta categoria passa agora a ser representada como um vetor de descritores $x \in \mathbb{R}^{20}$.

Na categoria de **poses variantes por translação** todas as poses estáticas são representadas com um conjunto de descritores variantes por tranlação, porém, que ainda sejam invariantes por rotação. Desta forma, nesta categoria as diferentes poses estáticas que na categoria de poses invariantes pertenceriam a mesma classe, aqui podem pertencer a classes de poses distintas dependendo da posição espacial da mão em relação ao sensor Leap

Figura 17 – Ilustração dos vetores normal e direcional da palma da mão. Note que estes vetores definem a rotação da palma da mão.

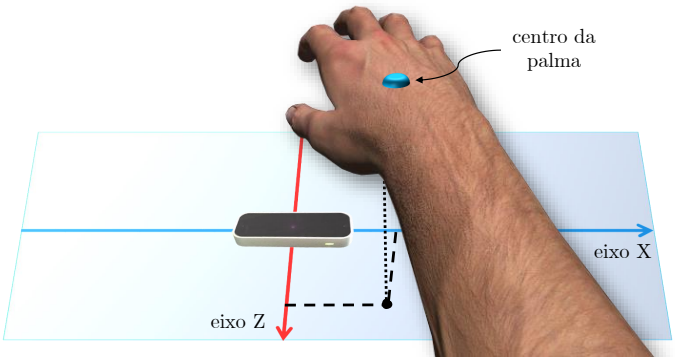


Fonte – Autor, 2017.

Motion. Isto é importante quando se deseja que a máquina reconheça poses que dependam da posição na qual foram realizadas, e também para reconhecer gestos caracterizados apenas por translações de uma mesma pose da mão. Para isto, foram escolhidos todos os 14 descritores de ângulos da categoria de poses invariantes, acrescentada das coordenadas horizontais do centro da palma da mão (eixos X e Z de acordo com o sistema de coordenadas do sensor Leap Motion mostrado na Figura 6), totalizando assim 16 descritores. Optamos por não utilizar a coordenada vertical (eixo Y) para dar ao usuário a liberdade de optar por realizar as poses mais próximas ou mais distantes do sensor, sem que o desempenho de reconhecimento seja afetado. Além disso, como no sistema de coordenadas do sensor os valores das coordenadas espaciais do centro da palma assumem valores altos quando comparados com os valores dos ângulos em radianos, optamos por normalizar as coordenadas do centro da palma por um fator de 1/300. Finalmente, cada exemplo de pose estática fica sendo representada como um vetor de descritores $x \in \mathbb{R}^{16}$.

Já na categoria de **poses invariantes por rotação e translação** todas as poses estáticas são representadas com um conjunto de descritores variantes simultaneamente por translação e rotação. Isto significa que poses estáticas que nas três classes anteriores pertenceriam a mesma classe de poses, aqui podem pertencer a classes de poses distintas dependendo da orientação e posição espacial da mão com relação ao sensor. Para esta categoria, foram concatenados todos os descritores das categorias anteriores, ou seja, os 14 descritores de ângulos da categoria invariante, os 6 da categoria variantes por rotação e

Figura 18 – Ilustração da projeção do centro da palma da mão no plano XZ. Esta projeção define a translação horizontal da mão em relação ao sensor Leap Motion.



Fonte – Autor, 2017.

os 2 da categoria variantes por translação, totalizando assim 22 descritores. Desta forma, cada exemplo de pose estática percente a esta categoria passa agora a ser representada como um vetor de descritores $x \in \mathbb{R}^{22}$.

4.1.2 Conjuntos de Treinamento e Validação

Definidas as 4 categorias de poses estáticas e seus respectivos descritores, vamos agora a etapa de criação dos conjuntos de treinamento de cada categoria de poses estáticas. Para melhor esclarecimento de cada etapa, acompanhar a Figura 19.

Escolhemos primeiramente qual o tipo de categoria de poses estásticas que desejamos criar o conjunto de treinamento, de acordo com o quadro 1 da Figura 19, e definimos as classes de poses estáticas que serão treinadas, quadro 2 da Figura 19. Cada classe de poses passa então a ser identificada por um rótulo de classe. Para nossos objetivos utilizaremos o conjunto de rótulos de classes como o subconjunto dos números naturais: $\mathcal{C} = \{1, 2, \ldots, m\}$.

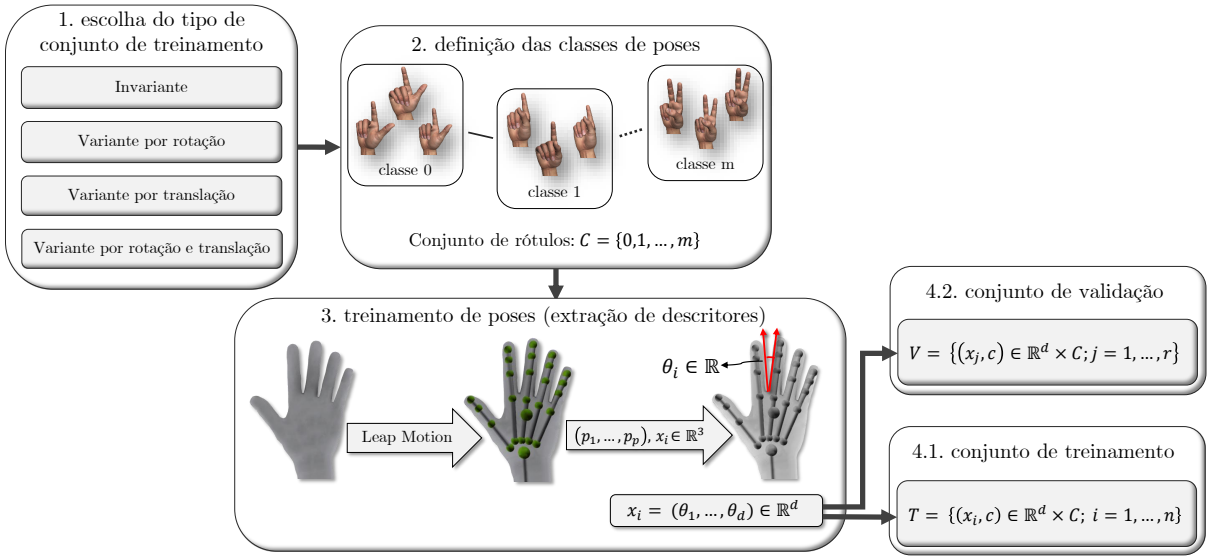
Definidas as classes de poses estáticas e seus respectivos rótulos, o usuário posiciona a mão acima do sensor e realiza exemplos de poses daquela classe, pressionando uma tecla específica para gravação de cada exemplo, como ilustra o quadro 3 da Figura 19. Mesmo que estes exemplos definam a mesma pose, é importante que o usuário realize os exemplos da mesma classe de formas variadas, afim de generalizar possíveis variações de atributos de uma classe de poses.

A cada exemplo de pose realizada, é feito a gravação do exemplo na forma de um

par ordenado $(x_i, y_i) \in \mathbb{R}^d \times \mathcal{C}$, onde $x_i \in \mathbb{R}^d$ é o vetor de descritores que representa o i-ésimo exemplo de pose realizada e y_i é o rótulo de classe da qual o exemplo realizado pertence, d é quantidade de descitores utilizados na categoria de poses em questão. Por exemplo, na categoria de poses invariantes, temos que $d = 14$, isto implica que o vetor de descritores das poses dessa categoria pertencem ao espaço euclidiano de dimensão 14.

Em nossos experimentos, o usuário forneceu em média 12 exemplos de poses por classe. Destes, 8 foram armazenados no conjunto de treinamento (quadro 4.1 da Figura 19) e os outros 4 foram separados para serem utilizados no conjunto de validação (quadro 4.2 da Figura 19).

Figura 19 – Criação dos conjuntos de treinamento e validação.



Fonte – Autor, 2017.

Ambos os conjuntos de treinamento e validação são utilizados na fase de aprendizagem de métrica, com a diferença de que o conjunto de treinamento é utilizado diretamente como entrada do algoritmo de aprendizagem de métrica, enquanto o conjunto de validação é utilizado apenas para estimar os parâmetros ótimos do algoritmo.

Os conjuntos de treinamento e validação têm os seguintes formatos: $\mathcal{T} = \{(x_i, y_i) \in \mathbb{R}^d \times \mathcal{C};\ i = 1, \dots, n\}$, $\mathcal{V} = \{(x_j, y_j) \in \mathbb{R}^d \times \mathcal{C};\ j = 1, \dots, r\}$, respectivamente.

4.2 Aprendizagem da Métrica

Na Seção 3.4, vimos que algoritmos de aprendizagem de máquina são mais eficientes quando utilizada internamente uma função de distância específica para o problema. Levando isto em conta, nesta etapa, utilizamos o conjunto de treinamento criado na seção anterior para aprender uma métrica global no espaço de descritores de poses da mão. Esta métrica é então utilizada com o objetivo de melhorar o desempenho de classificação automática de novas poses.

Para a aprendizagem de métrica neste trabalho, utilizamos o algoritmo LMNN com uma implementação em Matlab desenvolvida pelo próprio autor do método. Esta implementação pode pode ser encontrada em [39].

4.2.1 Caso Linear

Para aprender uma métrica linear no espaço de características de poses da mão, primeiramente determinamos alguns parâmetros do algoritmo LMNN necessários para aprendizagem da métrica, tais como: quantidade de vizinhos alvo (como definido na Subseção 3.4.4), dimensão de saída do espaço de características e número de iterações do algoritmo.

O parâmetro quantidade de vizinhos alvo foi escolhido como sendo o menor valor dentre as quantidades de exemplos de cada classe do conjunto de treinamento. A dimensão de saída do espaço de características é definida como sendo igual a dimensão de entrada; e o número máximo de iterações do algoritmo foi escolhido empiricamente. Este último parâmetro tem a ver com a quantidade máxima de iterações que o algoritmo LMNN deverá executar para encontrar a métrica ótima.

Finalmente, passamos o conjunto de treinamento $\mathcal{T} = \{(x_i, y_i) \in \mathbb{R}^d \times \mathcal{C};\ i = 1, \ldots, n\}$, juntamente com os parâmetros escolhidos, como entrada do algoritmo LMNN, que executa um processo de otimização convexa, e retorna uma matriz $G \in \mathbb{R}^{d \times d}$ que determina a melhor distância de Mahalanobis

$$d_G(x, y) = (x - y)^T G^T G (x - y) = \|Gx - Gy\|^2 \tag{4.28}$$

entre dois vetores de descritores de exemplos de poses x e $y \in \mathbb{R}^d$ quaisquer.

O problema dessa formulação linear é que nem sempre os dados de treinamento são

linearmente separáveis, e portanto, a métrica aprendida pode não ser tão eficiente. Isto nos motiva a utilizar o caso de aprendizagem de métrica não linear, descrito na Subseção 3.4.5.

4.2.2 Caso Não Linear

Para a aprendizagem de uma métrica não linear, foi utilizado o método *Kernel Principal Component Analysis* (KPCA), que possibilita a projeção não linear dos exemplos de treinamento para um espaço de dimensão mais alta, onde uma métrica pode ser aprendida com mais eficiência. Uma implementação (em Matlab) deste método pode ser encontrada em [44].

Primeiramente, definimos manualmente alguns parâmetros do algoritmo KPCA, tais como: **tipo de *kernel***, **argumento do *kernel*** e **nova dimensão** do espaço de características.

O tipo de *kernel* se refere ao *kernel* PSD (ver Definição 3.5) que será utilizado para o cálculo da projeção não linear dos exemplos de treinamento em um novo espaço de características. Este é o parâmetro mais importante do algoritmo, pois a escolha do tipo de *kernel* pode mudar consideravelmente a forma como os exemplos são projetados no novo espaço. Os *kernels* implementados e suportados pelo algoritmo são: linear, polinomial, gaussiano e sigmoidal. Cada um destes *kernels* tem suas particularidades. Neste trabalho, optamos por utilizar o *kernel* gaussiano, definido como:

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right), \tag{4.29}$$

onde σ é o parâmetro **argumento do *kernel*** que também deve ser passado para o algoritmo. Optamos por este *kernel* uma vez que o mesmo tém se mostrado ser um dos mais eficientes em trabalhos de pesquisa na área de aprendizagem de máquina.

O parâmetro **argumento do *kernel*** é definido por padrão como $\sigma = 1$. No entanto, optamos por fazer a calibração do parâmetro do *kernel* manualmente, levando em conta o desempenho de classificação.

Já o parâmetro **nova dimensão** se refere à dimensão do novo espaço de características para onde os exemplos de treinamento são mapeados não linearmente. Normalmente, o KPCA projeta os exemplos em um espaço euclideano de dimensão no máximo igual a quantidade de exemplos do conjunto de treinamento. No entanto, como vimos na Subseção

3.4.5, a dimensão desse espaço pode ser reduzida pela projeção dos exemplos de treinamento nos autovetores mais relevantesda matriz de covariância definida pela Equação 3.13. Dessa forma, a dimensão do espaço onde os exemplos são projetados pode ser reduzida com nenhuma perda de informação. No nosso modelo, podemos escolher manter a dimensão do espaço igual a quantidade de exemplos ou executar a redução de dimensão automática optando por não projetar os exemplos em autovetores correspondentes a autovalores próximos de zero.

Definidos estes parâmetros, passamos o conjunto de treinamento $\mathcal{T} = \{(x_i, y_i) \in \mathbb{R}^d \times \mathcal{C};\ i = 1, \ldots, n\}$, juntamente com os parâmetros citados acima, como entrada do algoritmo KPCA, que executa uma transformação não linear dos exemplos de treinamento para um espaço de dimensão maior, e retorna um novo conjunto de treinamento $\mathcal{T}' = \{(\varphi_i, y_i) \in \mathbb{R}^D \times \mathcal{C};\ i = 1, \ldots, n\}$, mantendo a correspondência $x_i \mapsto \varphi_i$ e com φ_i herdando o rótulo de classe y_i de x_i, para todo $i = 1, \ldots, n$.

Finalmente, passamos este novo conjunto de treinamento $\mathcal{T}'$, com os devidos pâmetros, como entrada do algoritmo LMNN. E, assim como no caso linear, o algoritmo executa um processo de otimização convexa e retorna uma matriz $Q \in \mathbb{R}^{D \times D}$ que corresponde a melhor distância de Mahalanobis

$$d_Q(\varphi, \varphi') = (\varphi - \varphi')^T Q^T Q(\varphi - \varphi') = \|Q\varphi - Q\varphi'\|^2 \tag{4.30}$$

entre dois exemplos φ e φ' quaisquer do novo espaço de características $\mathbb{R}^D$.

Note que como LMNN retorna uma matriz, então a métrica aprendida nesse novo espaço de características $\mathbb{R}^D$ é linear. No entanto, como a correspondência $x_i \mapsto \varphi_i$ é não linear, então a métrica linear aprendida no espaço $\mathbb{R}^D$ corresponde a uma métrica não linear aprendida no espaço de características inicial $\mathbb{R}^d$.

4.3 Reconhecimento de Poses

Após a fase de aprendizagem da melhor métrica para o conjunto de treinamento $\mathcal{T} = \{(x_i, y_i) \in \mathbb{R}^d \times \mathcal{C};\ i = 1, \ldots, n\}$, passamos a etapa de reconhecimento automático de novos exemplos de poses.

Nesta etapa, utiliza-se um classificador de poses que recebe como entrada um novo exemplo de pose estática $x \in \mathbb{R}^d$ e, baseado em uma regra de classificação, retorna a classe $y \in \mathcal{C}$ que mais se assemelha ao exemplo realizado. No entanto, em aprendizagem de

máquina, nem sempre uma nova entrada se assemelha a alguma das classes de exemplos treinadas, ou seja, nem toda pose deve ser classificada como alguma das classes de poses treinadas. Dessa forma, em geral, algoritmos de classificação automática necessitam saber quando um exemplo não deve ser classificado como alguma das classes treinadas.

Explicamos abaixo como resolver este problema para o caso específico de classificação automática de novos exemplos de poses.

4.3.1 Classificador de Distâncias Médias Mínimas (DMM)

Primeiramente, sejam $\mathcal{T}_X = \{x_1, \ldots, x_n\}$ o conjunto dos n exemplos de poses treinadas, $X_c = \{x_1, \ldots, x_{|X_c|}\} \subset \mathcal{T}_X$ o subconjunto de exemplos rotulados com a classe c e $X_{\mathcal{C}} = \{X_1, \ldots, X_m\}$.

Definimos a função de distância média $\delta : \mathbb{R}^d \times \mathcal{X} \to \mathbb{R}_+$, entre um novo exemplo de pose $x \in \mathbb{R}^d$ e a classe de poses c, por

$$\delta(x, X_c) = \frac{1}{|X_c|} \sum_{x_i \in X_c} d_G(x, x_i), \tag{4.31}$$

onde $d_G : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$ é a distância de Mahalanobis ótima entre dois exemplos de poses do espaço de características $\mathbb{R}^d$, definida pela Equação 4.28.

O classificador proposto, que denominamos por **Classificador de Distâncias Médias Mínimas (DMM)**, é a função $\rho : \mathbb{R}^d \to \mathcal{C} \cup \{-1\}$ definida por

$$\rho(x) = \begin{cases} c = \underset{y \in \mathcal{C}}{\operatorname{argmin}}\ \delta(x, X_y) & \text{se } \ \delta(x, X_c) < \varepsilon_c, \\ -1 & \text{caso contrário} \end{cases} \tag{4.32}$$

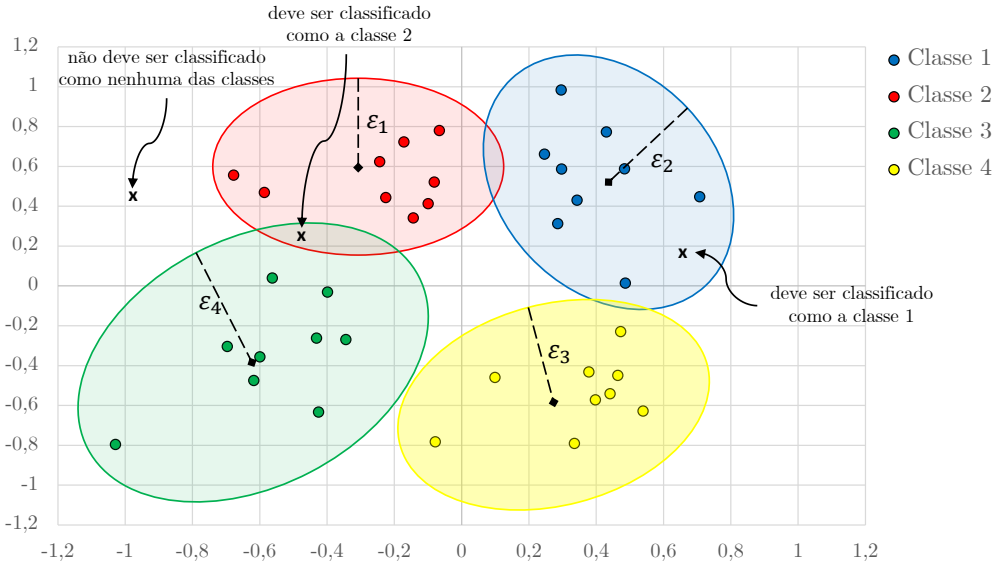
onde ε_c é o limiar de fronteira da classe de poses c mais semelhante ao novo exemplo.

Este classificador recebe um novo exemplo de pose $x \in \mathbb{R}^d$ de rótulo de classe desconhecido, e utilizando a função de distância média definida pela Equação 4.31, classifica o exemplo com o rótulo da classe que o exemplo mais se assemelha. Caso o exemplo não satisfaça a condição do limiar de fronteira da classe, o classificador retorna -1 para o exemplo. A condição de classificação imposta no classificador pelo uso do limiar de fronteira ε_c garante que o exemplo de pose realizado seja classificado como a classe c apenas caso o exemplo esteja o suficientemente próximo da classe mais semelhante.

Na Figura 20, por exemplo, cada limiar de fronteira estipula a distância média máxima permitida para que um novo exemplo de pose seja classificado como pertencentes a

classe. Isto significa que exemplos de poses que não pertençam ao interior de nenhuma das regiões definidas pelos limiares das classes não devem ser classificadas como pertencentes à nenhuma das classes treinadas.

Figura 20 – Ilustração de limiares de fronteiras para 4 classes de exemplos no espaço de características $\mathbb{R}^2$, e de como 3 novos exemplos (marcados com um X) deveriam ser classificados de acordo a condição imposta pelos limiares de fronteira na função de classificação.



Fonte – Autor, 2017.

4.3.2 Estimativa de Limiar de Fronteira de Classes de Poses

Seja $\mathcal{V}_X = \{x_1, \ldots, x_r\}$ o conjunto de poses de validação. Para cada classe de poses c, definimos o subconjunto de poses de validação $\mathcal{V}_c \subset \mathcal{V}_X$ cujos elementos têm rótulo c.

Com estes subconjuntos estimamos um limiar de fronteira ótimo $\varepsilon_c > 0$ para cada classe de poses. Para isto, consideramos primeiramente as funções de custo

$$\mathcal{L}_c(\varepsilon) = \sum_{x_j \in \mathcal{V}_c} [\varepsilon - \delta(x_j, X_c)]_+ [\delta(x_j, X_{c'}) - \varepsilon]_+, \quad c = 1, \ldots, m \tag{4.33}$$

onde $[*]_+ = max(*, 0)$, $c = \underset{y \in \mathcal{C}}{\text{argmin}}$ e $c' = \underset{y \in \mathcal{C}\setminus\{c\}}{\text{argmin}}$ é o rótulo da segunda classe de poses mais próxima de x_j. Em seguida, resolvemos os problemas de otimização correspondentes:

$$\begin{aligned} &\underset{\varepsilon}{\text{maximizar}} && \sum_{x_j \in \mathcal{V}_c} [\varepsilon - \delta(x_j, X_c)]_+ [\delta(x_j, X_{c'}) - \varepsilon]_+ \\ &\text{sujeito a} && \varepsilon > 0, \end{aligned} \tag{4.34}$$

para $c = 1, \ldots, m$.

Cada limiar ótimo (aquele que maximiza a função de custo correspondente), servirá para estimar a distância média máxima permitida para que novos exemplos de poses venham a ser classificadas como pertencentes a uma classe.

Observe que

$$\mathcal{L}_c(\varepsilon) = \begin{cases} \sum_{x_j \in \mathcal{V}_c} [\varepsilon - \delta(x_j, X_c)][\delta(x_j, X_{c'}) - \varepsilon] & \text{se } \varepsilon \in (\delta_{min}, \delta_{max}), \\ 0 & \text{caso contrário.} \end{cases} \tag{4.35}$$

Dessa forma, vemos que a função de custo $\mathcal{L}_c$ assume seu ponto de máximo no intervalo $(\delta_{min}, \delta_{max})$. Além disso, $\mathcal{L}_c$ é diferenciável neste intervalo, pois, expandido o produto na função de custo, notamos que $\mathcal{L}_c$ é o polinômio

$$\mathcal{L}_c(\varepsilon) = -\sum_{x_j \in \mathcal{V}_c} \varepsilon^2 + \sum_{x_j \in \mathcal{V}_c} [\delta(x_j, X_c) + \delta(x_j, X_{c'})]\varepsilon - \sum_{x_j \in \mathcal{V}_c} \delta(x_j, X_c)\delta(x_j, X_{c'}). \tag{4.36}$$

Logo, sua derivada é

$$\mathcal{L}'_c(\varepsilon) = -2 \sum_{x_j \in \mathcal{V}_c} \varepsilon + \sum_{x_j \in \mathcal{V}_c} [\delta(x_j, X_c) + \delta(x_j, X_{c'})]. \tag{4.37}$$

Sabemos que uma função diferenciável possui máximos ou mínimos locais nos pontos onde sua derivada se anula. Portanto, a função de custo $\mathcal{L}_c$ assume valor máximo no intervalo $(\delta_{min}, \delta_{max})$ no ponto ε_c onde $\mathcal{L}'_c(\varepsilon_c) = 0$, ou seja, em

$$\varepsilon_c = \frac{\sum_{x_j \in \mathcal{V}_c} [\delta(x_j, X_c) + \delta(x_j, X_{c'})]}{2|\mathcal{V}_c|}. \tag{4.38}$$

Intuitivamente, encontrar o limiar que maximiza a função de custo $\mathcal{L}_c$, equivale a encontrar um valor ε_c que maximize o somatório das parcelas $[\varepsilon - \delta(x_j, X_c)][\delta(x_j, X_{c'}) - \varepsilon]$ para cada $x_j \in \mathcal{V}_c$. Se $|\mathcal{V}_c| = 1$, então o limiar ótimo ocorre quando $\varepsilon - \delta(x_j, X_c) = \delta(x_j, X_{c'}) - \varepsilon$. Logo, o limiar de fronteira seria simplesmente o valor médio das distâncias $\delta(x_j, X_c)$ $\delta(x_j, X_{c'})$, ou seja,

$$\varepsilon_c = \frac{\delta(x_j, X_c) + \delta(x_j, X_{c'})}{2}. \tag{4.39}$$

Quando $|\mathcal{V}_c| > 1$, então o limiar de fronteira ótimo é o valor médio das distâncias médias

$$\delta_{med,c} = \frac{1}{|\mathcal{V}_c|} \sum_{x_j \in \mathcal{V}_c} \delta(x_j, X_c) \tag{4.40}$$

e

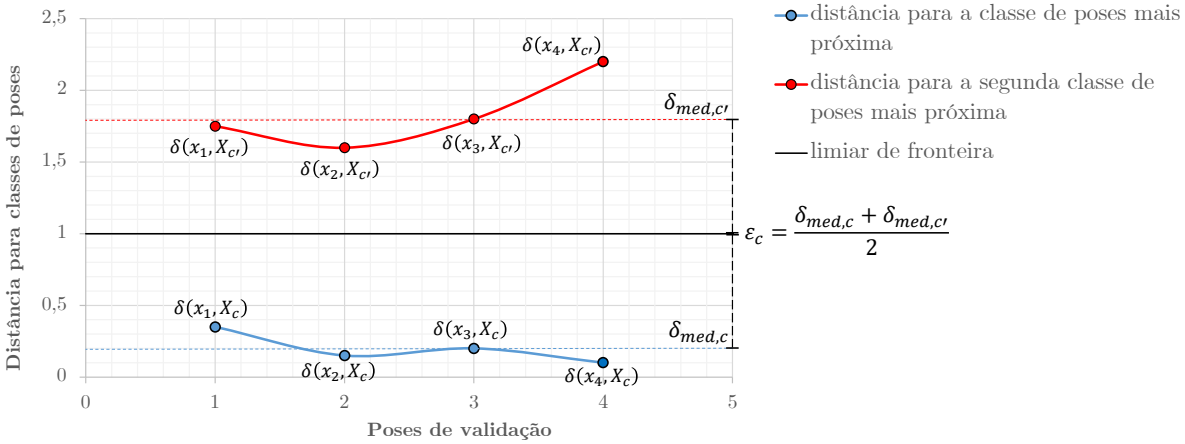
$$\delta_{med,c'} = \frac{1}{|\mathcal{V}_c|} \sum_{x_j \in \mathcal{V}_c} \delta(x_j, X_{c'}), \tag{4.41}$$

das poses de validação $x_j \in \mathcal{V}_c$ para as classes de poses mais próxima (classe c) e segunda mais próxima (classe c'), respectivamente. Ou seja,

$$\varepsilon_c = \frac{\delta_{med,c} + \delta_{med,c'}}{2}, \tag{4.42}$$

como ilustra a Figura 21. Observe que a equação acima é apenas uma outra forma de reescrever o valor de ε_c, descrito na Equação 4.38, utilizando os valores $\delta_{med,c}$ e $\delta_{med,c'}$.

Figura 21 – Ilustração da intuição por trás do cálculo do limiar de fronteira ε_c.



Fonte – Autor, 2017.

4.4 Treinamento de Gestos

Passamos agora à etapa de treinamento de gestos dinâmicos da mão. Assim como na etapa de treinamento de poses, aqui também é passado para a máquina um conjunto de treinamento supervisionado. A diferença é que ao invés de serem passados vetores de descritores de poses, agora são passadas para a máquina sequências de vetores de descritores de poses, correspondentes à execução de gestos pré-estabelecidos, juntamente com a informação de rótulo de classe de cada gesto.

Definimos um gesto $\tilde{g}_i$ como a sequência de vetores de descritores

$$\tilde{g}_i = (x_1, x_2, \ldots, x_{p_i}), \tag{4.43}$$

onde cada vetor de descritor de pose $x_t \in \mathbb{R}^d$ corresponde aos descritores de pose de mão extraídos através do sensor Leap Motion no quadro de rastreamento t.

4.4.1 Conjuntos de Treinamento de Gestos

Para a criação do conjunto de treinamento de gestos, o usuário define o conjunto de classes de gestos $\mathcal{F} = (1, 2, \ldots, m_g)$ a serem treinados de acordo com o tipo de categoria de poses estáticas treinada: **invariantes**, **variantes por rotação**, **variantes por translação** e **variantes por rotação e translação**. A categoria de poses treinadas implica diretamente nos tipos de gestos que a máquina poderá reconhecer. Feito isso, o usuário posiciona a mão acima do sensor e realiza diversos exemplos de cada classe de gestos dinâmicos do conjunto $\mathcal{F}$.

Cria-se então o conjunto de treinamento de gestos supervisionados $\mathcal{T}_{\tilde{G}} = \{(\tilde{g}_i, h_i) \in \tilde{G} \times \mathcal{F};\ i = 1, \ldots, n_g\}$, onde $\tilde{G}$ é o conjunto dos gestos $\tilde{g}_i$ treinados, $h_i \in \mathcal{F}$ é o rótulo de classe do gesto $\tilde{g}_i$ e n_g é quantidade de exemplos treinados.

4.4.2 Extração de Key Poses de Gestos

Cada elemento do conjunto de treinamento de gestos, criado na etapa anterior, é uma sequência de vetores de descritores de poses que descrevem o gesto treinado a cada quadro rastreado pelo sensor Leap Motion (Figura 22, quadro 1). No entanto, visto que um gesto dinâmico, em geral, pode ser perfeitamente caracterizado por uma pequena quantidade de poses estáticas, chamadas de *key poses*, então, utilizando o método de reconhecimento de poses proposto neste trabalho, extraimos de cada gesto dinâmico da mão uma pequena sequência de *key poses* que melhor caracterizam o gesto treinado. Observe, no entanto, que tais poses devem ser treinadas previamente para possibilitar a extração das mesmas de cada gesto. Dessa forma, é necessário que o usuário defina previamente um conjunto de *key poses* que seja capaz de representar os gestos que o usuário deseja treinar de maneira compacta e garantindo que gestos distintos serão representados por sequências distintas. Em seguida, o usuário deve realizar o treinamento destas utilizando o método proposto neste trabalho.

Seja $\mathcal{K} = \{1, 2, \ldots, |\mathcal{K}|\}$ o conjunto de classes de *key poses* treinadas. Para uma melhor compreensão de como é feita a extração de *key poses*, considere o gesto

$$\tilde{g}_i = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}),\ x_j \in \mathbb{R}^d. \tag{4.44}$$

Primeiramente, realizamos o reconhecimento de pose em cada vetor de descritor $x_j \in \mathbb{R}^d$ do gesto $\tilde{g}_i$, obtendo uma sequência de *key poses* $k_j \in \mathcal{K}$ (Figura 22, quadro 2). Digamos que esta sequência seja $(k_1, k_1, -1, -1, k_2, k_3, k_3, k_4, k_5, k_5)$, os -1's presentes indicam que os vetores de descritores x_3 e x_4 não correspondem a nenhuma das classes de *key poses* treinadas. O segundo passo é fazer a simplificação desta sequência (Figura 22, quadro 3). Para isto, removemos os -1's que aparecem; depois, removemos blocos de *key poses* repetidas com menos de M poses, pois estes podem corresponder a ruídos do sensor (M é um parâmetro calibrado manualmente); e, por último, removemos o excesso de *key poses* dos blocos de *key poses* repetidas sucessivamente e que possuem mais de M poses, deixando apenas uma *key pose* representante por bloco. Por exemplo, com $M = 2$, após esta simplificação, a sequência obtida anteriormente passa a ser apenas $g_i = (k_1, k_3, k_5),\ k_j \in \mathcal{K}$. Esta filtragem também será realizada na etapa de reconhecimento de gestos.

Finalizada a extração de *key poses* para todos os gestos treinados, cria-se um novo conjunto de treinamento de gestos simplificados $\mathcal{T}_G = \{(g_i, h_i) \in G \times \mathcal{F};\ i = 1, \ldots, n_g\}$, onde G é o conjunto dos gestos simplificados, $h_i \in \mathcal{F}$ é o rótulo de classe do gesto g_i e n_g é quantidade de gestos treinados (Figura 22, quadro 4).

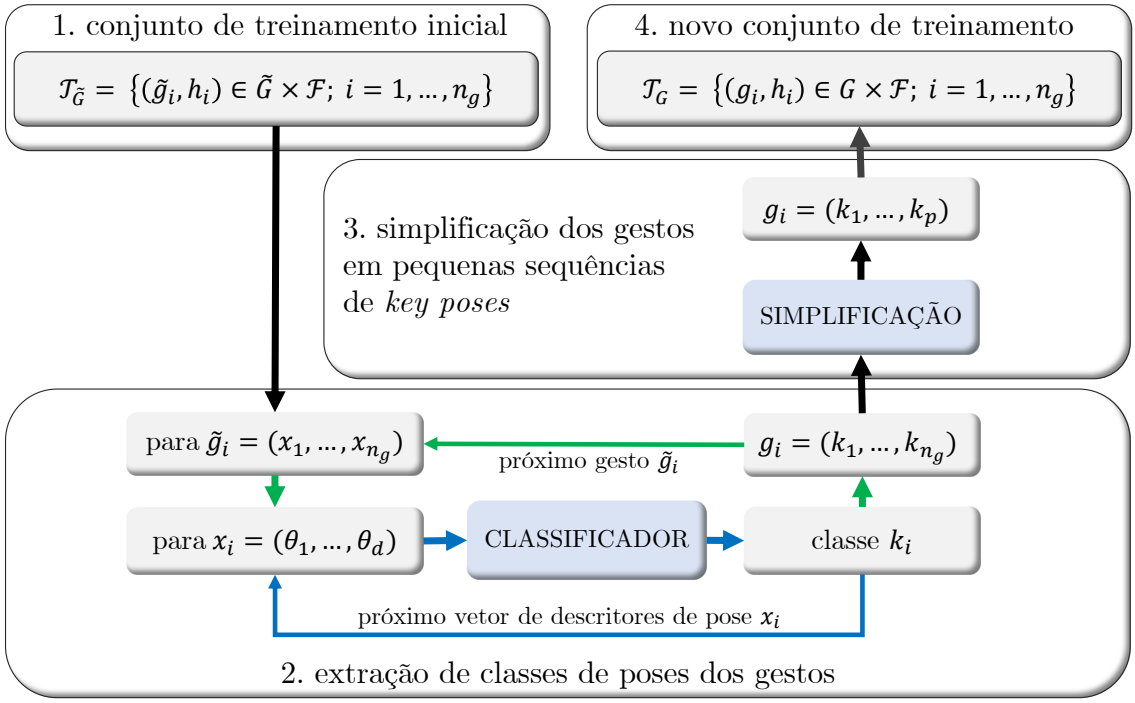
Na próxima seção veremos como funcionam os grafos de ação e como codificar o conjunto de treinamento de gestos $\mathcal{T}_G$ nessas estruturas.

4.4.3 Codificação em Grafos de Ação

Um grafo de ação é uma estrutura utilizada para armazenar informações de probabilidades de transições entre *key poses* referentes a uma classe de gestos. Cada grafo de ação A_h pode ser visto como um grafo orientado ponderado, onde: os nós correspondem às *key poses* do conjunto $\mathcal{K}$; as arestas orientadas e_{ij} indicam a transição entre as *key poses* k_i e k_j; e os pesos são as probabilidades de transições $p_h(k_j|k_i)$, que mede a probabilidade de ocorrer a transição da *key pose* k_i para k_j em cada classe de gestos h.

Para codificar o conjunto de treinamento de gestos simplificados $\mathcal{T}_G = \{(g_i, h_i) \in G \times \mathcal{F};\ i = 1, \ldots, n_g\}$, é criado um grafo de ação para cada classe de gestos treinada. Cada grafo é então utilizado para codificar as probabilidades de transições $p_h(k_j|k_i)$ entre as *key poses* que ocorrem na classe de gestos correspondente. De acordo com trabalhos

Figura 22 – Ilustração das etapas de simplificação dos gestos do conjunto de treinamento $\mathcal{T}_{\tilde{G}}$ em pequenas sequências de *key poses* para a criação de um novo conjunto de treinamento de gestos simplificados $\mathcal{T}_G$.



Fonte – Autor, 2017.

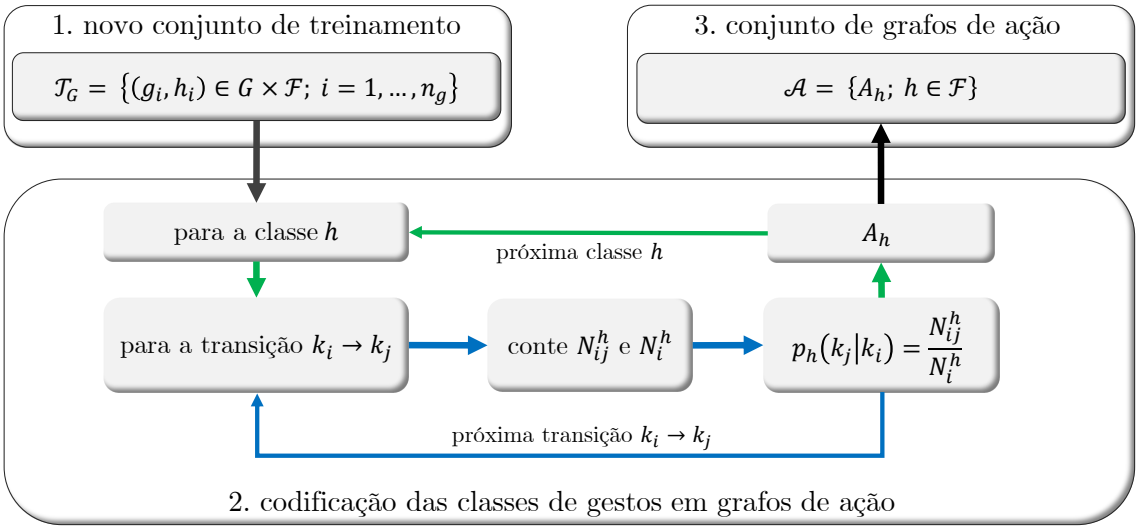
anteriores [13, 45] estas probabilidades podem ser

$$p_h(k_j|k_i) = \frac{N_{ij}^h}{N_i^h}, \tag{4.45}$$

onde N_{ij}^h é quantidade de vezes que ocorre a transição da *key pose* k_i para k_j na classe de gestos h e N_i^h é a quantidade total de transições da *key pose* k_i para quaisquer outras *key poses* que ocorrem nos exemplos da classe de gestos h do conjunto de treinamento de gestos simplificados. Fazendo isto para todas as classes de gestos, teremos o conjunto de treinamento de gestos inicial $\mathcal{T}_{\tilde{G}}$ codificado como o conjunto de grafos de ação $\mathcal{A} = \{A_h;\ h \in \mathcal{F}\}$. A Figura 23 ilusta o processo de codificação das classes de gestos do conjunto de treinamento $\mathcal{T}_G$ no conjunto de grafos de ação $\mathcal{A}$.

Para uma melhor compreensão, suponha, por exemplo, que um certo conjunto de treinamento tenha uma classe de gestos h que, após a etapa de extração de *key poses*, retornou as seguintes sequências para cada exemplo de gesto da classe: $g_{h_1} = (k_1, k_3, k_1)$, $g_{h_2} = (k_1, k_2, k_3, k_1)$, $g_{h_3} = (k_1, k_3, k_4, k_1)$, $g_{h_4} = (k_1, k_3, k_1)$ e $g_{h_5} = (k_1, k_3, k_4, k_1)$. Suponha também que o conjunto de classes de poses estáticas treinadas seja $\mathcal{C} = \{k_1, k_2, k_3, k_4\}$.

Figura 23 – Ilustração da etapa de codificação das classes de gestos treinadas em grafos de ação.



Fonte – Autor, 2017.

Para criar o grafo de ação corresponde a classe de gestos h, calculamos as probabilidades de transições entre *key poses* considerando todas as transições presentes nestas sequências.

Para a transição entre k_1 e k_2, por exemplo, temos que $N_{12}^h = 1$, pois entre as sequências $g_{h_1}, g_{h_2}, g_{h_3}$, g_{h_4} e g_{h_5}, a transição k_1 para k_2 aparece apenas 1 vezes; e $N_1^h = 5$, pois, existem 5 transições entre *key poses* começando em k_1. Logo, $p_h(k_2|k_1) = 1/5$. Analogamente, podemos ver que: $p_h(k_3|k_1) = 4/5$, $p_h(k_3|k_2) = 1$, $p_h(k_1|k_3) = 3/5$, $p_h(k_4|k_3) = 2/5$ e $p_h(k_1|k_4) = 1$. Todas as demais probabilidades de transições entre *key poses* da classe de gesto h são nulas. A Figura 24 ilustra o grafo de ação para este exemplo em particular.

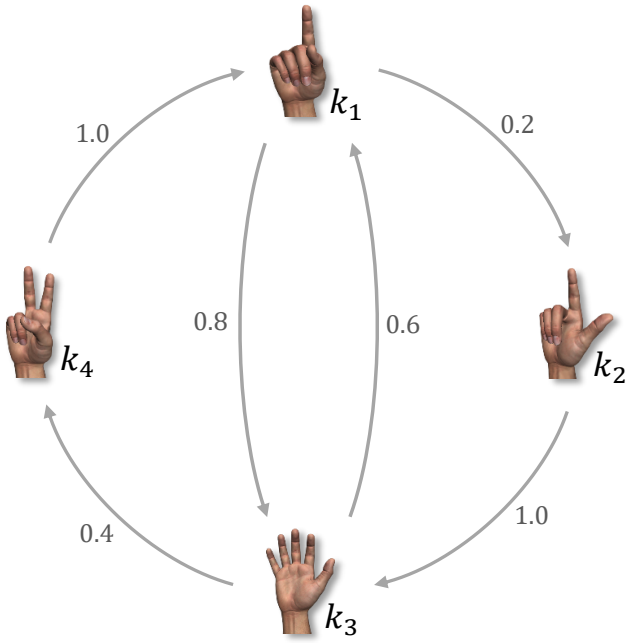
Na próxima seção veremos como é feito o reconhecimento de novos exemplos de gestos utilizando estes grafos de ação e de como esse tipo de codificação possibilita o reconhecimento *offline* e *online* de gestos.

4.5 Reconhecimento de Gestos

Tendo em vista que um gesto dinâmico da mão é caracterizado como uma sequência de vetores de descritores de pose observada ao longo do tempo, utilizamos o HMM (Seção 3.5), para realizar o reconhecimento de gestos dinâmicos da mão.

Nesta formulação, as *observações* correspondem aos vetores de descritores de pose

Figura 24 – Grafo de ação com probabilidades de transições entre 4 *key poses* da mão.



Fonte – Autor, 2017.

da mão (extraídos do sensor Leap Motion a cada quadro), e os *estados ocultos* do sistema correspondem às *key poses* pela qual um gesto executado transita ao longo de sua realização. No entanto, uma vez que temos um classificador de poses estáticas, podemos utilizá-lo para descobrir os *estados ocultos* do modelo, isto é, as *key poses* pela qual um gesto realizado transita ao longo do tempo. A partir desta observação, uma HMM que governa a realização de um gesto passa a ser praticamente uma Cadeia de Markov, onde os estados são as *key poses* do modelo.

Com esta formulação é possível efetuar o reconhecimento de gestos dinâmicos em dois cenários de reconhecimento bem distintos: o *offline*, onde os exemplos de gestos são realizados de forma segmentada e o reconhecimento só é ativado quando a realização do gesto é concluída; e o *online*, onde o reconhecimento é feito em tempo real, isto é, durante a realização do gesto, e o usuário não precisa segmentar cada exemplo.

4.5.1 Cenário de Reconhecimento Offline

Assim como na fase de treinamento, para o reconhecimento de gestos *offline*, o usuário realiza os exemplos de gestos dinâmicos de forma bem segmentada, isto é, com início e fim bem determinados. Em seguida, os gestos realizados passam pela etapa de

extração de *key poses*, exatamente como descrito na Seção 4.1.1. Finalizada a extração, cada exemplo é passado para a etapa de decodificação de sua sequência de *key poses* correspondente. Nesta etapa de decodificação é feita a classificação final do exemplo como uma das classes de gestos treinadas previamente.

Para isto, é utilizado um decodificador de gestos baseado na probabilidade de ocorrer uma sequência de estados na Cadeia de Markov de uma determinada classe de gestos. Este decodificador utiliza os grafos de ação de cada classe de gesto treinada para calcular a probabilidade da sequência de *key poses* extraídas ter sido gerada a partir de cada classe de gesto treinada previamente. Fazendo um paralelo com Cadeias de Markov, observe que cada grafo de ação seria o equivalente a uma estimativa da matriz de transição de probabilidades da Cadeia de Markov que governa as propriedades de transições entre *key poses* da classe de gesto correspondente.

É válido utilizar esta formulação de Cadeias de Markov porque de acordo com os grafos de ação criados na etapa de treinamento, a *key pose* corrente de um gesto que está sendo realizado depende apenas da *key pose* que ocorre anteriormente no gesto, ou seja, a realização de um gesto dinâmico satisfaz a **propriedade de Markov**. Dessa forma, podemos utilizar a Equação 3.25 para calcular a probabilidade da sequência de *key poses* ter sido gerada por cada classe de gestos treinada.

Portanto, se um gesto $\tilde{g}$ realizado gerou uma sequência de *key poses* $\{k_1, k_2, \ldots, k_p\}$, então a classe de gestos treinada que mais se assemelha ao gesto $\tilde{g}$ é

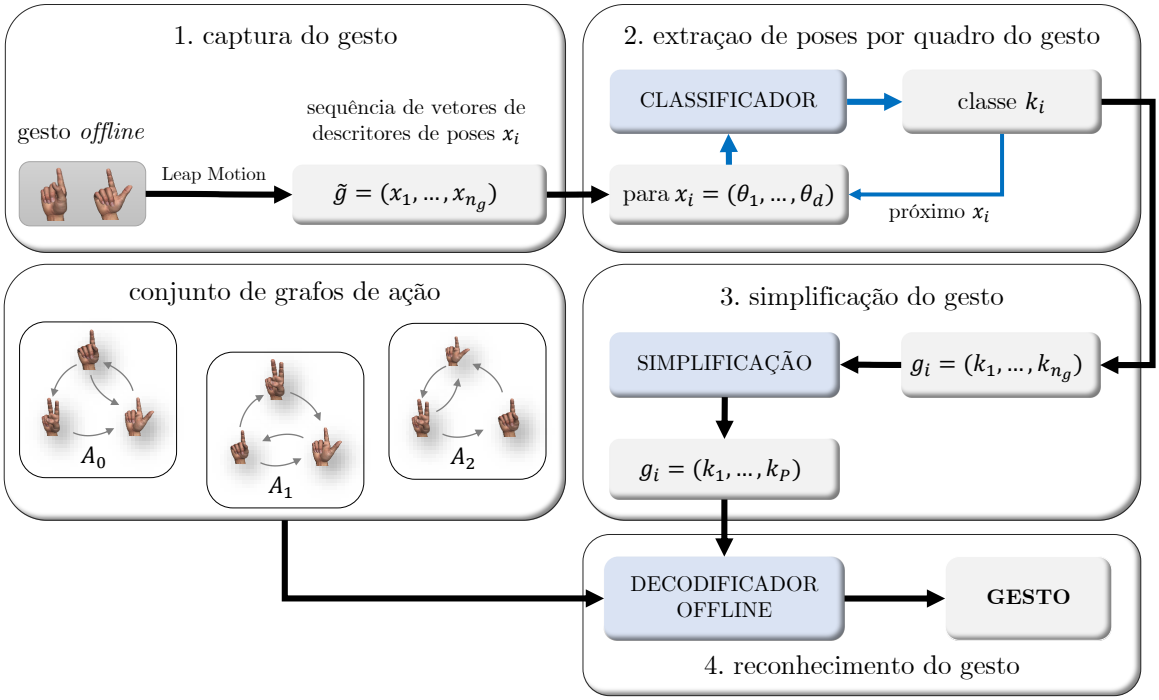
$$g' = \underset{i \in \mathcal{F}}{\operatorname{argmax}} \prod_{t=2}^{p} p_i(k_t|k_{t-1}), \tag{4.46}$$

onde $p_i(k_t|k_{t-1})$ é a probabilidade de um gesto da classe de gestos i realizar a transição de *key poses* k_{t-1} para k_t.

4.5.2 Cenário de Reconhecimento Online

Diferentemente do cenário de reconhecimento de gestos *offline*, aqui o usuário é livre para executar os exemplos sem a necessidade de pausa entre gestos ou qualquer tipo de segmentação inicial e final dos exemplos, sendo ideal para interfaces naturais de usuário, por exemplo. Neste cenário, o reconhecimento de gestos dinâmicos é feito em tempo real, isto é, de forma simultânea à realização dos exemplos.

Figura 25 – Ilustração das etapas do reconhecimento de gestos *offline*.



Fonte – Autor, 2017.

Para isto, vamos utilizar uma estratégia apresentada em [13], na qual o reconhecimento é ativado a cada nova *key pose* detectada durante a realização do gesto. Este processo funciona da seguinte forma: para cada vetor de descritores de pose estática $x \in \mathbb{R}^d$ obtidos do sensor Leap Motion (Figura 26, quadro 1), o classificador de poses reconhece a classe de pose estática correspondente (Figura 26, quadro 2) e insere-a em um *buffer* de *key poses* (Figura 26, quadro 3), obedecendo os mesmos critérios de geração das sequências de *key poses* visto na Seção 4.1.1, ou seja, este *buffer* deve conter as poses estáticas mais relevantes detectadas na realização do gesto até o instante corrente, sem que hajam *key poses* sequenciais repetidas. A cada *key pose* inserida no *buffer*, é ativado o decodificador de gestos (Figura 26, quadro 4), que calcula as probabilidades de cada classe de gestos ter gerado as sequências das últimas *key poses* inseridas no *buffer*.

Para esclarecer melhor as ideias, suponha que num dado instante de tempo foi inserido no *buffer* a *key pose* k_r e o *buffer* tornou-se a sequência de *key poses* $\{k_1, k_2, \ldots, k_{r-1}, k_r\}$. Ao ser inserida esta *key pose*, o decodificador de gestos é ativado. Este decodificador analisa as probabilidades das subsequências $\{k_{r-q-1}, \ldots, k_{r-1}, k_r\}$ de *key poses* do *buffer*, terem sido geradas a partir de cada classe de gestos treinadas, iniciando-se

de $q = 0$, e incrementando-o se necessário. A análise é interrompida quando é encontrado um q tal que a sequência de *key poses* corresponde tenha sido gerada por uma classe de gestos com probabilidade suficientemente maior do que a probabilidade da segunda classe de gestos mais provável. Isto significa que a subsequência $\{k_{r-q-1}, \ldots, k_{r-1}, k_r\}$ é suficiente para distinguir o gesto que foi executado por último.

Formalmente, suponha que em um determinado momento da realização de um gesto o *buffer* de *key poses* seja $\{k_1, k_2, \ldots, k_{r-1}, k_r\}$, e sejam

$$L_i(q) = \prod_{t=r-q}^{r} p_i(k_t|k_{t-1}),\ i \in \mathcal{F}, \tag{4.47}$$

as probabilidades da subsequência $\{k_{r-q-1}, \ldots, k_{r-1}, k_r\}$, das últimas *key poses* inseridas no *buffer*, ter sido gerada a partir de cada uma das classes de gestos treinadas $i \in \mathcal{F}$. O reconhecimento do gesto corrente é iniciado em $q = 0$ e para se

$$L_{g'}(q) > 0 \text{ e } L_{g''}(q) = 0 \tag{4.48}$$

ou

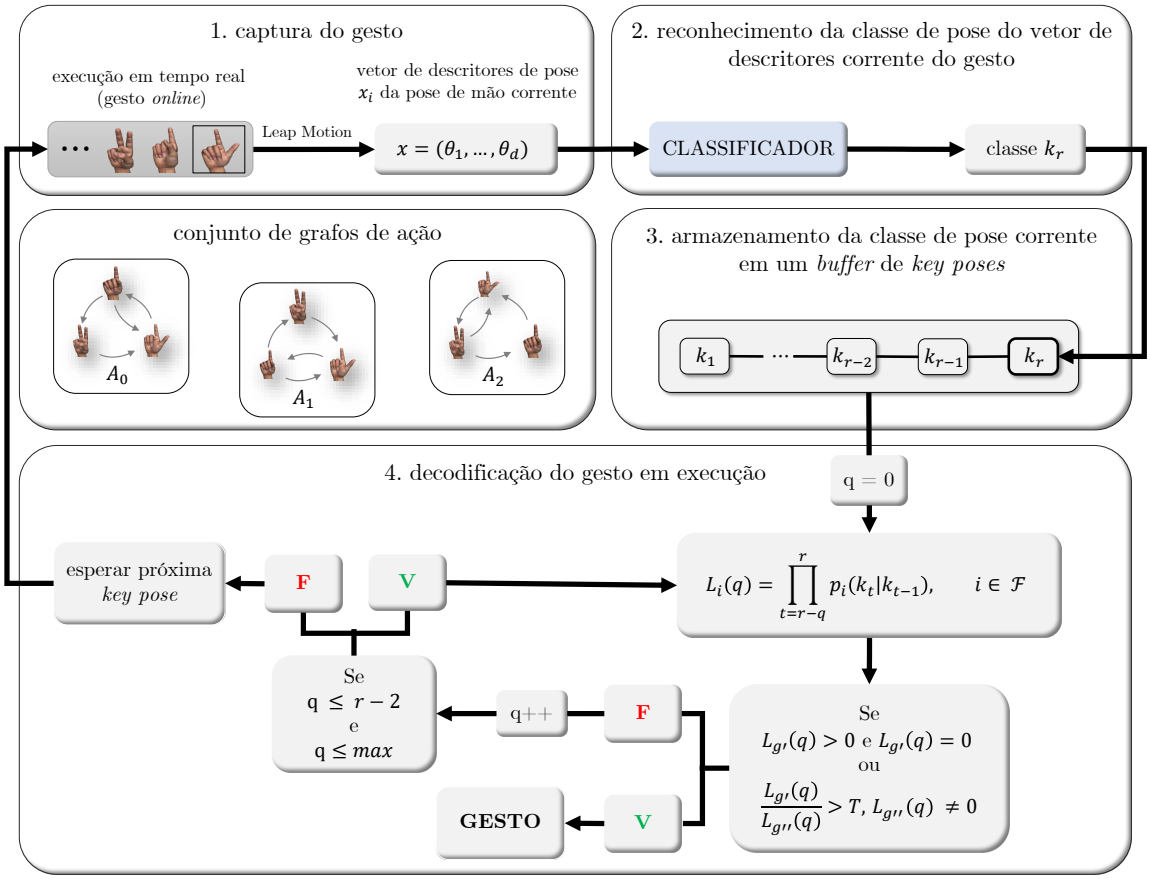
$$\frac{L_{g'}(q)}{L_{g''}(q)} > T,\ L_{g''}(q) \neq 0, \tag{4.49}$$

onde g' e $g'' \in \mathcal{F}$ são a primeira e segunda classes de gestos treinadas mais prováveis de que o gesto atual pertença, respectivamente; o parâmetro T é calibrado empiricamente.

Se uma destas condições de parada for satisfeita, o gesto atualmente sendo realizado é classificado como pertencente a classe de gestos g', mesmo que a realização do gesto não tenha sido ainda concluída. Caso nenhuma das condições de parada seja satisfeita para quaisquer dos valores q, com $q \leq r - 2$ e $q \leq max$; então, o decodificador de gestos aguarda uma nova *key pose* ser inserida no *buffer* e repete o mesmo processo com o *buffer* de *key poses* atualizado. O parâmetro max é utilizado para limitar a profundidade de busca por *key poses*, uma vez que na maioria das aplicações os gestos *online* são determinados por pequenas sequências de *key poses*, o que torna desnecessária a busca por *key poses* muito antigas do *buffer*. O quadro 4 da Figura 26 ilustra o funcionamento deste algoritmo.

Todo este processo é feito em tempo real, permitindo a previsão antecipada de gestos; além disso, deixa o usuário livre para realizar os gestos sem a necessidade de pausas ou qualquer outro tipo de segmentação para que a máquina seja capaz de reconhecê-los eficientemente.

Figura 26 – Ilustração das etapas do reconhecimento de gestos *online*.



Fonte – Autor, 2017.

5 RESULTADOS E APLICAÇÕES

Para gerar os experimentos deste capítulo, foi desenvolvido um software na linguagem de programação C++ com a biblioteca OpenGL para visualização do esqueleto da mão rastreado pelo sensor Leap Motion. Foi utilizado ainda, a parte, o software Matlab para a etapa de aprendizagem das métricas, tanto no caso linear quanto no não linear.

Foram criadas 4 aplicações distintas, uma para cada categoria de poses estáticas consideradas (ver Subseção 4.1.1). Em cada uma delas foi feita a análise de acurácia nas tarefas: reconhecimento de poses estáticas, reconhecimento de gestos dinâmicos de forma *offline* e reconhecimento de gestos dinâmicos de forma *online*.

Estas aplicações englobaram: reconhecimento de gestos dinâmicos para interfaces naturais de usuário; reconhecimento de sinais dos números em LIBRAS e operações aritméticas; e reconhecimento de sinais do alfabeto manual em LIBRAS de forma estática ou dinâmica. Descrevemos os resultados obtidos nas próximas seções.

5.1 Reconhecimento de Gestos Dinâmicos Para Interfaces Naturais de Usuário (NUIs)

Em computação, *Interface Natural de Usuário* (NUI) é um paradigma de interação humano-máquina onde as interações são realizadas a partir de ações naturais dos seres humanos, sem a necessidade de periféricos intermediários para interagir com a máquina, ou caso existam, que estes estejam em segundo plano. Pode-se interagir com as NUIs de diferentes formas, como por exemplo, utilizando toques, comandos de voz, comandos gestuais, entre outros. E é nesta última modadidade citada que propomos uma biblioteca de gestos dinâmicos simples para controle de uma NUI por comandos gestuais da mão utilizando o sensor Leap Motion.

Para isto, capturamos uma base de dados composta por um total de 7 classes de gestos dinâmicos da mão (Tabela 1). Para possibilitar o reconhecimento destes gestos, nossa base de dados contém um conjunto de poses estáticas **variantes por translação** composto de 8 poses estáticas bem definidas (Figura 27).

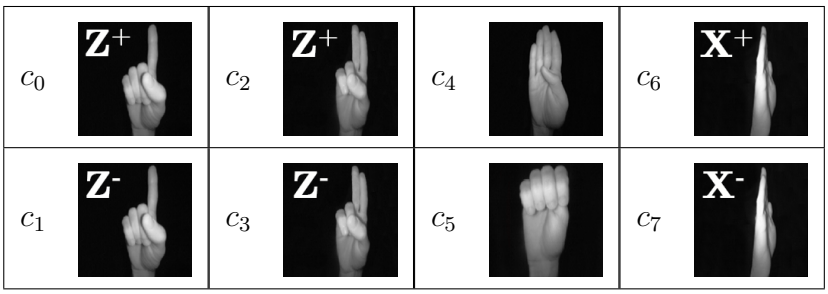
Classes marcadas com letra e sinal indicam a localização espacial das poses da classe em relação ao sensor Leap Motion (ver sistema de coordenadas do Leap Motion na Figura 6). Por exemplo, Z^+ indica que a classe está localizada no semi-espaço $z > 0$ (à

Tabela 1 – Descrição do conjunto de gestos dinâmicos da mão para utilização de comandos gestuais em NUIs.

Classe	**Gesto**	**Descrição**
g_0	SELECIONAR	Gesto utilizado para selecionar itens em um determinado contexto ou inicializar determinadas tarefas. Ação análoga ao clique simples do botão esquerdo do mouse.
g_1	EXECUTAR	Gesto utilizado para abrir ou executar itens selecionados em um determinado contexto ou inicializar determinadas tarefas. Ação análoga ao clique duplo do botão esquerdo do mouse.
g_2	OPÇÕES	Gesto utilizado para visualizar opções em um determinado contexto. Ação análoga ao clique simples do botão direito do mouse.
g_3	ESQUERDA	Gesto utilizado para alternar entre itens a esquerda em um determinado contexto. Análogo a seta para esquerda do teclado.
g_4	DIREITA	Gesto utilizado para alternar entre itens a direita em um determinado contexto. Análogo a seta para direita do teclado.
g_5	CIMA	Gesto utilizado para alternar entre itens a cima em um determinado contexto. Análogo a seta para cima do teclado.
g_6	BAIXO	Gesto utilizado para alternar entre itens a baixo em um determinado contexto. Análogo a seta para baixo do teclado.

Fonte – Autor, 2017.

Figura 27 – Conjunto de classes de poses estáticas variantes a translação utilizados para realizar comandos gestuais em NUIs.



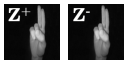
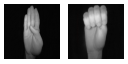
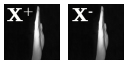
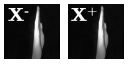
Fonte – Autor, 2017.

frente do sensor); analogamente, Z^- indica que a classe está localizada no semi-espaço $z < 0$ (atrás do sensor), X^+ no semi-espaço $x > 0$ (à direita do sensor) e X^- no $x < 0$ (à esquerda do sensor). Finalmente, definimos os gestos da nossa biblioteca de gestos dinâmicos como mostra a Tabela 2.

Para esta aplicação, criamos:

1. um conjunto de treinamento de classes de poses estáticas composto de 110 exemplos

Tabela 2 – Gestos dinâmicos da mão para reconhecimento de comandos gestuais em NUIs.

Classe	Key Poses	Representação Visual
g_0	c_0 c_1	
g_1	c_2 c_3	
g_2	c_4 c_5	
g_3	c_6 c_7	
g_4	c_6 c_7	
g_5	c_5 c_1 ou c_5 c_0	
g_6	c_2 c_5 ou c_3 c_5	

Fonte – Autor, 2017.

distribuídos entre as 8 classes de poses, sendo 67 para treinamento e 43 para validação;

2. um conjunto de treinamento de gestos dinâmicos composto de 99 exemplos de gestos distribuídos entre as 7 classes de gestos consideradas.

O conjunto de treinamento de poses estáticas foi então passado como entrada dos algoritmos LMNN e KLMNN para obter as melhores distâncias de mahalanobis linear e não linear, respectivamente, para a aplicação.

5.1.1 Análise do Reconhecimento de Poses Estáticas

Para análise de acurácia das classes de poses estáticas, criamos um conjuntos de testes com 80 exemplos de poses (10 por classe). Para o reconhecimento destas poses utilizamos o Classificador DMM (Seção 4.3.1), com d_G igual as 3 distâncias: **euclidiana**, **mahalanobis linear**, e **mahalanobis não linear**. Para a aprendizagem da distância de mahalanobis não linear foi utilizado o parâmetro argumento do *kernel gaussiano* $\sigma = 0.5$, calibrado manualmente.

Para as 3 distâncias utilizadas, o algoritmo classificou corretamente todas as 70 poses do conjunto de testes (acurácia de 100.0%). Tais índices altíssimos devem-se principalmente ao conjunto de classes de poses ser relativamente simples, não existindo

classes de poses com sobreposições de dedos ou oclusões que impossibilitem o sensor Leap Motion de fazer um rastreamento de juntas com alta precisão. A Tabela 3 exibe a matriz de confusão correspondente as informações de classifcação para as classes de poses estáticas.

Tabela 3 – Matriz de confusão do reconhecimento de poses estáticas da biblioteca de gestos dinâmicos utilizando as distâncias euclidiana, mahalanobis linear e mahalanobis não linear.

classe	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_D	(%)
c_0	10	0	0	0	0	0	0	0	0	100.0
c_1	0	10	0	0	0	0	0	0	0	100.0
c_2	0	0	10	0	0	0	0	0	0	100.0
c_3	0	0	0	10	0	0	0	0	0	100.0
c_4	0	0	0	0	10	0	0	0	0	100.0
c_5	0	0	0	0	0	10	0	0	0	100.0
c_6	0	0	0	0	0	0	10	0	0	100.0
c_7	0	0	0	0	0	0	0	10	0	100.0

Fonte – Autor, 2017.

Cada elemento a_{ij} representa a quantidade de exemplos da classe de poses i que foram classificados como pertencentes à classe de poses j. A coluna c_D representa as quantidades de exemplos de cada classe i que não foram classificados com pertencentes a nenhuma das classes j. Enquanto a coluna (%) exibe a acurácia de cada classe de poses.

5.1.2 Análise do Reconhecimento de Gestos Dinâmicos Offline

Para análise de acurácia das classes de gestos dinâmicos, criamos um conjunto de treinamento de gestos composto de 70 exemplos de gestos (10 exemplos por classe). Na etapa de extração de *key poses* dos gestos foi utilizado o parâmetro comprimento de blocos de *key poses* sucessivas, descrito na Seção 4.4.2, como $M = 5$. A Tabela 4 exibe os detalhes de reconhecimento por classe de gestos dinâmicos. As taxas de reconhecimento novamente foram máximas utilizando ambas as distâncias.

5.1.3 Análise do Reconhecimento de Gestos Dinâmicos Online

No reconhecimento *online* foram criados diversos conjuntos com execuções de gestos *online*. Em todos estes conjuntos o classificador de gestos *online* foi capaz de reconhecer de forma robusta e antecipada todos os gestos realizados, com uma latência de reconhecimento

Tabela 4 – Desempenho do reconhecimento das classes de gestos da biblioteca de gestos dinâmicos para comandos gestuais em NUIs.

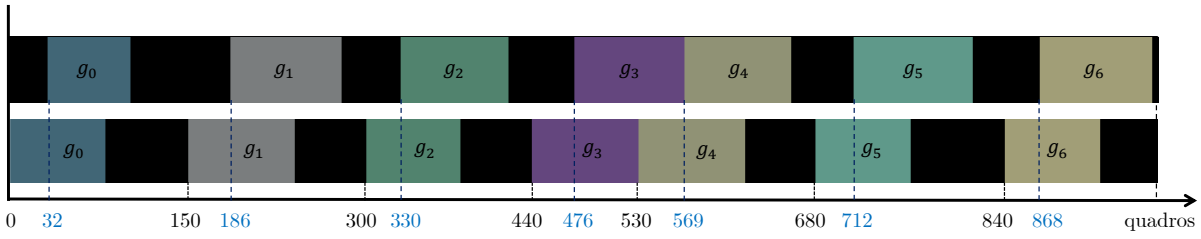
		Euclidiana		**Mahalanobis Linear**		**Mahalanobis Não Linear**	
Classe	Quantidade	Acertos	Acurácia (%)	Acertos	Acurácia (%)	Acertos	Acurácia (%)
g_0	10	10	100.0	10	100.0	10	100.0
g_1	10	10	100.0	10	100.0	10	100.0
g_2	10	10	100.0	10	100.0	10	100.0
g_3	10	10	100.0	10	100.0	10	100.0
g_4	10	10	100.0	10	100.0	10	100.0
g_5	10	10	100.0	10	100.0	10	100.0
g_6	10	10	100.0	10	100.0	10	100.0
Geral	70	70	100.0	70	100.0	70	100.0

Fonte – Autor, 2017.

inferior a 50% da execução do gesto para maioria das classes, ou seja, o gesto é reconhecido antes de sua execução completa.

A Figura 28 ilustra um destes conjuntos, composto pela realização de todas as 7 classes de gestos treinadas, onde foi utilizada a distância de **mahalanobis não linear** para detecção das *key poses*. Faixas pretas indicam que no período correspondente não foi detectado nenhum gesto sendo executado.

Figura 28 – Linhas do tempo de execução e reconhecimento de gestos da biblioteca de gestos dinâmicos para comandos gestuais em NUIs segmentados manualmente (linha do tempo inferior) e segmentados de forma automática pelo algoritmo (linha do tempo superior).



Fonte – Autor, 2017.

Utilizando as outras distâncias: euclidiana e mahalanobis linear, o reconhecimento e segmentação automática dos gestos se deram de forma bastante parecida. A Tabela 5 mostra as taxas de latência do reconhecimento dos gestos do conjunto analisado na Figura 28.

Tabela 5 – Latência do reconhecimento de gestos da biblioteca de gestos dinâmicos para comandos gestuais em NUIs em três abordagens: por quadros, por segundo e por porcentagem de execução do gesto.

Classe	**Latência (quadros)**	**Latência (segundos)**	**Latência** (%)
g_0	32	0.80	40.0
g_1	36	0.90	40.0
g_2	30	0.75	37.5
g_3	36	0.90	40.0
g_4	39	0.97	43.3
g_5	32	0.80	40.0
g_6	28	0.70	35.0

Fonte – Autor, 2017.

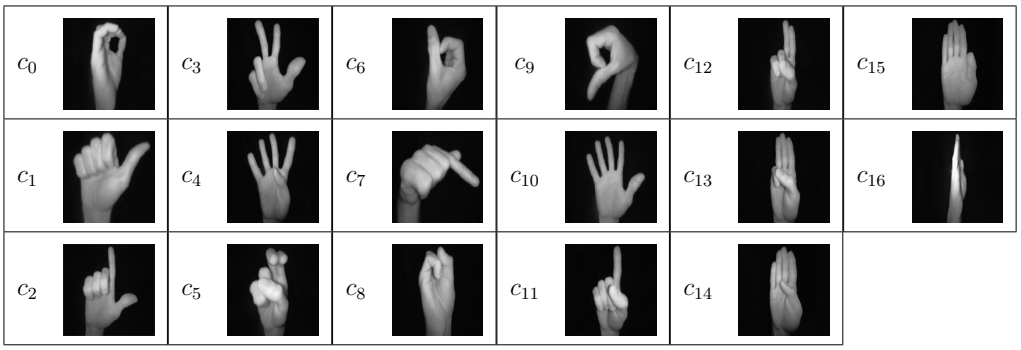
Observamos de modo geral com esta aplicação, que o desempenho do Classificador DMM apenas com a distância euclidiana já foi suficiente para um reconhecimento eficaz tanto de poses estáticas quanto de gestos dinâmicos executados de forma *offline* e *online*.

5.2 Reconhecimento de Sinais dos Números em LIBRAS e Gestos Personalizados Para as Operações Matemáticas Básicas

Nesta segunda aplicação, utilizamos nosso método de reconhecimento de gestos dinâmicos para o reconhecimento dos 10 números na *Língua Brasileira de Sinais* (LIBRAS) e de mais um conjunto de 5 gestos personalizados para representar as 4 operações matemáticas básicas e o operador de igualdade. Para possibilitar um reconhecimento preciso, treinamos o conjunto de *key poses* da Figura 29, composto de 15 poses estáticas **variantes por rotação**; onde: as poses das classe de c_0 a c_9 são os sinais estáticos em LIBRAS que representam os números de 0 a 9; as classes de c_{11} a c_{14} representam as operações matemáticas básicas: adição, subtração, multiplicação e divisão; as classes c_{15} e c_{16} são utilizadas para representar o gesto do operador de igualdade e a classe c_{10} é utilizada para facilitar o reconhecimento *online* dos gestos.

Vale a pena mencionar que durante a confecção desta base inserimos variações propositais nos exemplos de treinamento de cada classe de poses. Fazemos isto para agregar à máquina maior potencial de generalização sobre novas amostras desconhecidas de poses da mão.

Figura 29 – Classes de poses estáticas para o reconhecimento dos números em LIBRAS e gestos personalizados para as operações matemáticas básicas.



Fonte – Autor, 2017.

Finalmente, treinamos os números de 0 a 9, as operações matemáticas básicas e operador de igualdade como as classes de gestos dinâmicos mostrados na Tabela 6.

Utilizamos gestos dinâmicos terminados com a classe c_{10} para representar os números ao invés apenas da representação de cada número como uma pose estática, para possibilitar um reconhecimento *online* eficaz, pois desejamos evitar segmentações manuais, deixando o usuário livre para realizar os gestos. Esta estratégia garante também que números indesejados (fruto de ruídos do sensor ou de transições entre classes de poses) sejam reconhecidos pela máquina acidentalmente.

Com esta estratégia podemos realizar facilmente operações matemáticas básicas utilizando apenas gestos da mão. Por exemplo, se um usuário deseja efetuar a operação $2017 - 1995$, basta que ele realize de forma *online* a seguinte sequência de classes de poses:

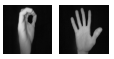
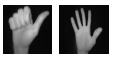
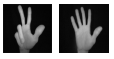
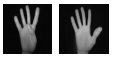
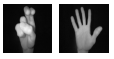
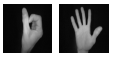
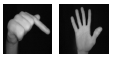
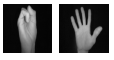
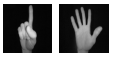
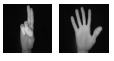
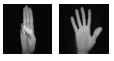
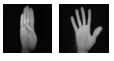
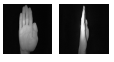
$\underbrace{c_2\ c_{10}}$	$\underbrace{c_0\ c_{10}}$	$\underbrace{c_1\ c_{10}}$	$\underbrace{c_7\ c_{10}}$	$\underbrace{c_{12}\ c_{10}}$	$\underbrace{c_1\ c_{10}}$	$\underbrace{c_9\ c_{10}}$	$\underbrace{c_9\ c_{10}}$	$\underbrace{c_5\ c_{10}}$	$\underbrace{c_{15}\ c_{16}}$
g_2	g_0	g_1	g_7	g_{11}	g_1	g_9	g_9	g_5	g_{14}
2	0	1	7	$-$	1	9	9	5	$=$

Para possibilitar o reconhecimento eficaz de poses e gestos para esta aplicação, criamos:

1. um conjunto de treinamento de classes de poses estáticas composto de 306 exemplos distribuídos entre as 17 classes de poses, sendo 185 para treinamento e 121 para validação;

2. um conjunto de treinamento de gestos dinâmicos composto de 99 exemplos de gestos

Tabela 6 – Gestos dinâmicos da mão para representação dos números em LIBRAS e as operações matemáticas básicas.

Classe	**Gesto**	***Key Poses***	**Representação Visual**
g_0	0	c_0 c_{10}	
g_1	1	c_1 c_{10}	
g_2	2	c_2 c_{10}	
g_3	3	c_3 c_{10}	
g_4	4	c_4 c_{10}	
g_5	5	c_5 c_{10}	
g_6	6	c_6 c_{10}	
g_7	7	c_7 c_{10}	
g_8	8	c_8 c_{10}	
g_9	9	c_9 c_{10}	
g_{10}	+	c_{11} c_{10}	
g_{11}	−	c_{12} c_{10}	
g_{12}	×	c_{13} c_{10}	
g_{13}	÷	c_{14} c_{10}	
g_{14}	=	c_{15} c_{16}	

Fonte – Autor, 2017.

distribuídos entre as 15 classes de gestos consideradas.

O conjunto de treinamento de poses estáticas foi então passado como entrada dos algoritmos LMNN e KLMNN para obter as melhores distâncias de mahalanobis linear e não linear, respectivamente, para a aplicação.

5.2.1 Análise do Reconhecimento de Poses Estáticas

Para análise de acurária das classes de poses estáticas, criamos um conjunto de testes de poses estáticas com 170 exemplos de poses (10 por classe). Abaixo exibimos os

resultados de classificação utilizando as 3 distâncias consideradas no Classificador DMM: **euclidiana**, **mahalanobis linear** e **mahalanobis não linear**.

No geral foram obtidos os seguintes resultados:

- com a distância euclidiana o reconhecimento foi de 94.7% (161 exemplos dos 170);
- com a distância de mahalanobis linear o reconhecimento foi de 98.2% (167 exemplos dos 170);
- com a distância de mahalanobis não linear o reconhecimento foi de 99.4% (169 exemplos dos 170), utilizando como argumento do *kernel* $\sigma = 0.8$, calibrado manualmente.

As Tabelas 7, 8 e 9 contêm as matrizes de confusões com as informações de reconhecimento por classe de poses utilizando as 3 distâncias consideradas.

Tabela 7 – Matriz de confusão do reconhecimento de exemplos de poses estáticas dos sinais dos números em LIBRAS e de poses estáticas dos gestos personalizados para as operações matemáticas básicas utilizando distância euclidiana.

classe	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_D	(%)
c_0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_1	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_2	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_3	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_4	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	1	90.0
c_5	0	0	0	0	0	9	0	0	0	0	0	0	1	0	0	0	0	0	90.0
c_6	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	100.0
c_7	0	1	0	0	0	0	0	6	3	0	0	0	0	0	0	0	0	0	60.0
c_8	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	100.0
c_9	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	100.0
c_{10}	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	100.0
c_{11}	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	100.0
c_{12}	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	100.0
c_{13}	0	0	0	0	0	0	0	0	0	0	0	0	1	9	0	0	0	0	90.0
c_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	2	0	0	80.0
c_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	100.0
c_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	100.0

Fonte – Autor, 2017.

5.2.2 Análise do Reconhecimento de Gestos Dinâmicos Offline

Para análise de acurácia das classes de gestos dinâmicos, criamos um conjunto de treinamento de gestos composto de 150 exemplos de gestos (10 exemplos por classe). Na etapa de extração de *key poses* dos gestos foi utilizado o parâmetro comprimento de blocos

Tabela 8 – Matriz de confusão do reconhecimento de exemplos de poses estáticas dos sinais dos números em LIBRAS e de poses estáticas dos gestos personalizados para as operações matemáticas básicas utilizando distância de mahalanobis linear.

classe	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_D	(%)
c_0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_1	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_2	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_3	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_4	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_5	0	0	0	0	0	9	0	0	0	0	0	0	1	0	0	0	0	0	90.0
c_6	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	100.0
c_7	0	1	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	90.0
c_8	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	100.0
c_9	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	100.0
c_{10}	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	100.0
c_{11}	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	100.0
c_{12}	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	100.0
c_{13}	0	0	0	0	0	0	0	0	0	0	0	0	1	9	0	0	0	0	90.0
c_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	100.0
c_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	100.0
c_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	100.0

Fonte – Autor, 2017.

Tabela 9 – Matriz de confusão do reconhecimento de exemplos de poses estáticas dos sinais dos números em LIBRAS e de poses estáticas dos gestos personalizados para as operações matemáticas básicas utilizando distância de mahalanobis não linear.

classe	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_D	(%)
c_0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_1	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_2	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_3	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_4	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_5	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_6	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	100.0
c_7	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	100.0
c_8	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	100.0
c_9	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	100.0
c_{10}	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	100.0
c_{11}	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	100.0
c_{12}	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	100.0
c_{13}	0	0	0	0	0	0	0	0	0	0	0	0	1	9	0	0	0	0	90.0
c_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	100.0
c_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	100.0
c_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	100.0

Fonte – Autor, 2017.

de *key poses* sucessivas como $M = 8$. A Tabela 10 exibe os detalhes de reconhecimento por classe de gestos dinâmicos.

Observe que houve uma melhora razoável nos resultados utilizando as distâncias

Tabela 10 – Desempenho do reconhecimento das classes de gestos que representam os números em LIBRAS e as operações matemáticas básicas.

		Euclidiana		**Mahalanobis Linear**		**Mahalanobis Não Linear**	
Classe	Quantidade	Acertos	Acurácia (%)	Acertos	Acurácia (%)	Acertos	Acurácia (%)
g_0	10	10	100.0	10	100.0	10	100.0
g_1	10	9	90.0	9	90.0	9	90.0
g_2	10	10	100.0	10	100.0	10	100.0
g_3	10	10	100.0	10	100.0	10	100.0
g_4	10	9	90.0	10	100.0	10	100.0
g_5	10	9	90.0	9	90.0	10	100.0
g_6	10	10	100.0	10	100.0	10	100.0
g_7	10	9	90.0	10	100.0	10	100.0
g_8	10	8	80.0	9	90.0	9	90.0
g_9	10	10	100.0	10	100.0	10	100.0
g_{10}	10	10	100.0	10	100.0	10	100.0
g_{11}	10	10	100.0	10	100.0	10	100.0
g_{12}	10	9	90.0	10	100.0	10	100.0
g_{13}	10	10	100.0	10	100.0	10	100.0
g_{14}	10	10	100.0	10	100.0	10	100.0
Geral	150	143	95.3	147	98.0	148	98.6

Fonte – Autor, 2017

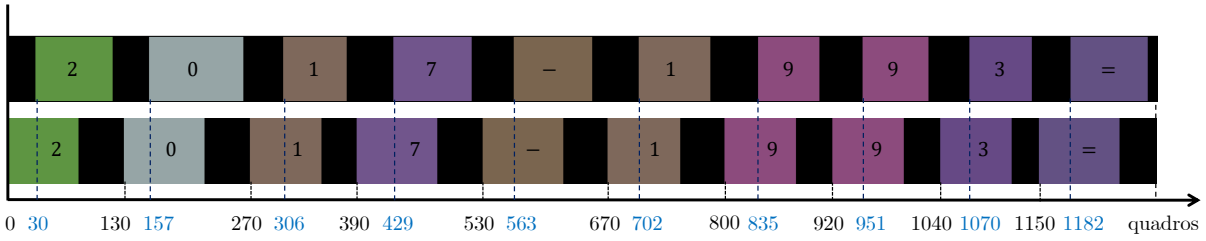
de mahalanobis linear e não linear quando comparados com a utilização da distância euclidiana. No entanto, vale destacar novamente que a utilização do Classificador DMM já ofereceu bons resultados apenas com a distância euclidiana. Isto pode ser explicado, de certa forma, por conta do conjunto de poses estáticas treinadas não oferecer grandes dificuldades de rastreamento para o sensor Leap Motion, que fornece um rastreamento bom para a maioria das classes de poses estáticas desta aplicação.

5.2.3 Análise do Reconhecimento de Gestos Dinâmicos Online

No reconhecimento *online* foram criados diversos conjuntos com execuções de gestos *online*. Na maioria deles o classificador de gestos *online* foi capaz de reconhecer de forma robusta e antecipada todos os gestos realizados, com uma latência de reconhecimento inferior a 50% da execução do gesto para todas as classes, isto já era esperado uma vez que cada classe de gestos é treinada como uma sequência de 2 *key poses* apenas.

A Figura 28 ilustra a linha do tempo de um destes conjuntos correspondente a execução da expressão " $2017 - 1993 =$ ", composto pelas execuções de 8 das 15 classes de gestos treinadas e segmentado utilizando a distância de **mahalanobis não linear** como base para a extração de *key poses*.

Figura 30 – Linhas do tempo de execução e reconhecimento de gestos utilizados para representar os números em LIBRAS e as operações matemáticas básicas segmentados manualmente (linha do tempo inferior) e segmentados de forma automática pelo algoritmo (linha do tempo superior).



Fonte – Autor, 2017.

Vale ressaltar que para ambas as distâncias euclidiana e mahalanobis linear, utilizadas no Classificador DMM, o resultado da segmentação automática dos gestos também foi satisfatória. A Tabela 11 apresenta as taxas de latência do reconhecimento dos gestos do conjunto analisado na Figura 30.

Tabela 11 – Latência do reconhecimento dos gestos utilizados para representar os números em LIBRAS e as operações matemáticas básicas em três abordagens: por quadros, por segundo e por porcentagem de execução do gesto.

Classe	**Latência (quadros)**	**Latência (segundos)**	**Latência** (%)
g_2	30	0.75	37.5
g_0	27	0.67	30.0
g_1	36	0.90	45.0
g_7	39	0.97	43.3
g_{11}	33	0.82	36.6
g_1	32	0.80	40.0
g_9	35	0.87	43.7
g_9	31	0.77	38.7
g_3	30	0.75	37.7
g_{14}	32	0.80	35.5

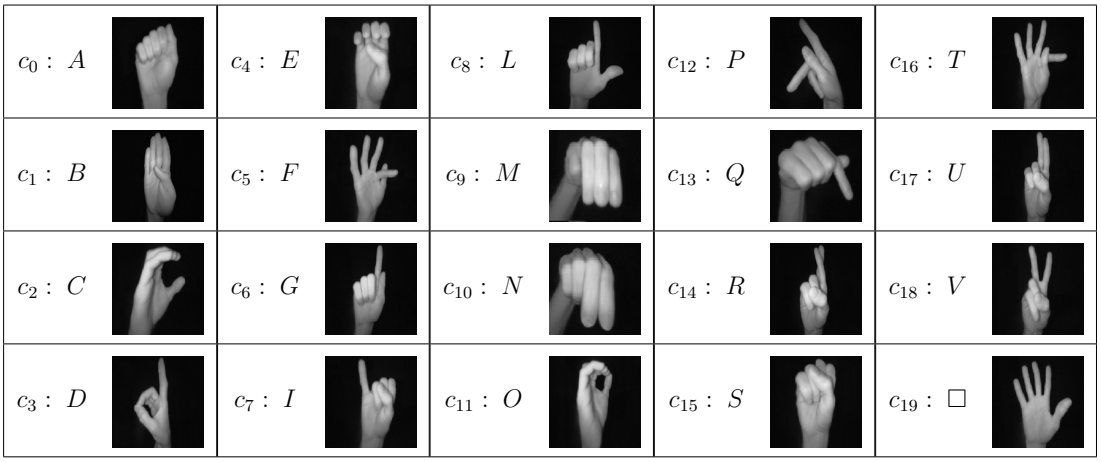
Fonte – Autor, 2017.

5.3 Reconhecimento de Sinais Estáticos do Alfabeto Manual em LIBRAS e de Palavras como Gestos Dinâmicos

Nesta terceira aplicação, utilizamos o método de reconhecimento de gestos para realizar o reconhecimento de palavras treinadas como gestos dinâmicos. Neste contexto, cada palavra é tratada como um gesto onde as poses são os sinais estáticos do alfabeto manual em *Lingua Brasileira de Sinais* (LIBRAS).

Iniciamos treinamos o conjunto de poses da mão descrito na Figura 31, composto de 20 classes de poses estáticas **invariantes por rotação e translação**. As classes de c_0 a c_{18} são os sinais estáticos do alfabeto manual em LIBRAS e c_{19}, representada como "□", é uma classe de poses auxiliar utilizada para evitar que palavras menores sejam treinadas como subgestos de palavras maiores. Abaixo explicamos com mais detalhes este processo.

Figura 31 – Classes de poses estáticas para o reconhecimento de sinais estáticos do alfabeto manual em LIBRAS e palavras como gestos dinâmicos.



Fonte – Autor, 2017.

Para esta aplicação, consideramos o conjunto de palavras da Tabela 12 e treinamos cada palavra como sendo um gesto dinâmico periódico (de período 2), obtido pela realização dos sinais estáticos correspondentes as suas letras, seguida da pose correspondente a classe c_{19}. Isto é feito para possibilitar o reconhecimento de palavras que são subpalavras de outras, uma vez que a abordagem de reconhecimento de gestos *online* utilizando grafos de ação não detecta gestos que são subgestos de algum outro gesto do conjunto.

Por exemplo, para reconhecer palavras como "DA" e "OS", que são subpalavras de "APLICADA" e "GESTOS", respectivamente; treinamo-las como os gestos

DA□DA□ e OS□OS□. De forma análoga, treinamos os gestos APLICADA□APLICADA□ e GESTOS□GESTOS□ para as palavras “APLICADA” e “GESTOS”, respectivamente.

Finalmente, para que a máquina reconheça essas palavras, o usuário deve executar o respectivo gesto com período igual a 2 para palavras que sejam subpalavras de outras, enquanto para as demais, o usuário precisa em geral realizar apenas 1 período do respectivo gesto. Dessa forma, para que a máquina reconheça a palavra “OS”, por exemplo, o usuário deve realizá-la exatamente como foi treinada, isto é, como OS□OS□. Enquanto para o reconhecimento da palavra “GESTOS”, basta realizar o gesto GESTOS□, ao invés de GESTOS□GESTOS□ como treinada.

A inclusão da classe de pose c_{19} (□) no fim do gesto correspondente a cada palavra é útil também para possibilitar o reconhecimento *online* de letras, quando consideradas como palavras. O motivo disto é o mesmo pelo qual foi utilizada uma segunda pose no reconhecimento dos números em LIBRAS, quando considerados como gestos dinâmicos.

Para possibilitar o reconhecimento de poses e gestos para esta aplicação, criamos:

1. um conjunto de treinamento de classes de poses estáticas, composto de 381 exemplos distribuídos entre as 20 classes de poses estáticas, sendo 230 para treinamento e 151 para validação;

2. um conjunto de treinamento de gestos dinâmicos, composto de 368 exemplos de gestos distribuídos entre as 20 classes de gestos consideradas.

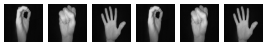
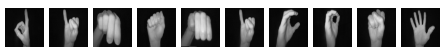
O conjunto de treinamento de poses estáticas foi então passado como entrada dos algoritmos LMNN e KLMNN para obter as melhores distâncias de mahalanobis linear e não linear, respectivamente, para a aplicação.

5.3.1 Análise do Reconhecimento de Poses Estáticas

Para análise de acurácia dos sinais estáticos, criamos um conjunto de testes de poses com 200 exemplos de poses (10 por classe). Abaixo exibimos os resultados utilizando as 3 distâncias consideradas no Classificador DMM: **euclidiana**, **mahalanobis linear** e **mahalanobis não linear**.

No geral foram obtidos os seguintes resultados:

Tabela 12 – Gestos dinâmicos da mão para reconhecimento de palavras utilizando o alfabeto manual em LIBRAS.

Classe	Gesto	Key Poses	Representação Visual
g_0	A	$c_0\ c_{19}\ c_0\ c_{19}$	
g_1	E	$c_4\ c_{19}\ c_4\ c_{19}$	
g_2	O	$c_{11}\ c_{19}\ c_{11}\ c_{19}$	
g_3	AS	$c_0\ c_{15}\ c_{19}\ c_0\ c_{15}\ c_{19}$	
g_4	OS	$c_{11}\ c_{15}\ c_{19}\ c_{11}\ c_{15}\ c_{19}$	
g_5	DA	$c_3\ c_0\ c_{19}\ c_3\ c_0\ c_{19}$	
g_6	DO	$c_3\ c_{11}\ c_{19}\ c_3\ c_{11}\ c_{19}$	
g_7	DE	$c_3\ c_4\ c_{19}\ c_3\ c_4\ c_{19}$	
g_8	UM	$c_{17}\ c_9\ c_{19}\ c_{17}\ c_9\ c_{19}$	
g_9	UMA	$c_{17}\ c_9\ c_0\ c_{19}\ c_{17}\ c_9\ c_0\ c_{19}$	
g_{10}	EM	$c_4\ c_9\ c_{19}\ c_{17}\ c_9\ c_{19}$	
g_{11}	NÃO	$c_{10}\ c_0\ c_{11}\ c_{19}\ c_{10}\ c_0\ c_{11}\ c_{19}$	
g_{12}	LINEAR	$c_8\ c_7\ c_{10}\ c_4\ c_0\ c_{14}\ c_{19}$	
g_{13}	APLICADA	$c_0\ c_{12}\ c_8\ c_7\ c_2\ c_0\ c_3\ c_0\ c_{19}$	
g_{14}	POSE	$c_{12}\ c_{11}\ c_{15}\ c_4\ c_{19}$	
g_{15}	POSES	$c_{12}\ c_{11}\ c_{15}\ c_4\ c_{15}\ c_{19}$	
g_{16}	ESTÁTICA	$c_4\ c_{15}\ c_{16}\ c_0\ c_{16}\ c_7\ c_2\ c_0\ c_{19}$	
g_{17}	ESTÁTICAS	$c_4\ c_{15}\ c_{16}\ c_0\ c_{16}\ c_7\ c_2\ c_0\ c_{15}\ c_{19}$	
g_{18}	GESTO	$c_6\ c_4\ c_{15}\ c_{16}\ c_{11}\ c_{19}$	
g_{19}	GESTOS	$c_6\ c_4\ c_{15}\ c_{16}\ c_{11}\ c_{15}\ c_{19}$	
g_{20}	DINÂMICO	$c_3\ c_7\ c_{10}\ c_0\ c_9\ c_7\ c_2\ c_{11}\ c_{19}$	
g_{21}	DINÂMICOS	$c_3\ c_7\ c_{10}\ c_0\ c_9\ c_7\ c_2\ c_{11}\ c_{15}\ c_{19}$	
g_{22}	MESTRADO	$c_9\ c_4\ c_{15}\ c_{16}\ c_{14}\ c_0\ c_3\ c_{11}\ c_{19}$	
g_{23}	COMPUTAÇÃO	$c_2\ c_{11}\ c_9\ c_{12}\ c_{17}\ c_{16}\ c_0\ c_2\ c_0\ c_{11}\ c_{19}$	
g_{24}	GRÁFICA	$c_6\ c_{14}\ c_0\ c_5\ c_7\ c_7\ c_2\ c_0\ c_{19}$	
g_{25}	MATEMÁTICA	$c_9\ c_0\ c_{16}\ c_4\ c_9\ c_0\ c_{16}\ c_7\ c_2\ c_0\ c_{19}$	
g_{26}	OLIMPÍADA	$c_{11}\ c_8\ c_7\ c_9\ c_{12}\ c_7\ c_0\ c_3\ c_0\ c_{19}$	
g_{27}	MÉTRICA	$c_9\ c_4\ c_{16}\ c_{14}\ c_7\ c_2\ c_0\ c_{19}$	

Fonte – Autor, 2017.

- com a distância euclidiana o reconhecimento foi de 72.5% (145 exemplos dos 200);
- com a distância de mahalanobis linear o reconhecimento foi de 82.0% (164 exemplos dos 200);
- com a distância de mahalanobis não linear o reconhecimento foi de 84.5% (169 exemplos dos 200), utilizando como argumento do *kernel* $\sigma = 0.33$ calibrado manualmente.

As Tabelas 13, 14 e 15 contêm as matrizes de confusão com as informações de reconhecimento por classe de poses utilizando as 3 distâncias consideradas.

Tabela 13 – Matriz de confusão do reconhecimento de exemplos de poses estáticas do conjunto de testes de poses dos sinais estáticos em LIBRAS utilizando distância euclidiana.

classe	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	c_{18}	c_{19}	c_D	(%)
c_0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	70.0
c_1	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_2	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_3	0	0	0	8	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80.0
c_4	0	0	1	0	8	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	80.0
c_5	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	90.0
c_6	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_7	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_8	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_9	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	3	70.0
c_{10}	2	0	0	0	0	0	0	0	0	2	1	0	0	2	0	0	0	0	0	0	3	10.0
c_{11}	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	100.0
c_{12}	0	0	0	0	0	0	5	0	0	0	0	0	4	0	0	0	0	0	0	0	1	40.0
c_{13}	0	0	0	0	0	0	0	0	0	0	2	0	0	8	0	0	0	0	0	0	0	80.0
c_{14}	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	4	0	0	4	0.0
c_{15}	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	1	30.0
c_{16}	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	30.0
c_{17}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	100.0
c_{18}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	7	0	1	70.0
c_{19}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	100.0

Fonte – Autor, 2017.

5.3.2 Análise do Reconhecimento do Gestos Dinâmicos Offline

Para análise de acurácia das palavras da Tabela 12, criamos um conjunto de treinamento de gestos composto de 280 exemplos, 10 exemplos por classe de palavra. Na etapa de extração de *key poses* dos gestos foi utilizado o parâmetro comprimento de blocos de *key poses* sucessivas como $M = 15$. A Tabela 16 exibe os detalhes de reconhecimento por palavra treinada.

Tabela 14 – Matriz de confusão do reconhecimento de exemplos de poses estáticas do conjunto de testes de poses de sinais estáticos em LIBRAS utilizando distância de mahalanobis linear.

classe	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	c_{18}	c_{19}	c_D	(%)
c_0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	1	70.0
c_1	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_2	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_3	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_4	0	0	1	0	8	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	80.0
c_5	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	90.0
c_6	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	1	90.0
c_7	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_8	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_9	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	1	0	0	0	0	0	90.0
c_{10}	2	0	0	0	0	0	0	0	0	3	5	0	0	0	0	0	0	0	0	0	0	50.0
c_{11}	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	100.0
c_{12}	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	100.0
c_{13}	2	0	0	0	0	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	1	60.0
c_{14}	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	4	0	0	3	0.0
c_{15}	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	70.0
c_{16}	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	80.0
c_{17}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	100.0
c_{18}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	6	0	2	60.0
c_{19}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	100.0

Fonte – Autor, 2017.

Tabela 15 – Matriz de confusão do reconhecimento de exemplos de poses estáticas do conjunto de testes de poses de sinais estáticos em LIBRAS utilizando distância de mahalanobis não linear.

classe	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	c_{18}	c_{19}	c_D	(%)
c_0	3	0	0	0	0	0	0	0	0	0	1	0	0	0	0	5	0	0	0	0	1	30.0
c_1	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_2	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_3	0	0	0	9	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	90.0
c_4	0	0	1	0	7	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	70.0
c_5	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_6	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_7	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_8	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_9	0	0	0	0	0	0	0	0	0	7	2	0	0	0	0	0	0	0	0	0	1	70.0
c_{10}	0	0	0	0	0	0	0	0	0	0	6	0		2	0	0	0	0	0	0	2	60.0
c_{11}	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	100.0
c_{12}	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	1	90.0
c_{13}	0	0	0	0	0	0	0	0	0	0	1	0	0	9	0	0	0	0	0	0	0	90.0
c_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	2	1	0	1	60.0
c_{15}	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	90.0
c_{16}	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	70.0
c_{17}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	100.0
c_{18}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	7	0	1	70.0
c_{19}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	100.0

Fonte – Autor, 2017.

Tabela 16 – Desempenho do reconhecimento das classes de gestos que representam as palavras treinadas como gestos dinâmicos.

		Euclidiana		**Mahalanobis Linear**		**Mahalanobis Não Linear**	
Classe	Quantidade	Acertos	Acurácia (%)	Acertos	Acurácia (%)	Acertos	Acurácia (%)
g_0	10	10	100.0	10	90.0	9	90.0
g_1	10	8	80.0	8	70.0	8	80.0
g_2	10	9	90.0	10	100.0	10	100.0
g_3	10	10	100.0	9	100.0	9	90.0
g_4	10	10	100.0	10	70.0	10	100.0
g_5	10	9	90.0	9	100.0	8	80.0
g_6	10	7	70.0	9	100.0	7	70.0
g_7	10	9	90.0	10	100.0	10	100.0
g_8	10	7	70.0	8	80.0	8	80.0
g_9	10	8	80.0	8	80.0	7	70.0
g_{10}	10	10	100.0	10	100.0	10	100.0
g_{11}	10	9	90.0	7	70.0	7	70.0
g_{12}	10	8	80.0	10	100.0	6	60.0
g_{13}	10	9	90.0	8	80.0	7	70.0
g_{14}	10	10	100.0	10	100.0	9	90.0
g_{15}	10	10	100.0	10	100.0	9	90.0
g_{16}	10	9	90.0	9	90.0	7	70.0
g_{17}	10	10	100.0	9	90.0	9	90.0
g_{18}	10	7	70.0	10	100.0	8	80.0
g_{19}	10	6	60.0	8	80.0	9	90.0
g_{20}	10	9	90.0	9	90.0	9	90.0
g_{21}	10	7	70.0	6	60.0	7	70.0
g_{22}	10	6	60.0	7	70.0	8	80.0
g_{23}	10	7	70.0	8	80.0	7	70.0
g_{24}	10	8	80.0	9	90.0	7	70.0
g_{25}	10	6	60.0	9	90.0	9	90.0
g_{26}	10	8	80.0	10	100.0	8	80.0
g_{27}	10	10	100.0	10	100.0	10	100.0
Geral	280	236	84.3	250	89.3	231	82.5

Fonte – Autor, 2017.

Em nossos testes obtivemos uma melhoria razoável na classificação de palavras utilizando a distância de mahalanobis linear. No entanto, houve uma queda de desempenho utilizando a distância de mahalanobis não linear, isto provavelmente se deve ao fato de a calibração manual do parâmetro do *kernel* σ não ter sido o parâmetro ótimo para o conjunto de palavras considerado. Isto pode ser corrigido em trabalhos futuros pela implementação de uma calibração automática. Observamos, no entanto, que utilizando-se o mesmo parâmetro na classificação dos sinais descritos na seção anterior, o reconhecimento com a distância de mahalanobis não linear foi muito superior ao reconhecimento utilizando a distância euclidiana, e ainda relativamente superior a distancia de mahalanobis. Este fato nos leva a crer que o parâmetro escolhido se adequa relativamente bem ao conjunto de sinais estáticos treinado, porém é insuficiente para uma boa generalização de cada classe

de sinais, uma vez que durante o reconhecimento de gestos existem diversas variações dos sinais treinados que talvez não estejam presentes no conjunto de testes de sinais estáticos criado.

5.3.3 Análise do Reconhecimento de Gestos Dinâmicos Online

No reconhecimento *online* foram criados diversos conjuntos com execuções de gestos correspondentes à frases elaboradas a partir do conjunto de palavras treinadas. Na maioria destes conjuntos houveram erros de classificação e segmentação das palavras que cumpunham os conjuntos para ambas as 3 distâncias consideradas. Isto se deve principalmente a imprecisão de ratreamento do sensor Leap Motion para algumas classes de sinais da Figura 31, sinais estes que posssuem oclusões parciais ou totais de dedos da mão, fazendo com que o sensor não rastrei corretamente dedos importantes para diferenciar algumas classes de sinais que são parecidos. Abaixo exibimos o resultado de reconhecimento e segmentação para dois destes conjuntos *onlines*.

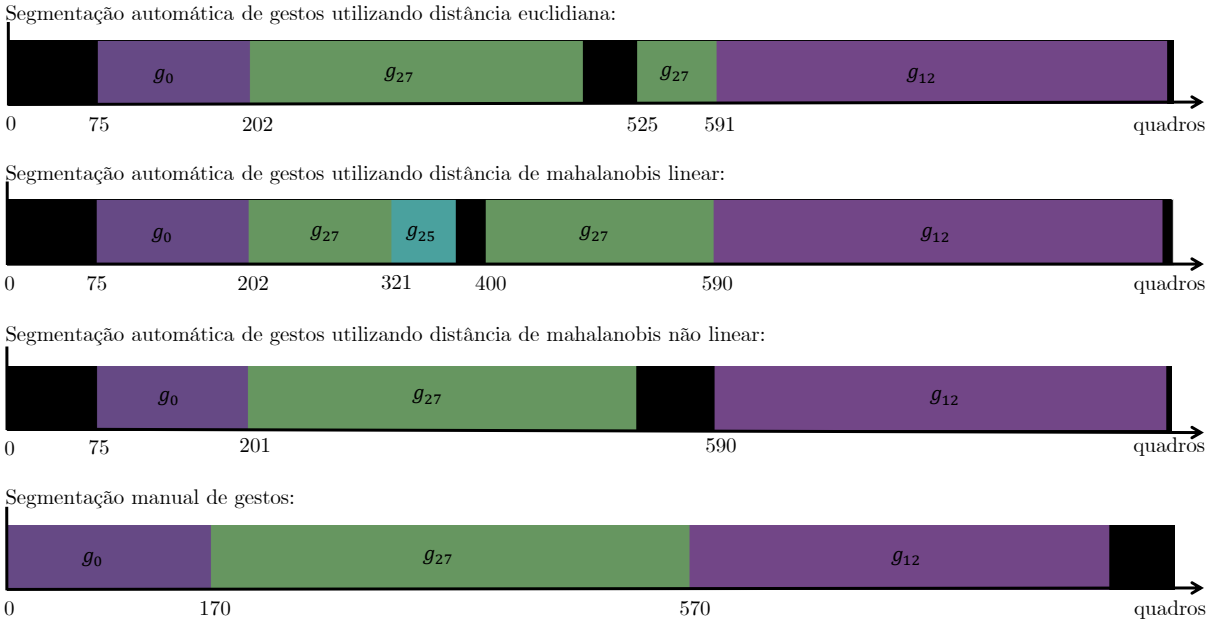
Primeiro conjunto: "A MÉTRICA LINEAR"

Para este conjunto o reconhecimento e segmentação das palavras se deu de forma parecida para cada distância utilizada. As únicas diferenças notáveis se deram no reconhecimento da palavra "MÉTRICA", onde em dados instantes de sua realização o classificador reconheceu sinais equivocadamente, o que induziu a uma quebra no fluxo de reconhecimento da palavra. Com a distância de mahalanobis, por exemplo, no quadro 321 o algoritmo confundiu o gesto corrente com o gesto da palavra "MATEMÁTICA" (classe g_{25}).

A Figura 32 ilustra as linhas do tempo de execução e de como se deu o reconhecimento e segmentação das palvras do conjunto. As faixas pretas indicam que no período correspondente o algoritmo não reconheceu nenhuma palavra. A Tabela 17 apresenta as taxas de latência do reconhecimento das palavras para este conjunto.

Observe que a latência do reconhecimento da palavra "A" foi alta em relação as demais palavras do conjunto. Isto se deu por conta do fato já comentado no início desta seção, o de que subpalavras precisam de uma segunda execução para serem reconhecidos adequadamente; enquanto as palavras "MÉTRICA" e "LINEAR" com poucos sinais estáticos realizados já são facilmente detectadas pelo algoritmo.

Figura 32 – Linhas do tempo de execução e reconhecimento das palavras da frase “A MÉTRICA LINEAR” segmentada manualmente (linha do tempo inferior) e segmentada de forma automática pelo algoritmo utilizando diferentes distâncias (linhas do tempo superiores).



Fonte – Autor, 2017.

Tabela 17 – Latência do reconhecimento dos gestos utilizados para representar palavras da frase “A MÉTRICA LINEAR” como gestos dinâmicos em LIBRAS utilizando a distância de mahalanobis não linear em três abordagens: por quadros, por segundo e por porcentagem de execução do gesto.

Classe	**Latência (quadros)**	**Latência (segundos)**	**Latência** (%)
g_0	75	1.87	44.1
g_{27}	31	0.77	7.7
g_{12}	20	0.50	5.7

Fonte – Autor, 2017.

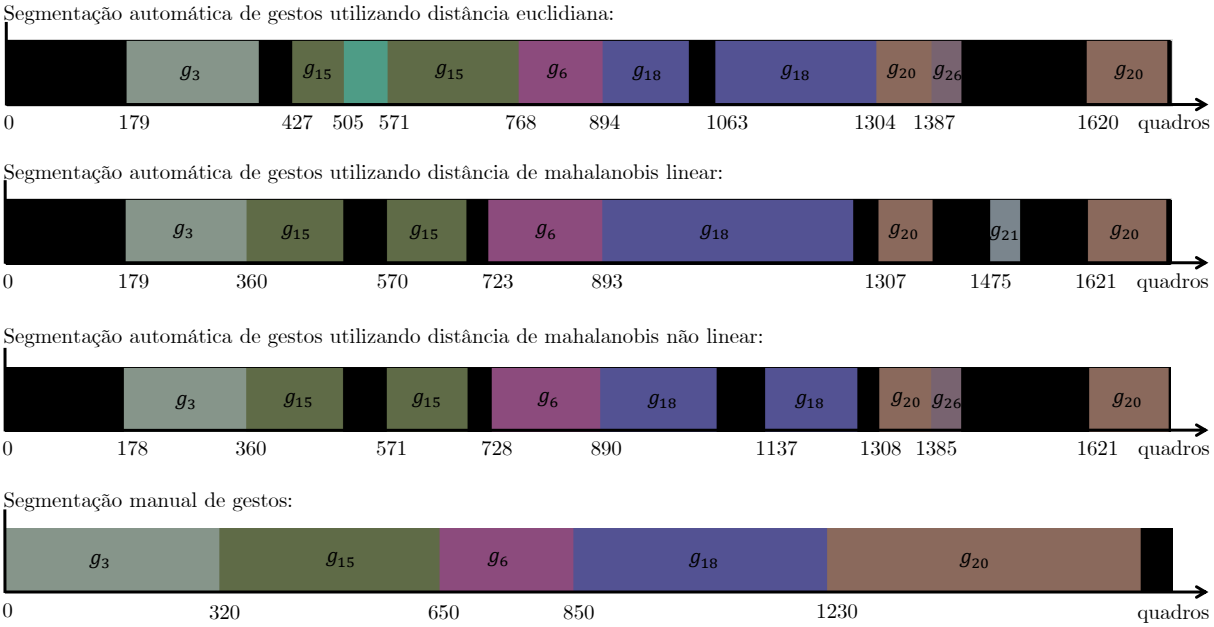
Segundo conjunto: “AS POSES DO GESTO DINÂMICO”

O reconhecimento e segmentação das palavras deste segundo conjunto também se deu de forma parecida para cada distância utilizada. No entanto, desta vez o algortimo cometeu erros de segmentação e classificação com todas as 3 distâncias utilizadas. A palavra “DINÂMICO” (classe g_{20}) se mostrou a de reconhecimento mais difícil, sendo segmentada equivocadamente com todas as distâncias. Isto se deve provavelmente ao fato

da palavra conter as letras M (classe c_9) e N (classe c_{10}), que possuem sinais em LIBRAS difíceis de o sensor Leap Motion rastrear com precisão.

A Figura 33 ilustra as linhas do tempo de execução e de como se deu o reconhecimento e segmentação das palvras do conjunto. As faixas pretas indicam que no período correspondente o algoritmo não reconheceu nenhuma palavra. A Tabela 18 apresenta as taxas de latência do reconhecimento das palavras para este conjunto.

Figura 33 – Linhas do tempo de execução e reconhecimento das palavras da frase "AS POSES DO GESTO DINÂMICO" segmentada manualmente (linha do tempo inferior) e segmentada de forma automática pelo algoritmo utilizando diferentes distâncias (linhas do tempo superiores).



Fonte – Autor, 2017.

5.4 Reconhecimento de Sinais Dinâmicos do Alfabeto Manual em LIBRAS

Nesta última aplicação, utilizamos o método de reconhecimento de gestos dinâmicos para efetuar o reconhecimento dos sinais dinâmicos do alfabeto manual na *Lingua Brasileira de Sinais* (LIBRAS). Para isto, definimos primeiramente o conjunto de *key poses* da Figura 34; composto de 15 poses estáticas **variantes por rotação e translação**.

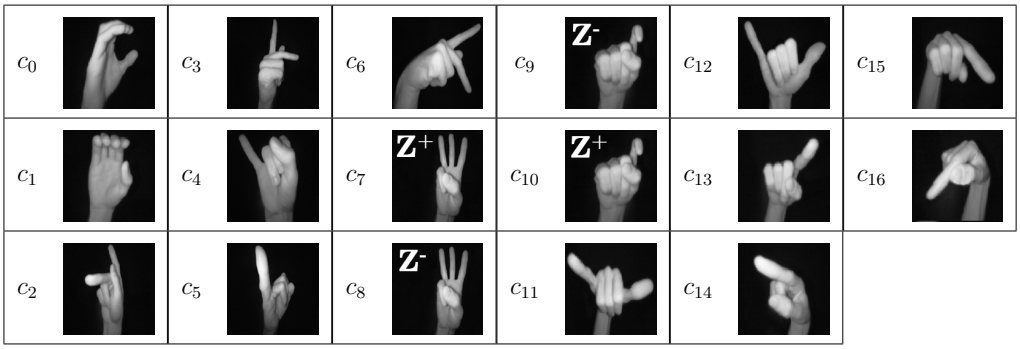
A partir destas *key poses* treinamos os sinais dinâmicos do alfabeto manual em LIBRAS como os gestos dinâmicos da Tabela 19.

Tabela 18 – Latência do reconhecimento dos gestos utilizados para representar palavras da frase “AS POSES DO GESTO DINÂMICO” como gestos dinâmicos em LIBRAS utilizando a distância de mahalanobis não linear em três abordagens: por quadros, por segundo e por porcentagem de execução do gesto.

Classe	**Latência (quadros)**	**Latência (segundos)**	**Latência** (%)
g_3	138	3.45	43.1
g_{15}	40	1.00	12.1
g_6	78	1.95	39.0
g_{18}	40	1.00	10.5
g_{20}	78	1.95	16.6

Fonte – Autor, 2017.

Figura 34 – Classes de poses estáticas utilizadas para o reconhecimento dos sinais dinâmicos do alfabeto manual em LIBRAS.



Fonte – Autor, 2017.

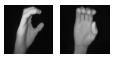
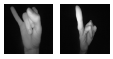
Para possibilitar o reconhecimento eficaz destes gestos, criamos:

1. o conjunto de treinamento de classes de poses estáticas, composto por 215 exemplos distribuídos entre as 17 classes de poses, sendo 128 para treinamento e 87 para validação;

2. o conjunto de treinamento dos sinais dinâmicos, composto por 88 exemplos de gestos distribuídos entre os 8 sinais dinâmicos consideradas.

Passamos então o conjunto de *key poses* como entrada dos algoritmos LMNN e KLMNN para obter as melhores distâncias de mahalanobis linear e não linear, respectivamente, para a aplicação.

Tabela 19 – Gestos dinâmicos da mão para representação dos sinais dinâmicos do alfabeto manual em LIBRAS.

Classe	Gesto	Key Poses	Representação Visual
g_0	Ç	c_0 c_1	
g_1	H	c_2 c_3	
g_2	J	c_4 c_5	
g_3	K	c_6 c_3	
g_4	W	c_7 c_8	
g_5	X	c_9 c_{10}	
g_6	Y	c_{11} c_{12}	
g_7	Z	c_{13} c_{14} c_{15} c_{16}	

Fonte – Auto, 2017.

5.4.1 Análise do Reconhecimento de Poses Estáticas

Para análise de acurária das classes de poses estáticas, criamos um conjunto de testes de poses estáticas com 170 exemplos de poses (10 por classe). Abaixo exibimos os resultados utilizando as 3 distâncias consideradas no Classificador DMM: **euclidiana**, **mahalanobis linear** e **mahalanobis não linear**.

No geral foram obtidos os seguintes resultados:

- com a distância euclidiana o reconhecimento foi de 85.9% (146 exemplos dos 170);
- com a distância de mahalanobis linear o reconhecimento foi de 93.5% (159 exemplos dos 170);
- com a distância de mahalanobis não linear o reconhecimento foi de 98.2% (167 exemplos dos 170), utilizando como argumento do *kernel* $\sigma = 0.8$ calibrado manualmente.

Note como a utilização da distância de mahalanobis proporcionou um desempenho de classificação muito superior ao quando utilizado a distância euclidiana comum.

As Tabelas 20, 21 e 22 contém as matrizes de confusões com as informações de reconhecimento por classe de poses utilizando as 3 distâncias consideradas.

Tabela 20 – Matriz de confusão do reconhecimento de exemplos de poses estáticas dos sinais dinâmicos do alfabeto manual em LIBRAS utilizando distância euclidiana.

classe	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_D	(%)
c_0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_1	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_2	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	80.0
c_3	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	2	80.0
c_4	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	2	80.0
c_5	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	2	80.0
c_6	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	3	70.0
c_7	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	100.0
c_8	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	100.0
c_9	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	6	40.0
c_{10}	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	100.0
c_{11}	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	1	90.0
c_{12}	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	100.0
c_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	5	50.0
c_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	100.0
c_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	100.0
c_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	1	90.0

Fonte – Autor, 2017.

Tabela 21 – Matriz de confusão do reconhecimento de exemplos de poses estáticas dos sinais dinâmicos do alfabeto manual em LIBRAS utilizando distância de mahalanobis linear.

classe	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_D	(%)
c_0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	90.0
c_1	1	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	80.0
c_2	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_3	0	0	1	8	0	0	0	0	0	0	0	0	0	0	0	0	0	1	80.0
c_4	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_5	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_6	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	100.0
c_7	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	100.0
c_8	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	100.0
c_9	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	1	90.0
c_{10}	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	100.0
c_{11}	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	1	90.0
c_{12}	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	100.0
c_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	2	80.0
c_{14}	0	0	1	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	90.0
c_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	100.0
c_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	1	90.0

Fonte – Autor, 2017.

Tabela 22 – Matriz de confusão do reconhecimento de exemplos de poses estáticas dos sinais dinâmicos do alfabeto manual em LIBRAS utilizando distância de mahalanobis não linear.

classe	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_D	(%)
c_0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_1	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_2	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_3	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_4	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_5	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	100.0
c_6	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	100.0
c_7	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	100.0
c_8	0	0	0	0	0	0	0	1	9	0	0	0	0	0	0	0	0	0	90.0
c_9	0	0	0	0	0	0	0	0	0	9	0	0	0	0	1	0	0	0	90.0
c_{10}	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	100.0
c_{11}	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	100.0
c_{12}	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	100.0
c_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	100.0
c_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	100.0
c_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	100.0
c_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	9	0	90.0

Fonte – Autor, 2017.

5.4.2 Análise do Reconhecimento de Gestos Dinâmicos Offline

Para análise de acurácia das classes de sinais dinâmicos, criamos um conjunto de treinamento de gestos composto de 80 exemplos de gestos (10 exemplos por classe). Na etapa de extração de *key poses* dos gestos foi utilizado o parâmetro comprimento de blocos de *key poses* sucessivas como $M = 8$. A Tabela 23 exibe os detalhes de reconhecimento por classe de sinais dinâmicos.

Tabela 23 – Desempenho do reconhecimento das classes de gestos que representam os sinais dinâmicos do alfabeto manual em LIBRAS.

		Euclidiana		**Mahalanobis Linear**		**ahalanobis não linear**	
Classe	Quantidade	Acertos	Acurácia (%)	Acertos	Acurácia (%)	Acertos	Acurácia (%)
g_0	10	10	100.0	9	90.0	10	100.0
g_1	10	9	90.0	10	100.0	10	100.0
g_2	10	10	100.0	10	100.0	10	100.0
g_3	10	7	70.0	10	100.0	10	100.0
g_4	10	10	100.0	10	100.0	10	100.0
g_5	10	10	100.0	10	100.0	10	100.0
g_6	10	10	100.0	10	100.0	10	100.0
g_7	10	10	100.0	10	100.0	10	100.0
Geral	80	76	95.0	79	98.7	80	100.0

Fonte – Autor, 2017.

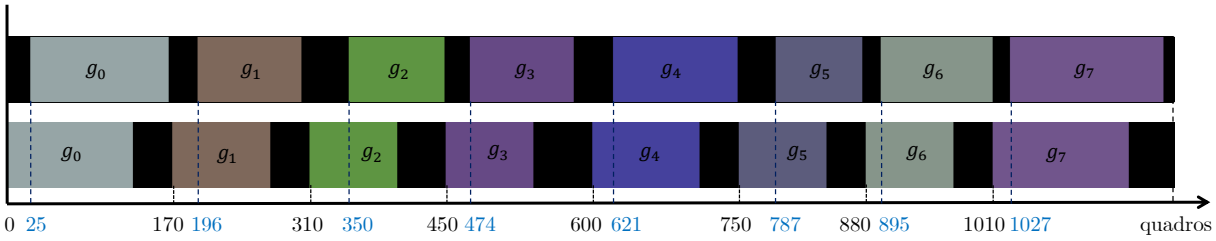
Observe que houve uma melhoria razoável nos resultados utilizando as distâncias de mahalanobis Linear e não linear se comparados com a utilização da distância euclidiana. No entanto, vale destacar novamente que a utilização do Classificador DMM já ofereceu boms resultados apenas com a utilização da distância euclidiana, mesmo com a acurácia de reconhecimento de poses estáticas ter sido bastante inferior.

5.4.3 Análise do Reconhecimento de Gestos Dinâmicos Online

No reconhecimento *online* foram criados diversos conjuntos com execuções de gestos *online*. Em todos estes conjuntos o classificador de gestos *online* conseguiu reconhecer de forma robusta e antecipada todos os gestos realizados, com uma latência de reconhecimento inferior a 50% da execução do gestos para todas as classes de gestos.

A Figura 35 ilustra um destes conjuntos, composto pela execução de todas as 8 classes de gestos treinadas, onde foi utilizada a distância de mahalanobis não linear para detecção das *key poses*.

Figura 35 – Linhas do tempo de execução e reconhecimento de sinais dinâmicos do alfabeto manual em LIBRAS segmentados manualmente (linha do tempo inferior) e segmentados de forma automática pelo algoritmo (linha do tempo superior).



Fonte – Autor, 2017.

Utilizando as outras distâncias: euclidiana e mahalanobis linear, o reconhecimento e segmentação automática dos gestos se deram de forma bastante parecida. A Tabela 5 mostra as taxas de latência do reconhecimento dos gestos do conjunto analisado na Figura 28.

Tabela 24 – Latência do reconhecimento dos sinais dinâmicos do alfabeto manual em LIBRAS utilizando três abordagens: por quadros, por segundo e por porcentagem de execução do gesto.

Classe	**Latência (quadros)**	**Latência (segundos)**	**Latência** (%)
g_0	25	0.62	19.2
g_1	26	0.65	26.0
g_2	40	1.00	44.4
g_3	24	0.60	26.6
g_4	21	0.52	19.1
g_5	37	0.92	41.1
g_6	15	0.37	16.6
g_7	17	0.42	12.1

Fonte – Autor, 2017.

CONCLUSÃO

Este trabalho apresentou um método de reconhecimento de poses estáticas e gestos dinâmicos da mão de forma *offline* e *online*. Para isto utilizamos aprendizado supervisionado para ambas as tarefas. No reconhecimento de poses estáticas foi utilizado mais especificamente aprendizagem de métrica para melhorar um classificador de poses estáticas simples. No reconhecimento de gestos foi utilizado a abordagem probabilística de grafos de ação.

Nosso método foi divido em 4 etapas globais. A primeira foi o treinamento das classes de poses estáticas e a aprendizagem das métricas linear e não linear que melhor se adaptavam aos exemplos de poses estáticas treinados. Para encontrar estas métricas foi utilizado o algoritmo LMNN nas versões linear e não linear. A segunda etapa foi a utilização do Classificador DMM com as métricas aprendidas. Estas duas etapas constituíram o método de reconhecimento de poses estáticas. A terceira etapa foi o treinamento das classes de gestos dinâmicos, onde cada classe de gestos foi codificada em um grafo de ação que continha as probabilidades de transições entre poses estáticas que ocorreram nos exemplos de gestos treinados da classe. Na quarta etapa foram utilizados os grafos de ação gerados na etapa anterior para deduzir os rótulos de classes de novos gestos realizados. Esta última etapa de reconhecimento foi realizada de forma *offline* e *online*.

Para avaliar o método de reconhecimento foram criadas 4 aplicações. Em cada uma delas foi analisada a acurácia do método nas seguintes tarefas: reconhecimento de poses estáticas, reconhecimento de gestos dinâmicos *offline* e reconhecimento de gestos dinâmicos realizados de forma *online*. Na primeira aplicação (Seção 5.1), tanto o desempenho do reconhecimento de poses estáticas quanto o de gestos dinâmicos *offline* foram máximos para todas as 3 distâncias utilizadas. Esta aplicação demonstrou que nos casos em que o sensor Leap Motion consegue rastrear todas as classes de poses com precisão então nosso método pode obter índices de acurácia máximos. Na segunda e quarta aplicação (Seção 5.2 e Seção 5.4, respectivamente), que foram um pouco mais desafiadoras que a primeira, podemos notar como a utilização das métricas aprendidas tornaram o método mais robusto com relação a a utilização do método apenas com a distância euclidiana. Já na terceira aplicação (Seção 5.3), a mais desafiadora dentre as quatro, foi observado um índice de acurácia menor com a métrica não linear na tarefa de reconhecimento de

gestos dinâmicos *offline*. Isto ocorreu pelo fato de nesta aplicação muitas classes de poses diferenciarem pouquíssimo uma das outras, diferenciando-se muitas vezes apenas pela posição de um dedo. Além disso, muitas vezes estes dedos ficavam escondidos do sensor, o que acabou prejudicando bastante a classificação das poses correspondentes. Este foi o caso, por exemplo, dos sinais que representam as letras "R" e "U" em LIBRAS.

O baixo desempenho com a utilização da métrica não linear na terceira aplicação evidenciou também a necessidade de desenvolvimento de um algoritmo de calibração automática do parâmetro do *kernel* no algoritmo KPCA utilizado na fase de aprendizagem da métrica não linear; pois, como foi observado, a utilização de um parâmetro do *kernel* ruim pode não só prejudicar parcialmente o desempenho do reconhecimento das poses estáticas, como pode também tornar o método totalmente ineficaz.

Para a tarefa de reconhecimento de gestos dinâmicos da mão de forma *online*, observamos excelentes resultados nas aplicações 5.1, 5.2 e 5.4. Em todos os conjuntos de testes *online* destas aplicações, nosso método reconheceu corretamente todos os gestos executados com latência média inferior a 40% da execução para a maioria dos gestos. No entanto, para a aplicação 5.3 os resultados não foram totalmente satisfatórios. Na maioria dos conjuntos *online* criados para esta aplicação, nosso método reconheceu erroneamente ao menos 1 gesto de cada conjunto.

De um modo geral, nosso trabalho obteve excelentes resultados, principalmente para aplicações onde não existiam muitas oclusões de dedos da mão, pois nestes casos o sensor Leap Motion fornecia um esqueleto sensível que não correspondia totalmente a pose de mão realizada.

6.1 Trabalhos Futuros

Para trabalhos futuros, podemos incluir:

1. Calibração automática do parâmetro do *kernel* no algoritmo KPCA utilizado na etapa de aprendizagem da métrica não linear.

2. Utilização das duas imagens fornecidas pelas câmeras do sensor Leap Motion como recurso extra, a fim de melhorar o reconhecimento de poses da mão com muita oclusão.

3. Desenvolvimento de um sistema de aprendizagem profunda (*Deep Learning*) sobre os recursos fornecidos pelo sensor Leap Motion, a fim de aprender as características que melhor descrevam poses da mão.

Referências

1 CORPORATION, M. **Meet Kinect for Windows**. 2017. Disponível em: <https://developer.microsoft.com/en-us/windows/kinect>. Citado na página 18.

2 CORPORATION, I. **Intel RealSense Technology**. 2017. Disponível em: <https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html>. Citado na página 18.

3 MOTION, I. L. **Leap Motion For Mac and PC**. 2015. Disponível em: <https://www.leapmotion.com/product/desktop>. Citado 2 vezes nas páginas 18 e 29.

4 CHENG, H.; YANG, L.; LIU, Z. Survey on 3D Hand Gesture Recognition. **IEEE Transactions on Circuits and Systems for Video Technology**, IEEE, v. 26, n. 9, p. 1659–1673, 2016. Citado 2 vezes nas páginas 18 e 22.

5 COMMUNITY, B. for the E. U. **Elon Student Creates Interactive Games for Children**. 2016. Disponível em: <http://blogs.elon.edu/technology/elon-student-creates-interactive-games-for-children/>. Citado na página 18.

6 MIRANDA, L. et al. Real-Time Gesture Recognition from Depth Data Through Key Poses Learning and Decision Forests. In: **Proceedings of the 2012 25th SIBGRAPI Conference on Graphics, Patterns and Images**. Washington, DC, USA: IEEE Computer Society, 2012. (SIBGRAPI '12), p. 268–275. ISBN 978-0-7695-4829-6. Disponível em: <http://dx.doi.org/10.1109/SIBGRAPI.2012.44>. Citado 2 vezes nas páginas 19 e 20.

7 BERLEMONT, S. C. **Automatic non Linear Metric Learning - Application to Gesture Recognition**. Tese (Thesis) — Université de Lyon 1, 2016. Disponível em: <https://tel.archives-ouvertes.fr/tel-01379579>. Citado na página 20.

8 BROMLEY, J. et al. Signature Verification Using a Siamese Time Delay Neural Network. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 1994. p. 737–744. Citado na página 20.

9 LAVIOLA, J. J. 3D Gestural Interaction: The State of the Field. **International Scholarly Research Notices**, Hindawi Publishing Corporation, 2013. Citado na página 20.

10 PRESTI, L. L.; CASCIA, M. L. 3D Skeleton-Based Human Action Classification: A Survey. **Pattern Recognition**, Elsevier, v. 53, p. 130–147, 2016. Citado na página 20.

11 POPPE, R. A Survey on Vision-based Human Action Recognition. **Image and Vision Computing**, v. 28, n. 6, p. 976–990, 2010. ISSN 0262-8856. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0262885609002704>. Citado na página 20.

12 PADILLA-LÓPEZ, J. R.; CHAARAOUI, A. A.; FLÓREZ-REVUELTA, F. A Discussion on the Validation Tests Employed to Compare Human Action Recognition Methods Using The MSR Action3D Dataset. **CoRR**, abs/1407.7390, 2014. Disponível em: <http://arxiv.org/abs/1407.7390>. Citado na página 20.

13 VIEIRA, T. et al. Online Human Moves Recognition Through Discriminative Key Poses and Speed-Aware Action Graphs. **Machine Vision and Applications**, v. 28, n. 1, p. 185–200, 2017. Citado 4 vezes nas páginas 20, 22, 71 e 75.

14 LI, W.; ZHANG, Z.; LIU, Z. Expandable Data-Driven Graphical Modeling of Human Actions Based on Salient Postures. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 18, n. 11, p. 1499–1510, 2008. ISSN 1051-8215. Citado na página 20.

15 CHEN, Q. et al. Interacting with Digital Signage Using Hand Gestures. In: SPRINGER. **International Conference Image Analysis and Recognition**. [S.l.], 2009. p. 347–358. Citado na página 21.

16 WANG, R. Y.; POPOVIĆ, J. Real-time Hand-tracking with a Color Glove. Citado na página 21.

17 KUMAR, P.; VERMA, J.; PRASAD, S. Hand Data Glove: A Wearable Real-time Device for Human-computer Interaction. **International Journal of Advanced Science and Technology**, v. 43, 2012. Citado na página 21.

18 REN, Z. et al. Robust Part-based Hand Gesture Recognition Using Kinect Sensor. **IEEE Transactions on Multimedia**, IEEE, v. 15, n. 5, p. 1110–1120, 2013. Citado na página 21.

19 MARIN, G.; DOMINIO, F.; ZANUTTIGH, P. Hand Gesture Recognition with Leap Motion and Kinect Devices. In: IEEE. **2014 IEEE International Conference on Image Processing (ICIP)**. [S.l.], 2014. p. 1565–1569. Citado na página 21.

20 HU, J.-T.; FAN, C.-X.; MING, Y. Trajectory Image Based Dynamic Gesture Recognition with Convolutional Neural Networks. In: IEEE. **Automation and Systems (ICCAS), 2015 15th International Conference on Control**. [S.l.], 2015. p. 1885–1889. Citado na página 21.

21 SCHMIDT, T. et al. Real-time Hand Gesture Recognition Based on Sparse Positional Data. In: **Proceedings of WVC 2014, Brazilian Workshop on Computer Vision**. [S.l.: s.n.], 2014. p. 243–248. Citado na página 21.

22 BREIMAN, L. Random Forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001. Citado na página 21.

23 LU, W.; TONG, Z.; CHU, J. Dynamic Hand Gesture Recognition with Leap Motion Controller. **IEEE Signal Processing Letters**, IEEE, v. 23, n. 9, p. 1188–1192, 2016. Citado na página 21.

24 GUO, D. et al. Sign Language Recognition Based on Adaptive HMMS with Data Augmentation. In: IEEE. **Image Processing (ICIP), 2016 IEEE International 'Conference on'**. [S.l.], 2016. p. 2876–2880. Citado na página 21.

25 JOHN, V. et al. Real-time Hand Posture and Gesture-based Touchless Automotive User Interface Using Deep Learning. In: IEEE. **Intelligent Vehicles Symposium (IV), 2017 IEEE**. [S.l.], 2017. p. 869–874. Citado na página 21.

26 SHOTTON, J. et al. Real-time Human Pose Recognition in Parts from Single Depth Images. In: **CVPR**. [S.l.: s.n.], 2011. p. 1297–1304. Citado na página 27.

27 MOTION, I. L. **API Overview**. 2015. Disponível em: <https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Overview.html>. Citado 3 vezes nas páginas 28, 29 e 30.

28 WEICHERT, F. et al. Analysis of the Accuracy and Robustness of the Leap Motion Controller. **Sensors (Basel, Switzerland)**, 2013. Citado na página 28.

29 MOHRI, M.; ROSTAMIZADEH, A.; TALWALKAR, A. **Foundations of Machine Learning**. [S.l.]: The MIT Press, 2012. Citado na página 31.

30 TSOI, A. C.; YONG, S. L.; HAGENBUCHNER, M. Ranking Web Pages Using Machine Learning Approaches. **IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology**, 2008. Citado na página 31.

31 GOMEZ-URIBE, C. A.; HUNT, N. The Netflix Recommender System: Algorithms, Business Value, and Innovation. **ACM Transactions on Management Information Systems (TMIS)**, 2015. Citado na página 31.

32 BELLET, A.; HABRARD, A.; SEBBAN, M. A Survey and Data, Structured on Metric Learning for Feature Vectors. **CoRR**, abs/1306.6709, 2013. Disponível em: <http://arxiv.org/abs/1306.6709>. Citado 2 vezes nas páginas 36 e 37.

33 KULIS, B. Metric Learning: A Survey. **Foundations and Trends in Machine Learning**, v. 5, n. 4, p. 287–364, 2013. ISSN 1935-8237. Disponível em: <http://dx.doi.org/10.1561/2200000019>. Citado na página 38.

34 WANG, J.; WOZNICA, A.; KALOUSIS, A. Parametric Local Metric Learning for Nearest Neighbor Classification. **ArXiv e-prints**, set. 2012. Citado na página 39.

35 NOH, Y.-K.; ZHANG, B.-t.; LEE, D. D. Classification, Generative Local Metric Learning for Nearest Neighbor. In: LAFFERTY, J. D. et al. (Ed.). **Advances in Neural Information Processing Systems 23**. Curran Associates, Inc., 2010. p. 1822–1830. Disponível em: <http://papers.nips.cc/paper/4040-generative-local-metric-learning-for-nearest-neighbor-classification.pdf>. Citado na página 39.

36 MAHALANOBIS, P. C. On the Generalised Distance in Statistics. In: **Proceedings National Institute of Science, India**. [s.n.], 1936. v. 2, n. 1, p. 49–55. Disponível em: <http://ir.isical.ac.in/dspace/handle/1/1268>. Citado na página 39.

37 DATTORRO, J. **Convex Optimization and Euclidean Distance Geometry**. [S.l.]: Meboo Publishing USA, 2005. Citado na página 40.

38 WEINBERGER, K.; SAUL, L. Distance Metric Learning for Large Margin Nearest Neighbor Classification. **The Journal of Machine Learning Research**, MIT Press, v. 10, p. 207–244, 2009. Citado 4 vezes nas páginas 40, 41, 42 e 43.

39 WEINBERGER, K. **Large Margin Nearest Neighbors**. 2015. Disponível em: <http://www.cs.cornell.edu/~kilian/code/lmnn/lmnn.html>. Citado 2 vezes nas páginas 43 e 62.

40 CHATPATANASIRI, R. et al. On Kernelization of Supervised Mahalanobis Distance Learners. **CoRR**, abs/0804.1441, 2008. Disponível em: <http://arxiv.org/abs/0804.1441>. Citado 2 vezes nas páginas 45 e 52.

41 SHLENS, J. A Tutorial on Principal Component Analysis. In: **Systems Neurobiology Laboratory, Salk Institute for Biological Studies**. [S.l.: s.n.], 2005. Citado na página 45.

42 MERCER, J. Functions of Positive and Negative Type, and their Connection with the Theory of Integral Equations. **Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences**, The Royal Society, v. 209, n. 441-458, p. 415–446, 1909. ISSN 0264-3952. Disponível em: <http://rsta.royalsocietypublishing.org/content/209/441-458/415>. Citado na página 51.

43 RABINER, L. R. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In: **Proceedings of the IEEE**. [S.l.: s.n.], 1989. p. 257–286. Citado na página 55.

44 FRANC, V. **Statistical Pattern Recognition Toolbox**. 2008. Disponível em: <https://cmp.felk.cvut.cz/cmp/software/stprtool/>. Citado na página 63.

45 VIEIRA, A. W. et al. On the Improvement of Human Action Recognition from Depth Map Sequences Using Space-time Occupancy Patterns. **Pattern Recognition Letters**, v. 36, p. 221–227, 2014. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167865513002778>. Citado na página 71.