UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

Análise em tempo e frequência para extração de características de sinais obtidos de acelerômetros triaxiais com aplicação no estudo da marcha

Rafael Shibana Fayan

São Carlos – SP

Análise em tempo e frequência para extração de características de sinais obtidos de acelerômetros triaxiais com aplicação no estudo da marcha

Rafael Shibana Fayan

Orientador: **Prof. Dr. Moacir Antonelli Ponti**

Monografia referente ao projeto de conclusão de curso dentro do escopo da disciplina SSC0670 - Projeto de Formatura I do Departamento de Sistemas de Computação do Instituto de Ciências Matemáticas e de Computação – ICMC-USP para obtenção do título de Bacharel em Engenharia da Computação.
Área de Concentração: Aprendizado de Máquina e Processamento de Imagens

USP – São Carlos
Novembro de 2018

Fayan, Rafael Shibana
Análise em tempo e frequência para extração de características de sinais obtidos de acelerômetros triaxiais com aplicação no estudo da marcha / Rafael Shibana Fayan. - São Carlos - SP, 2018.
51 p.; 29,7 cm.

Orientador: Moacir Antonelli Ponti.
Monografia (Graduação) - Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos - SP, 2018.

1. aprendizado não supervisionado. 2. autoencoders. 3. extração de características. 4. análise espectral. I. Ponti, Moacir Antonelli. II. Instituto de Ciências Matemáticas e de Computação (ICMC/USP). III. Título.

RESUMO

RAFAEL SHIBANA FAYAN. **Análise em tempo e frequência para extração de características de sinais obtidos de acelerômetros triaxiais com aplicação no estudo da marcha**. 2018. 51 f. Monografia (Graduação) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Devido ao envelhecimento da população mundial, uma vasta gama de preocupações começa a surgir quanto à qualidade de vida dos indivíduos na terceira idade. Uma das maiores na área de saúde é o risco de quedas. O teste médico *Timed Up and Go* (TUG) busca identificar indivíduos com alta suceptibilidade à quedas baseando-se em dados temporais do tempo de execução de um certo conjunto de ações pelo paciente. Este trabalho busca realizar uma análise da efetividade de predição de risco de queda utilizando dados de aceleração provenientes de sensores acelerômetros vestidos por um conjunto de pacientes durante o teste TUG. A análise dos dados dos sensores foi feita no espectro da frequência, utilizando técnicas de aprendizado de máquina não supervisionado para extração de padrões e classificação do paciente em perfis de alto ou baixo risco de queda. Através dos resultados obtidos, espera-se obter uma maior acurácia na predição do risco de queda dos pacientes.

Palavras-chave: aprendizado não supervisionado, autoencoders, extração de características, análise espectral.

LISTA DE ILUSTRAÇÕES

Figura 1 – Diagrama de etapas envolvidas no teste Timed Up and Go (TUG). Fonte: (ROMANOVAS, 2012) . . . 13
Figura 2 – Exemplo de espectrograma. Fonte: <https://en.wikipedia.org/wiki/Spectrogram> 15
Figura 3 – Estrutura generalizada de auto-encoders . . . 16
Figura 4 – Uma ilustração da arquitetura de um AE undercomplete com: entrada x, duas camadas para o encoder que é uma sequência de duas funções $f_2(f_1(.))$ produzindo o código z, seguido de duas camadas para o *decoder* com a sequência de duas funcões $g_2(g_1(.))$ produzindo a saída $\hat{x}$. . . . 17
Figura 5 – Exemplo de uma rede neural com uma camada escondida densa. . . . 17
Figura 6 – Ao utilizar convolução, processa-se informações locais utilizando cada posição (x, y) como centro. Essa região é chamada de campo receptivo. Seus valores são então usados como entrada para um filtro i com parâmetros w_i, produzindo um único valor (pixel) no mapa de características $f_(i,x,y)$ gerado como saída. Fonte: (PONTI; COSTA, 2017) . . . 18
Figura 7 – Representação simplificada do algoritmo Random Forest. Fonte: <https://dsc-spidal.github.io/harp/docs/examples/rf/> . . . 21
Figura 8 – Representação simplificada da técnica de Boosting. Fonte: <https://hackernoon.com/boosting-algorithms-adaboost-gradient-boosting-and-xgboost-f74991cad38c> 21
Figura 9 – Sinais de aceleração no domínio do tempo (em amostragem) . . . 24
Figura 10 – Sinais de aceleração no domínio da frequência . . . 24
Figura 11 – Espectrogramas gerados a partir dos sinais de aceleração originais . . . 25
Figura 12 – Sinais de aceleração separados por testes TUGs de um mesmo paciente . . . 26
Figura 13 – Sinal de aceleração composto pela concatenação dos testes TUGs e seu espectrograma . . . 26
Figura 14 – Gráfico de estimativa de densidade de kernel (KDE) para o número de colunas dos espectrogramas dos sinais compostos pela concatenação dos testes TUG 27
Figura 15 – Espectrogramas dos sinais compostos pela concatenação dos testes TUG, truncados ou expandidos até o tamanho de colunas pré-definido . . . 27
Figura 16 – Gráfico de estimativa de densidade de kernel (KDE) para o número de colunas dos espectrogramas dos sinais originais . . . 29
Figura 17 – Gráfico de treinamento do autoencoder completamente conectado . . . 30
Figura 18 – Conjunto de imagens de teste (acima) e respostas de predição do autoencoder de camadas densas (abaixo) . . . 30

Figura 19 – Conjunto de imagens de teste e respostas de predição do autoencoder convolucional de 128 características 31
Figura 20 – Conjunto de imagens de teste e respostas de predição do autoencoder convolucional de 272 características 31
Figura 21 – Conjunto de imagens de teste e respostas de predição do autoencoder convolucional de 544 características 33
Figura 22 – Conjunto de imagens de teste e respostas de predição do autoencoder convolucional de 1088 características 33
Figura 23 – Conjunto de imagens de teste e respostas de predição do autoencoder de TUGs concatenadas de 128 características 33
Figura 24 – Conjunto de imagens de teste e respostas de predição do autoencoder de TUGs concatenadas de 250 características 34
Figura 25 – Conjunto de imagens de teste e respostas de predição do autoencoder de TUGs concatenadas de 500 características 34
Figura 26 – Variância cumulativa e por componente da transformação PCA ao *dataset* de 128 *features* 35
Figura 27 – Variância cumulativa e por componente da transformação PCA ao *dataset* de 272 *features* 35
Figura 28 – Variância cumulativa e por componente da transformação PCA ao *dataset* de 544 *features* 35
Figura 29 – Variância cumulativa e por componente da transformação PCA ao *dataset* de 1088 *features* 36
Figura 30 – Sumário das métricas obtidas, agregando os modelos por tipo de entrada . . 39
Figura 31 – Sumário das métricas obtidas para os modelos de entrada com espectrogramas dos sinais TUG, agregando por tipo de classificador 39
Figura 32 – Sumário das métricas obtidas para os modelos de entrada com espectrogramas dos sinais TUG e classificadores AdaBoosting ou GradientBoosting, agregando por número de features 40
Figura 33 – Sumário das métricas obtidas para os modelos de entrada com espectrogramas dos sinais TUG, classificadores AdaBoosting ou GradientBoosting e 250 features, agregando por número de componentes PCA 40
Figura 34 – Matriz de Confusão, Curva ROC e Curva de Precisão-revocação para o classificador Gradient Boosting com sinais TUG, 250 características e 64 componentes PCA 41
Figura 35 – Matriz de Confusão, Curva ROC e Curva de Precisão-revocação para o classificador SVM com sinais TUG, 500 características e 32 componentes PCA 41
Figura 36 – Matriz de Confusão, Curva ROC e Curva de Precisão-revocação para o classificador Ada Boosting com sinais TUG, 128 características e 32 componentes PCA 41

LISTA DE TABELAS

Tabela 1 – Sumário das métricas de perda para os autoencoders implementados 33
Tabela 2 – Melhores 10 resultados com base na métrica ROC AUC 37
Tabela 3 – Piores 10 resultados com base na métrica ROC AUC 37
Tabela 4 – Melhores 10 resultados com base na métrica F1 Score 38

SUMÁRIO

1 INTRODUÇÃO 11
1.1 Motivação e Contextualização 11
1.2 Objetivos 12
1.3 Organização 12

2 MÉTODOS, TÉCNICAS E TECNOLOGIAS UTILIZADAS 13
2.1 Considerações iniciais 13
2.2 Timed Up and Go (TUG) Test 13
2.3 Análise em Frequência e Espectrogramas 14
2.4 Auto-encoders (AEs) 15
2.4.1 Camadas densas e convolucionais *17*
2.4.2 Denoising Autoencoders *18*
2.5 Principal Component Analysis (PCA) 18
2.6 Algoritmos de Classificação 19
2.7 Métricas de Avaliação para algoritmos de classificação 20
2.8 Considerações finais 22

3 DESENVOLVIMENTO 23
3.1 Considerações Iniciais 23
3.2 Descrição das Atividades Realizadas 23
3.2.1 Obtenção dos dados de aceleração e características para comparação *23*
3.2.2 Geração dos espectrogramas a partir dos dados de aceleração originais *24*
3.2.3 Construção de diferentes arquiteturas de Autoencoders para extração de features *27*
3.2.4 Aplicação de técnicas de aprendizado de máquina para classificação dos pacientes *34*
3.3 Resultados Obtidos 37
3.4 Dificuldades e Limitações 42
3.5 Considerações Finais 43

4 CONCLUSÃO 45
4.1 Contribuições 45
4.2 Relacionamento entre o Curso e o Projeto 45
4.3 Considerações sobre o Curso de Graduação 46

4.4 Trabalhos Futuros 46

REFERÊNCIAS 49

Capítulo 1

INTRODUÇÃO

1.1 Motivação e Contextualização

Com o aumento da população idosa ao redor do mundo, uma série de preocupações relacionadas a saúde e segurança deste grupo começa a surgir (ALMEIDA *et al.*, 2012). Dentre elas, uma das maiores é a preocupação com o risco de quedas durante a terceira idade devido a alta taxa de incidência e severidade das consequências (SOUZA, 2011). Segundo a Organização Mundial de Saúde, mais de um terço dos idosos acima de 65 anos cai ao menos uma vez ao ano, e essa porcentagem tende a aumentar com a idade (World Health Organization, 2007).

Devido a isso, é de grande interesse da comunidade investigar padrões para rastreio de risco de queda em idosos através de testes clínicos assesíveis para terceira idade. Atualmente este risco é comunmente avaliado através de questionários e testes clinicos que mensuram tempo gasto para realizar um conjunto de procedimentos durante um período de acompanhemto (PHELAN *et al.*, 2015). O Teste Timed Up and Go (TUG), é um exemplo deste tipo de teste clínico, onde o paciente tem se levanta de uma cadeira, anda 3 metros à frente, vira, retorna à cadeira e se senta. Porém, a avaliação do risco de queda através de tais conjuntos de teste não se mostra consistente para diferentes perfis de idosos, pois as métricas utilizadas podem não representar o risco de queda efetivo para um grupo de idosos mais ativos e saudáveis, por exemplo.

Perante a esta situação, dispositivos móveis que auxiliam na detecção de caidores estão sendo estudados para o desenvolvimento de práticas que permitam não somente a detecção de quedas no momento da ocorrência, mas também sua predição a partir da avaliação do risco de queda com os dados coletados. Sensores inerciais, como giroscópios e acelerômetros, presentes em pequenos dispositivos vestíveis, surgem como uma alternativa prática para coleta de dados e avaliação dos riscos de queda em aplicações clínicas (PONTI *et al.*, 2017).

Busca-se neste estudo analisar dados de aceleração coletados com voluntários idosos durante a realização do TUG. Em particular, nesse projeto estamos interessados em avaliar a capacidade discriminativa de características obtidas a partir dos espectrogramas dos sinais de acelerometria (MUS, 2011). Bons resultados provenientes deste estudo podem contribuir para o desenvolvimento de equipamentos acessíveis para a avaliação do perfil de risco de queda em idosos e servir como alternativa para grupos de idosos onde o procedimento comum do teste TUG não se mostra sensível. Além disso, resultados preliminares promissores podem como estímulo para coleta e geração de mais bases de dados de padrões de movimento reais e risco de

queda, contribuindo para futuros estudos, pesquisas e desenvolvimento de programas em áreas correlacionadas.

1.2 Objetivos

O objetivo é investigar amostras de dados de aceleração obtidos durante o Teste Timed Up and Go (TUG) no domínio da frequência através da geração de espectrogramas a partir dos dados de aceleração originais, extração de características relevantes dos espectrogramas aplicando métodos de aprendizado de máquina não supervisionado e, por fim, classificação dos pacientes amostrados entre caidores e não caidores utlizando as características extraídas como entrada para classificadores com aprendizado supervisionado.

Ao final da análise, espera-se obter informações que evidenciem diferenças entre os padrões de idosos caidores e não caidores a partir dos dados de aceleração de marcha, facilitando a detecção de perfis de alto risco de queda.

1.3 Organização

No Capítulo 2 apresenta-se a base teórica utilizada no projeto, incluindo uma descrição breve do teste TUG e os principais conceitos utilizados no projeto. No Capítulo 3 descrevem-se as atividades realizados ao longo do desenvolvimento do projeto, as análises dos resultados obtidos em cada etapa, assim como as limitações e dificuldades encontradas. No Capítulo 4 apresentam-se as conclusões obtidas após a obtenção dos resultados finais e uma análise pessoal da correlação do projeto com o curso de graduação.

Capítulo 2

MÉTODOS, TÉCNICAS E TECNOLOGIAS UTILIZADAS

2.1 Considerações iniciais

2.2 Timed Up and Go (TUG) Test

Os dados de aceleração referentes ao estudo da marcha foram coletados através um acelerômetro durante a realização do teste Timed Up and Go (TUG), que permite a avaliação da mobilidade e equilíbrio do paciente durante mudanças de postura, de sentado para de pé e vice-versa, e marcha.

O teste TUG consiste o paciente, inicialmente sentado em uma cadeira, se levantar, andar 3 metros em linha reta, se virar, retornar andando 3 metros e se sentar novamente. A Figura 1 representa estas etapas. O tempo de execução é cronometrado e correlacionado ao nível de mobilidade da pessoa seu risco de queda.

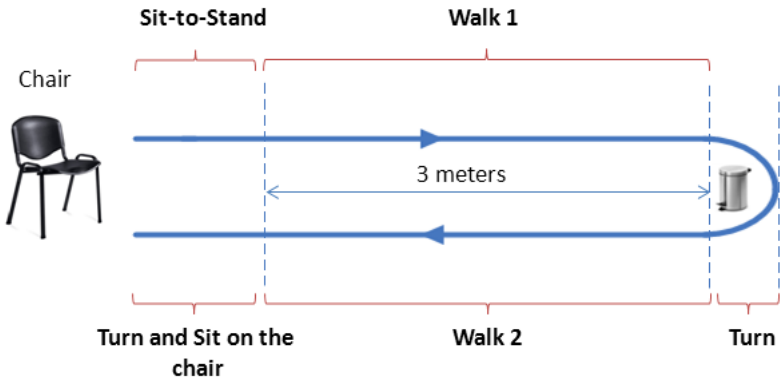

Figura 1 – Diagrama de etapas envolvidas no teste Timed Up and Go (TUG). Fonte: (ROMANOVAS, 2012)

Idosos que conseguem realizar o teste em menos de 20 segundos se mostraram independentes quanto às atividades de mobilidade do dia-a-dia, enquanto os que necessitaram de em torno de 30 segundos tendem a ser dependentes de uma pessoa fornecendo assistência para muitas atividades (SHUMWAY-COOK; BRAUER; WOOLLACOTT, 2013). Atualmente, o teste TUG apresenta a medida mais sensível e confiável para a avaliação do risco de queda em idosos.

No contexto brasileiro, o melhor preditivo para discriminar idosos caidores e não-caidores é de 12.47 segundos (ALEXANDRE *et al.*, 2012).

Porem, em (PONTI *et al.*, 2017), foi observado que a métrica de tempo de execução do teste não foi tão sensível quando a regra era aplicada a conjuntos de idosos saudáveis. Para este conjunto de indivíduos, dados de acelerometria apresentaram melhores resultados para a discriminação de caidores, em particular utilizando características de frequência dos sinais.

2.3 Análise em Frequência e Espectrogramas

A Transformada de Fourier é uma função matemática que transforma um sinal no domínio do tempo em um somatório de frequências que compõem o sinal original. Ela expressa uma função temporal periódica em termos de funções de base sinusoidais, sendo considerada como a extensão da Série de Fourier resultante quando se aproxima o período da função ao infinito. A Transformada de Fourier e sua transformada inversa, com frequência definida em *Hertz*, possuem as seguintes definições:

$$X(f) = \int_{-\infty}^{+\infty} x(t) e^{-j2\pi ft} dt \tag{2.1}$$

$$x(t) = \int_{-\infty}^{+\infty} X(f) e^{j2\pi ft} df \tag{2.2}$$

Porém, no contexto de aplicações de processamento digitais de sinais, os dados são discretos e compostos por n amostragens, com valores $x[n]$. Nestes casos, é preciso aplicar a Transformada Discreta de Fourier, onde ela e sua inversa são definidas pelas seguintes equações:

$$X_k = \sum_{n=0}^{+N-1} x[n] e^{\frac{-j2\pi kn}{N}} \quad \text{para k = 0, 1, ..., N-1} \tag{2.3}$$

$$x[n] = \frac{1}{N} \sum_{n=0}^{+N-1} X_k e^{\frac{j2\pi kn}{N}} \quad \text{para n = 0, 1, ..., N-1} \tag{2.4}$$

Para os estudos feitos neste projeto, utilizamos a representação dos sinais em formato de espectrogramas. Espectrogramas são representações visuais do sinal que nos fornecem 3 informações contidas no sinal: tempo ou amostragem, frequência e amplitude. Um exemplo de espectrograma pode ser visto na Figura 2. O processo de geração dos espectrogramas é análogo a aplicar a Transformada de Fourier de curto termo (SMITH, 2018), na qual é feita a Transformada de Fourier para uma janela deslizante de tamanho fixo ao longo do sinal a ser avaliado, cujas respostas resultantes de cada transformada correspondem a uma coluna do espectrograma e, quando colocadas lado a lado, resultam na geração de sua imagem. A partir desse espectrograma, serão obtidas as características a serem avaliadas para detecção de caidores.

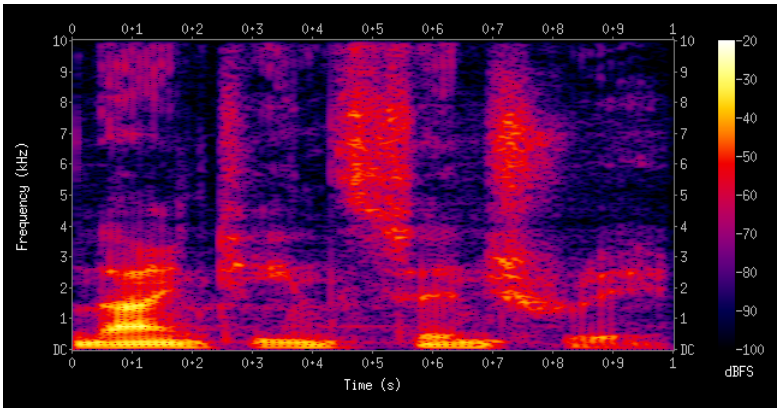

Figura 2 – Exemplo de espectrograma. Fonte: <https://en.wikipedia.org/wiki/Spectrogram>

2.4 Auto-encoders (AEs)

Um autoencoder é uma rede neural que tem como objetivo aproximar uma função identidade. Ou seja, dado uma amostra da base de treinamento x, ele tentará produzir uma saída $\hat{x}$ que seja o mais semelhante possível à entrada x (PONTI *et al.*, 2017). Em outras palavras, ele busca minimizar para cada entrada a função:

$$\ell(x) = ||x - \hat{x}||^2 \tag{2.5}$$

Nosso interesse não está, no entanto, na saída do autoencoder, mas sim na representação dos dados de entrada extraídas na menor das camadas da rede. O valor de tais redes está na sua propriedade de descoberta e extração dos dados mais relevantes.

Sua arquitetura pode ser divida em duas partes: um *encoder* f e um *decoder* g. O *encoder* toma a entrada original e cria uma representação mais compacta dela, denominada de código. Em seguida, o *decoder* tenta reconstruir a entrada original a partir do código (Ver Figura 3). Porém, o código é uma representação compacta dos dados originais, e o *encoder* não será capaz de reproduzir a entrada perfeitamente. Desta forma, o *encoder* tentará fazer a extração das características mais relevantes e representativas dos dados de entrada e o *decoder* tentará reproduzir os dados de entrada da melhor maneira a partir delas.

O autoencoder generaliza bem quando compreende como as informações contidas dados se distribuem, o que resulta num erro de reconstrução dos dados pequeno. O espaço de informações gerado pelo *encoder* é comumente chamado de espaço latente Z, e pode ser visto como uma compressão dos exemplos de entrada em um espaço de menor dimensão de forma que, a partir desse, é possível reconstruir os dados originais com baixo erro. Esse espaço mostrou ser discriminativo e genérico quando consideradas imagens de domínios diferentes (CAVALLARI; RIBEIRO; PONTI, 2018).

Auto-encoders podem ser de dois tipos: *undercomplete* e *overcomplete*. Iremos descrever apenas os undercomplete. Auto-encoders (AEs) undercomplete têm a dimensionalidade do

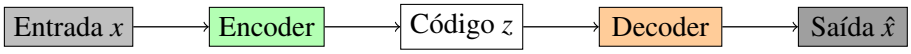

Figura 3 – Estrutura generalizada de auto-encoders

código menor do que a entrada. Assim, o código não é capaz replicar a entrada e é forçado a encontrar uma compressão que melhor retenha as características relevantes dos dados. Para treinar este modelo, minimizamos a seguinte função de custo por imagem de entrada:

$$\ell(x, g(f(x))) = ||x - g(f(x))||^2 \tag{2.6}$$

em que l computa o erro médio quadrático, x é a amostra de entrada, g representa o *decoder*, f representa o *encoder*.

Considerando um *decoder f()* como realizando uma transformação linear, e *L()* o erro médio quadrático, então o AE undercomplete é capaz de aprender o mesmo subespaço que o PCA (Principal Component Analysis), isto é, o subespaço do componente principal do conjunto de treinamento (GOODFELLOW *et al.*, 2016). Por conta desse tipo de comportamento, AEs são frequentemente utilizados para redução de dimensionalidade. Dizemos que um AE tem *tied weights*, ou pesos amarrados, quando a matriz de pesos do *decoder* é espelhada com relação ao *encoder*. Em particular se o *encoder* emprega pesos W, então o respectivo decoder emprega pesos W^t (a transposta da matriz de pesos do *encoder*) no processo de treinamento do autoencoder. Esse cenário assume que seja possível aprender apenas uma única matriz inversível de pesos capaz de guiar os mapeamentos $f : X \to Z$ (gerando o espaço latente) e $g : Z \to \hat{X}$ (gerando o espaço das imagens reconstruídas).

Denominamos arquiteturas como sendo as configurações das camadas do *encoder*/*decoder*. Um AE com múltiplas camadas (ou funções) empregadas é chamado de Autoencoder empilhado (*stacked*). Quando se utiliza camadas convolucionais (ver seção 2.4.1 para mais detalhes) é chamado de Autoencoder convolucional. A fim de ilustrar uma possível arquitetura, a Figura 4 demonstra um exemplo de AE undercomplete, com *encoder* e *decoder* compostos por duas camadas cada, com o código é computado por $z = f(x) = f_2(f_1(x))$ e a saída por $\hat{x} = g(h) = g_2(g_1(z))$.

Vale destacar que ao permitir que as funções sejam não-lineares e adicionando diversas camadas, estamos aumentando a capacidade do AE. Nestes cenários, apesar do código ser menor que a entrada, AEs undercomplete ainda são capazes de aprender a copiar a entrada, porque são dados a eles muita capacidade. Como afirmado em Goodfellow *et al.* (2016), se o *encoder* e o *decoder* tiverem capacidade suficiente, seria possível aprender um código unidimensional em que todo conjunto de treinamento x^i é mapeado para um código i usando o autoencoder. Em seguida, o *decoder* mapearia o código de volta para a entrada original. Este exemplo em particular, apesar de não acontecer na prática, ilustra os problemas que podem surgir se um AE undercomplete tiver muita capacidade.

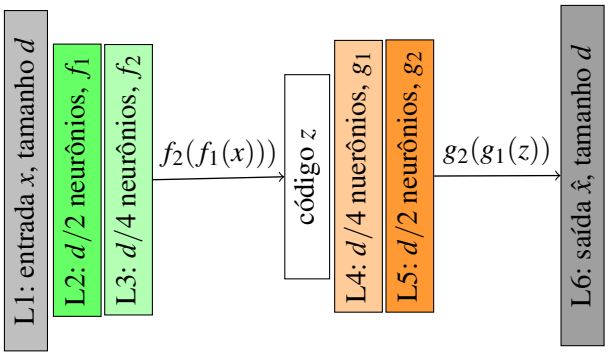

Figura 4 – Uma ilustração da arquitetura de um AE undercomplete com: entrada x, duas camadas para o encoder que é uma sequência de duas funções $f_2(f_1(.))$ produzindo o código z, seguido de duas camadas para o *decoder* com a sequência de duas funcões $g_2(g_1(.))$ produzindo a saída $\hat{x}$.

2.4.1 Camadas densas e convolucionais

Auto-encoders podem ter camadas densas e convolucionais. Nas camadas densas, ou totalmente conectadas, cada neurônio possui um peso associado a cada elemento do vetor de entrada. A Figura 5 ilustra uma rede neural densa.

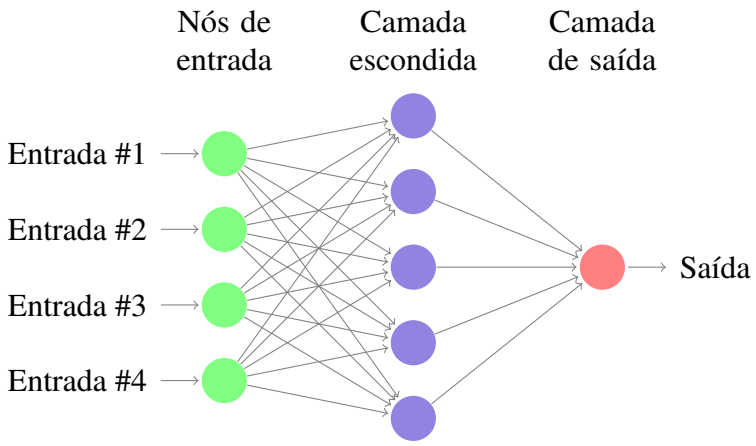

Figura 5 – Exemplo de uma rede neural com uma camada escondida densa.

É possível construir AEs convolucionais substituindo as camadas totalmente conectadas de um AE tradicional por camadas convolucionais. Na camada convolucional cada neurônio é um filtro aplicado a uma imagem de entrada e cada filtro é uma matriz de pesos. Estes modelos são úteis pois eles não necessitam de dados rotulados e podem ser construídos com a finalidade de obter uma extração de características hierárquicas ao utilizar esta arquitetura. A Figura 6 descreve o que é uma convolução.

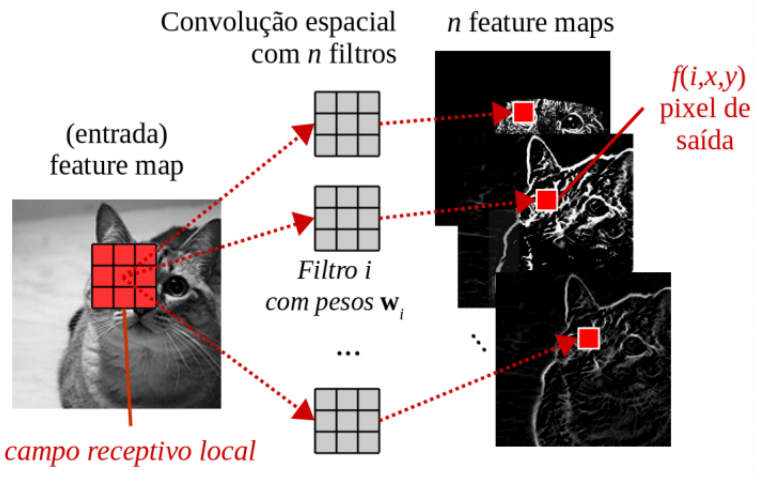

Figura 6 – Ao utilizar convolução, processa-se informações locais utilizando cada posição (x, y) como centro. Essa região é chamada de campo receptivo. Seus valores são então usados como entrada para um filtro i com parâmetros w_i, produzindo um único valor (pixel) no mapa de características $f_(i,x,y)$ gerado como saída. Fonte: (PONTI; COSTA, 2017)

2.4.2 Denoising Autoencoders

Denoising auto-encoders são variações de AEs que aplicam ruído na camada de entrada, de forma que o modelo deverá aprender a reconstruir não apenas a imagem de entrada, mas também a remover o ruído. Para isso é aplicado ruído, e.g. Gaussiano, na entrada x produzindo $\tilde{x}$ que é fornecido como entrada para a rede. No entanto a função de custo deve minimizar a divergência entre a saída e a versão não ruidosa x, e.g.

$$\ell(x) = ||x - f(\tilde{x})||^2 = ||x - \hat{x}||^2. \tag{2.7}$$

Esse tipo de abordagem permite que o autoencoder produza padrões de ativação mais fortes, pois as camadas deverão filtrar o ruído e ao mesmo tempo aprender a compressão (VINCENT *et al.*, 2010).

2.5 Principal Component Analysis (PCA)

O método PCA (Principal Component Analysis) é utilizado para realizar a redução de um conjunto de observações a partir da transformação de vetores e seleção de características. Em suma, o PCA analisa a relevância de cada característica e seleciona aquelas que são as mais prominentes, enquanto as restantes são transformadas e reduzidas. A obtenção do vetor mais importante se dá através dos seguintes passos:

1. **Matriz de Covariância:** É a matriz Σ simétrica obtida ao multiplicar uma matríz M, onde cada elemento $a_{i,j}$ é um dado subtraído pela média μ dos dados no eixo j, por sua

transposta, obtendo o valor esperado da matriz resultante. A fórmula é:

$$\Sigma = M \cdot M^T \tag{2.8}$$

2. **Autovalores e Autovetores:** Para obter autovalores de uma matriz podemos calcular as raízes do polinômio característico. Para cada autovalor Λ, podemos achar o autovetor x se existir um vetor $x \neq 0$ tal que:

$$\Sigma x = \Lambda x \tag{2.9}$$

Onde Σ é a Matriz de Covariância obtida no passo anterior.

3. **Componentes Principais:** Os autovetores obtidos no passo anterior são chamados de Componentes Principais (*Principal Components*). Cada valor de um autovetor representa uma quantidade de energia, e aqueles autovetores com maior energia acumulada são considerados pelo PCA como mais relevantes.

4. **Transformação do Espaço:** Nesse ponto é possível remover a correlação linear identificada pela matriz de covariância ao realizar a multiplicação entre matriz transposta dos Componentes Principais e a matriz transposta dos dados originais

2.6 Algoritmos de Classificação

Algoritmos de classificação são modelos de aprendizado de máquina com capacidade de predizer a quais classes pertencem um conjunto de dados determinados. Podemos visualizá-los de forma que para cada entrada x, o nosso classificador fornecera como resposta y a partir de uma função de mapeamento descrita por $y = f(x)$. Um classificador utiliza uma base de dados de treino para definir o comportamento de f, ou seja, como interpretar os dados de entrada x a fim de obter uma resposta y tida como objetivo. Caso o treinamento tenha sido feito de forma eficiente, espera-se que o classificador compreenda os padrões presentes nos dados de entrada a fim generalizar as predições do modelo e fazer com que elas respondam corretamente para dados de entrada que o classificador nunca viu antes. Descreveremos brevemente abaixo os algoritmos de classificação utilizados no projeto:

- **Support Vector Machines (SVM)**: SVM (HEARST *et al.*, 1998) é um classificador que busca encontrar a separação entre classes de um problema de classificação por meio de hiperplanos que definem as melhores fronteiras com base no critério de máxima margem. Como o SVM é o classificador que encontra separações lineares, a acurácia de sua classificação pode ser vista como uma medida de separabilidade linear entre as classes dado um espaço de características. A complexidade do hiperplano que discrimina as classes do conjunto de dados é definida pelo seu kernel, o que permite que o método continue útil mesmo quando a dimensionalidade dos dados aumenta

- **Random Forests**: Random Forest (BREIMAN, 2001) é um classificador baseado em *ensembles*, técnica que cria diversos classificadores menores e simplificados e combina os seus resultados para produzir a resposta de predição. Ele utiliza em seu aprendizado uma outra técnica chamada *Boostrap Aggregating*, ou simplesmente *Bagging*, que consiste em selecionar aleatoriamente partes do conjunto de dados de treino para o treinamento de cada um dos vários classificadores que compõem o modelo como um todo. Random Forest faz uso dessas técnicas para criar inúmeras árvores de decisão e utiliza um sistema de voto para determinar o resultado da predição. Este modelo minimiza a *variância* e aumenta o *bias* do modelo como um todo, útil para casos onde os classificadores estão decorando as características do conjunto de treino. A Figura 7 é uma representação simplificada do algoritmo.

- **Gradient Boosting**: Gradient Boosting (FRIEDMAN, 2001) é outro algoritmo de classificação baseado em *ensembles*, mas que utiliza uma outra técnica denominada *boosting*, que consiste em construir um classificador eficiente a partir de vários outros classificadores pobres. A ideia por trás desta técnica é começar com um classificador simples, avaliar sua performance, criar um segundo classificador a partir do primeiro tentando corrigir seus erros e repetir este processo. Através dele, o último modelo tende a ser robusto e eficiente para as predições do *dataset*. Gradient Boosting faz o incremento dos modelos fazendo re-treinamento do próximo classificador sob os erros do anterior e posteriormente aplicando otimização de gradiente descendente para o reajuste dos pesos.

- **Ada Boosting**: Ada Boosting (FREUND; SCHAPIRE, 1997; HASTIE *et al.*, 2009), encurtado de Adaptive Boosting, também é um algoritmo baseado em *ensembles* que utiliza *boosting*, mas que difere do Gradient Boosting no modo em que os pesos dos modelos são adaptados a cada iteração. Neste caso, o novo classificador realiza o treinamento sob o conjunto de erros do último modificando os pesos associados a cada amostra, e posteriormente realiza-se uma adição dos pesos do novo classificador ao classificador forte fazendo uma escala de acordo com a performance das predições obtidas. A Figura 8 ilustra o processo de *boosting* descrito.

2.7 Métricas de Avaliação para algoritmos de classificação

Após o treinamento de um classificador, é preciso avaliar de forma quantitativa o quão bem estão sendo suas predições. Para isto, são coletadas diversas métricas referentes a predições que foram feitas no conjunto de testes. A partir do número de verdadeiro-positivos, verdadeiro-negativos, falso-positivos e falso-negativos obtidos ao cruzar os resultados das predições com os resultados esperados, que podem ser mapeadas em uma matriz de confusão, assim como as

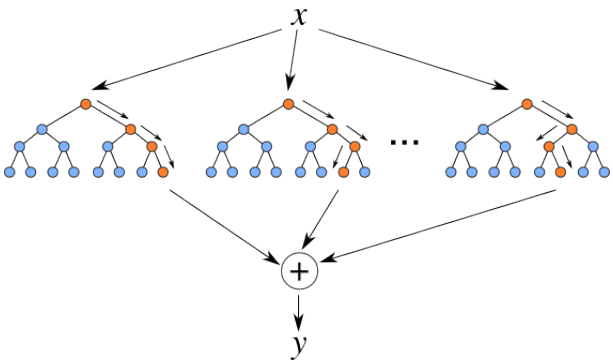

Figura 7 – Representação simplificada do algoritmo Random Forest. Fonte: <https://dsc-spidal.github.io/harp/docs/examples/rf/>

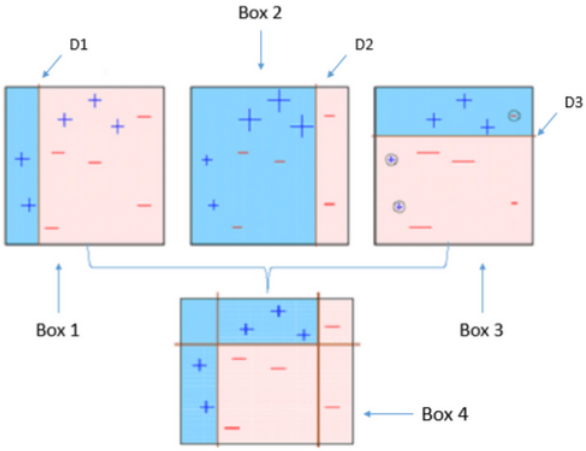

Figura 8 – Representação simplificada da técnica de Boosting. Fonte: <https://hackernoon.com/boosting-algorithms-adaboost-gradient-boosting-and-xgboost-f74991cad38c>

probabilidades de cada predição, é possível obter diversas métricas úteis para avaliar a efetividade do classificador. Descreveremos a seguir algumas delas:

- Acurácia: Probabilidade da predição estar correta.
  Acurácia = $(VP+VN)/Total$
- Revocação (*Recall*) ou Sensibilidade: Probabilidade da predição ser positivas dada uma amostra positiva.
  Sensibilidade = $VP/(VP+FN)$
- Precisão: Probabilidade da amostra ser positiva dada uma predição positiva.
  Precisão = $VP/(VP+FP)$
- Especificidade: Probabilidade da predição ser negativa dada uma amostra negativa.
  Especificidade = $VN/(VN+FP)$
- F1 Score: Média harmônica da precisão e sensibilidade.
  F1 Score = $2*$(Precisão * Revocação)$/$(Precisão + Revocação)

- ROC AUC: Área debaixo da curva ROC, que descreve a variação da sensibilidade no eixo *y* e (1 - especificidade) no eixo *x* para os diferentes *threshold* para classificação das amostras.
- Acurácia Balanceada: Média das sensibilidades obtidas em cada classe.
  Acurácia Balanceada = $1/2 * (VP/(VP+FN) + (VN/(VN+FP))$
- Precisão Média: Área debaixo da curva de Precisão-Revocação, que descreve a variação da precisão no eixo *y* e o *recall* no eixo *x* para os diferentes *threshold* para classificação das amostras.

Exemplos de matrizes de confusão, curvas ROC e curvas de precisão-revocação podem ser encontradas nas Figuras 34, 35 e 36.

2.8 Considerações finais

Para a realização do projeto, foi necessário a obtenção de uma grande bagagem teórica em diferentes áreas do conhecimento. Os conhecimento mencionados neste capítulo foram essenciais para a realização deste estudo, mas além dos tópicos citados, houve muitos outros conceitos estudados e utilizados no desenvolvimento que contribuíram para os resultados obtidos.

Durante praticamente todo desenvolvimento do projeto, sempre foi necessário buscar explicações para os resultados e problemas encontrados, e pesquisar novas tecnologias e abordagens para contornar e solucionar os impedimentos da melhor maneira.

Capítulo 3

DESENVOLVIMENTO

3.1 Considerações Iniciais

Neste capítulo serão apresentadas as atividades realizadas, descrevendo detalhes importantes durante cada etapa e observações sob os resultados obtidos. Ao final do capítulo será feita uma análise final dos resultados obtidos, a efetividade dos métodos e tecnologias utilizadas para a finalidade da aplicação e possíveis hipóteses sobre pontos que não foram conclusivos.

3.2 Descrição das Atividades Realizadas

3.2.1 Obtenção dos dados de aceleração e características para comparação

Para os estudos realizados neste projeto, foi utilizada uma base de dados contendo medições de aceleração de um conjunto de 79 voluntários acima de 60 anos obtidos durante a execução de 9 testes TUG sequencialmente. Um sensor tri-axial foi utilizado para a coleta dos dados, localizado à frente do centro de massa do corpo dos pacientes, coletando amostras de aceleração nas três direções espaciais, nos eixos *x, y e z*, e amostragem em 100Hz. Os sinais utilizados neste estudo correspondem à fusão das acelerações nos três eixos a fim de reduzir a dimensionalidade do problema, utilizando a norma Euclidiana:

$$s = \sqrt{x^2 + y^2 + z^2}$$

Além dos sinais de aceleração fundidos, foram utilizadas também, para fins de comparação com os resultados finais, características extraídas a partir de análises estatísticas sob os dados de aceleração convertidos para o espectro da frequência.

As etapas mencionadas foram abordadas por (PONTI *et al.*, 2017) e (BET, 2018b) e não fazem parte deste estudo, porém alguns dos dados e códigos (BET, 2018a) gerados foram utilizados como base para as atividades deste projeto. Para reprodução da base de dados e extração das características mencionadas a partir dos sinais fundidos, foi realizado um *fork* do repositório e feito algumas alterações nas funções de segmentações dos testes TUG e extração da matriz de *features* (FAYAN, 2018).

3.2.2 Geração dos espectrogramas a partir dos dados de aceleração originais

Com a base de dados em mãos, a primeira atividade realizada foi uma análise exploratória dos dados em Python, utilizando o *framework web* Jupyter Notebook. Carregando o arquivo contendo os sinais de aceleração no domínio do tempo, obtivemos a Figura 9 após visualizar os sinais dos pacientes 5 e 14, o primeiro representando um caidor e o segundo um não caidor.

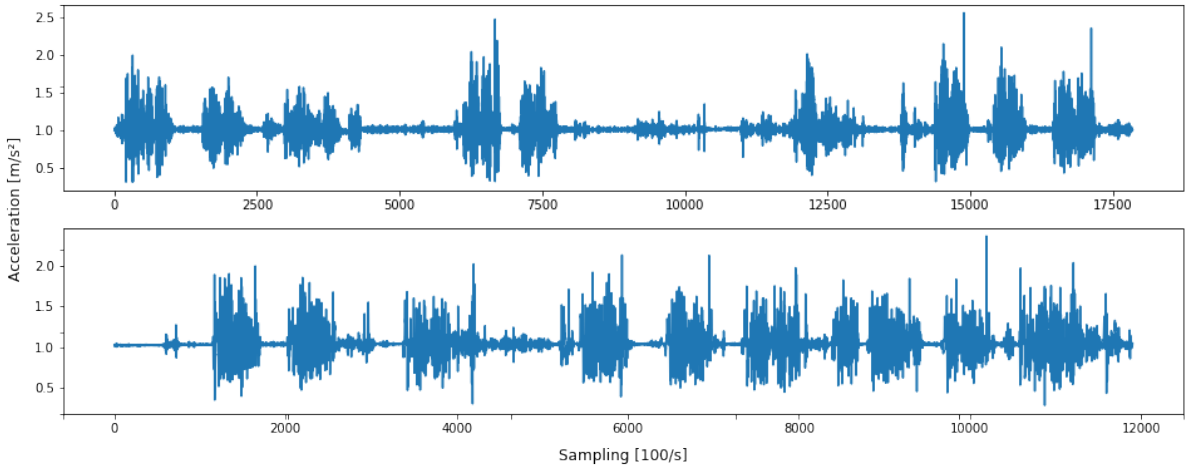

Figura 9 – Sinais de aceleração no domínio do tempo (em amostragem)

Utilizando a biblioteca SciPy (JONES *et al.*, 2001–) e os sinais de aceleração dos pacientes anteriores, aplicou-se a transformada discreta de Fourier (DFT) (ver Seção 2.3) para a geração dos sinais no domínio da frequência e também seus espectrogramas, representados pelas Figuras 10 e 11 respectivamente.

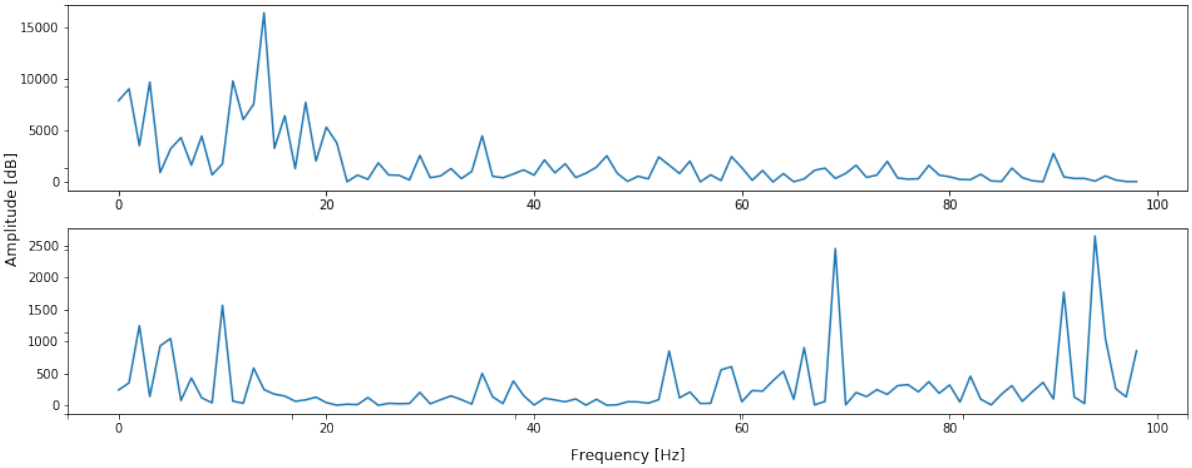

Figura 10 – Sinais de aceleração no domínio da frequência

Estão presentes na base de dados utilizada 73 sinais válidos, pois 6 pacientes do conjunto de 79 foram excluídos devido aos sinais coletados se mostrarem inadequados ou com erros. As

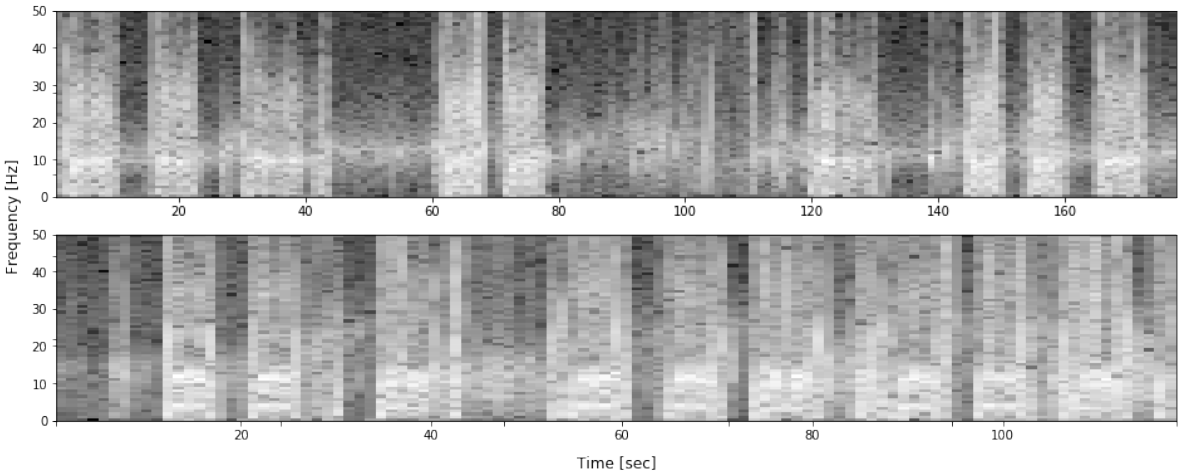

Figura 11 – Espectrogramas gerados a partir dos sinais de aceleração originais

Figuras 9, 10 e 11 representam apenas dois sinais deste conjunto de 73. Através de uma análise mais abrangente do conjunto de dados, alguns pontos de interesse foram notados:

- Não há um padrão bem definido que distingua facilmente os idosos caidores dos não caidores. Podemos ver na Figura 10 que o paciente não caidor possui um alto pico de amplitude perto da frequência de 15 Hz e o paciente caidor possui picos de amplitude acima de 60 Hz, no entanto este padrão é encontrado de forma consistente no conjunto de dados.
- É possível observar bem a separação entre cada um dos 9 testes TUG realizados no sinal de aceleração, pois entre as janelas de maior aceleração, que representam o período de teste, há janelas de pouca aceleração constante, que representam o período de repouso.
- A duração dos testes, e consequentemente o número de amostragens de aceleração, varia muito entre pacientes. Esta diferença é refletida nas dimensões dos espectrogramas gerados para cada sinal, o que é um problema para a etapa de extração de características pelo autoencoder pois todas as entradas devem possuir as mesmas dimensões especificadas na arquitetura. Logo, os espectrogramas terão de ser tratados e redimensionados, o que não é ideal pois essencialmente estaremos cortando e/ou alterando informação do nosso *dataset* gerado.

Além da geração dos espectrogramas utilizando os sinais de aceleração completos, também foram gerados espectrogramas utilizando a concatenação de cada um dos testes TUG em um único sinal, removendo todos períodos de repouso do sinal. Utilizando o paciente 5 novamente como exemplo para visualização, os sinais de aceleração dos 9 testes TUG segmentados podem ser vistos na Figura 12 e o sinal concatenado, assim como seu espectrograma, se encontra na Figura 13

Adicionalmente, como uma tentativa de solucionar o problema dos espectrogramas com dimensões diferentes, optou-se por definir um limiar para o número de colunas da matriz de

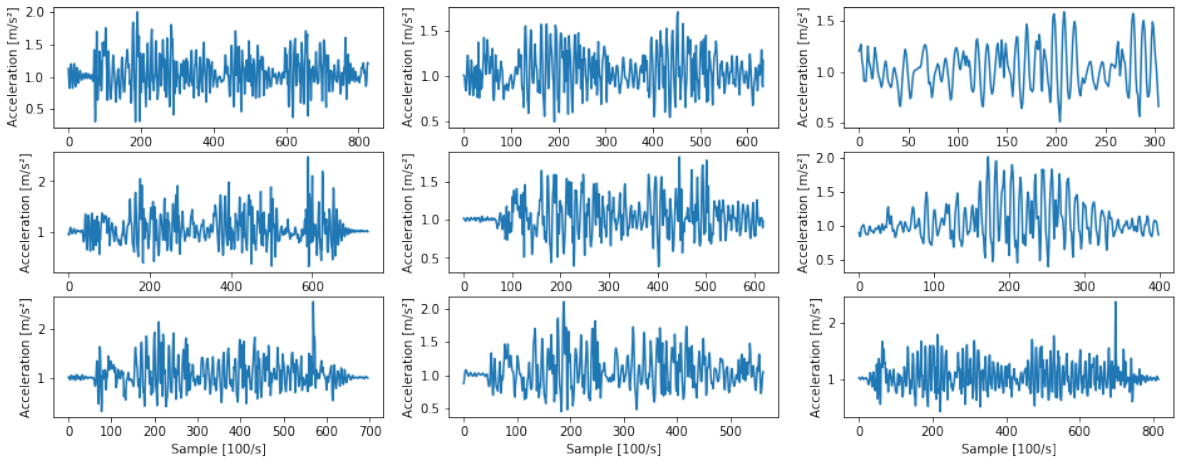

Figura 12 – Sinais de aceleração separados por testes TUGs de um mesmo paciente

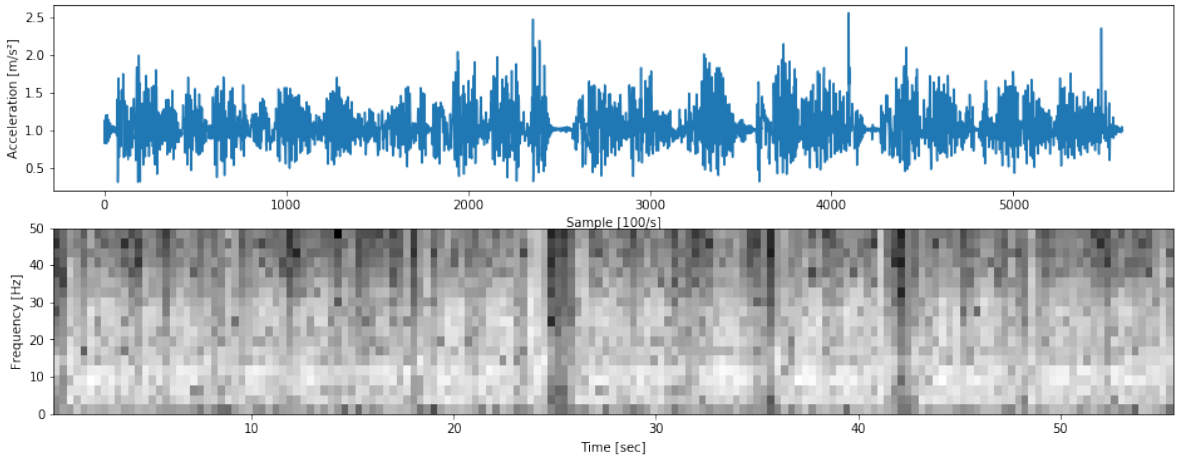

Figura 13 – Sinal de aceleração composto pela concatenação dos testes TUGs e seu espectrograma

pixeis que representa a imagem, de modo que as imagens menores seriam preenchidas com colunas de pixeis com valor 0, e as maiores seriam truncadas até este ponto. A escolha de atribuição do valor 0 se deve ao fato de que valores próximos a 0 representam baixa magnitude e valores próximos a 1 representam alta magnitude. Como desejamos detectar padrões de alta aceleração durante os testes TUG, espera-se que valores de baixa magnitude sejam ignorados pelos classificadores. Com isto, espera-se que as técnicas de extração de *features* e classificação convirjam para a conclusão de que este segmento da imagem seja irrelevante para a determinação do perfil de risco do idoso.

Para determinar o ponto de corte para o número de colunas da imagem, desenhou-se o gráfico de *kernel density estimation* (KDE) (ZAMBOM; DIAS, 2012), representado pela Figura 14, para melhor visualização dos dados.

Observando a distribuição dos dados, vemos que somente 3 espectrogramas possuem acima de 200 colunas, evidenciado pelas pequenas barras verticais próximo ao eixo x, onde cada barra representa uma amostra distinta. Definiu-se 200 colunas como o limiar para o truncamento

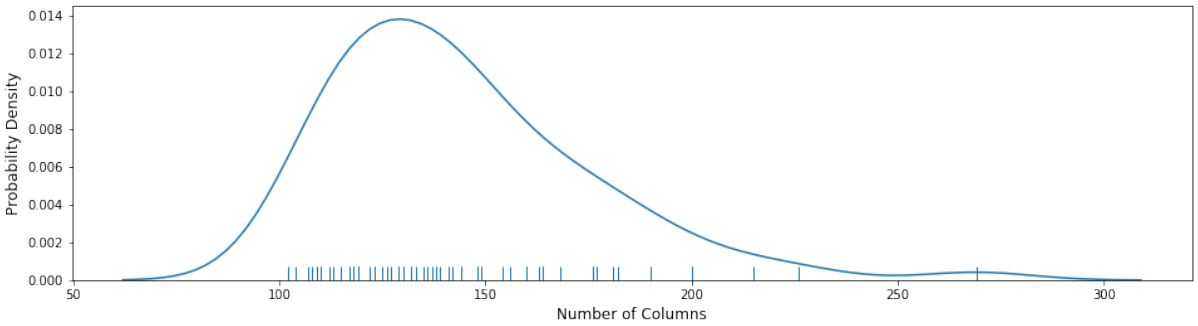

Figura 14 – Gráfico de estimativa de densidade de kernel (KDE) para o número de colunas dos espectrogramas dos sinais compostos pela concatenação dos testes TUG

e expansão das imagens. A Figura 15 nos mostra este efeito aplicado aos espectrogramas dos testes TUGs concatenados para os pacientes 5 e 14. Note que a ausência do do segmento em preto do segundo espectrograma se deve ao fato de que o espectrograma gerado ultrapassa o limiar de 200 colunas.

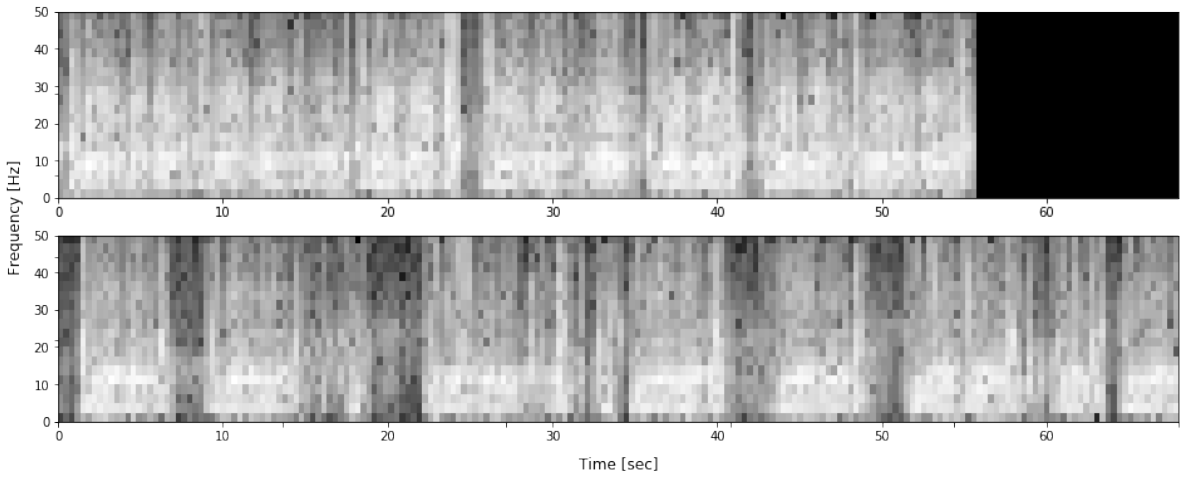

Figura 15 – Espectrogramas dos sinais compostos pela concatenação dos testes TUG, truncados ou expandidos até o tamanho de colunas pré-definido

3.2.3 Construção de diferentes arquiteturas de Autoencoders para extração de features

Com os espectrogramas obtidos, a próxima etapa foi a construção dos autoencoders para extração de *features* relevantes. Para a implementá-los, utilizou-se a biblioteca Tensorflow (ABADI *et al.*, 2016) como *backend* para os algoritmos de aprendizado de máquina em conjunto com a API de alto nível Keras (CHOLLET *et al.*, 2015) para abstração das funções de redes neurais em Python, acelerando o desenvolvimento dos códigos implementados.

A primeira arquitetura de autoencoder implementada foi uma rede neural densa ou completamente conectada, composta por algumas camadas de Dropout para minimizar a ocorrência

Listing 1: Arquitetura do Autoencoder composto somente por camadas densas

```
Layer (type)                 Output Shape              Param #
=================================================================
input_10 (InputLayer)        (None, 4480)              0
_________________________________________________________________
dense_55 (Dense)             (None, 1280)              5735680
_________________________________________________________________
dropout_46 (Dropout)         (None, 1280)              0
_________________________________________________________________
dense_56 (Dense)             (None, 320)               409920
_________________________________________________________________
dropout_47 (Dropout)         (None, 320)               0
_________________________________________________________________
dense_57 (Dense)             (None, 80)                25680
_________________________________________________________________
dropout_48 (Dropout)         (None, 80)                0
_________________________________________________________________
dense_58 (Dense)             (None, 320)               25920
_________________________________________________________________
dropout_49 (Dropout)         (None, 320)               0
_________________________________________________________________
dense_59 (Dense)             (None, 1280)              410880
_________________________________________________________________
dropout_50 (Dropout)         (None, 1280)              0
_________________________________________________________________
dense_60 (Dense)             (None, 4480)              5738880
=================================================================
Total params: 12,346,960
Trainable params: 12,346,960
Non-trainable params: 0
```

de *overfitting* na rede e com um espaço latente (*bottleneck*) de 80 características a serem extraídas pelo *encoder*. A descrição das camadas pode ser vista na Listagem 1.

A fim de garantir que todos os espectrogramas do conjunto de dados pudessem ser utilizados como entrada do modelo de autoencoder definido, foi necessário truncar a matriz dos espectrogramas para garantir as mesmas dimensões para todo conjunto de dados. Como o número de linhas do espectrograma dependente apenas da janela da transformada de Fourier utilizada para sua geração, devemos analisar o número de colunas da matriz de pixeis para determinar um limiar de corte. Para isso, desenhou-se mais uma vez o gráfico de *kernel density estimation* para o número de colunas do conjunto de espectrogramas, representado pela Figura 16

Analisando a distribuição dos dados, conclui-se que o espectrograma com menor número de colunas possui 68 colunas. Definimos, portanto, este valor como o número máximo de colunas a ser utilizado por espectrograma nos autoencoders, selecionando apenas estas primeiras colunas da matriz da imagem. No entanto, como grande parte dos espectrogramas possui em torno

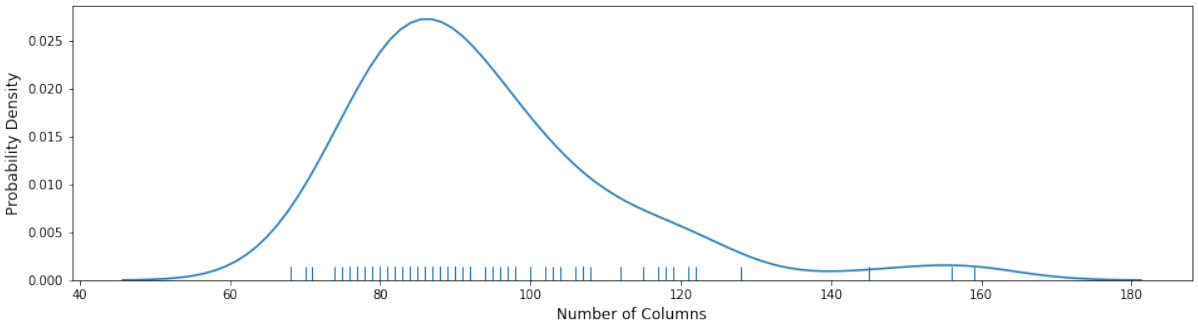

Figura 16 – Gráfico de estimativa de densidade de kernel (KDE) para o número de colunas dos espectrogramas dos sinais originais

de 80 a 100 colunas, para minimizar a perda de informação ao limitar o número de colunas, foram utilizados também as últimas 68 colunas da matriz para casos onde o número de colunas é superior a 50% do limite, ou seja, 102 colunas. Desta forma, evitamos demasiada perda de informação devido à limitação do autoencoder e ao mesmo tempo incrementamos o nosso *dataset* ao adicionar novas amostras.

Aumentação de dados (Data Augmentation): posteriormente, separou-se a base de dados entre conjunto de treino e teste, utilizando 25% da base como teste e o restante como treino. Aplicaram-se técnicas de *data augmentation* no conjunto de treino para obter uma maior amostragem, em específico a inverteu-se das imagens nos eixos x, y e ambos simultaneamente, quadruplicando o conjunto de treino.

Por fim, realizou-se o treinamento do autoencooder utilizando os seguintes hiper-parâmetros para a compilação e treino do modelo:

- Função de perda: Erro quadrático médio
- Otimizador: Adadelta
- *Batch size*: 10
- Épocas: 100

O gráficos de treinamento gerado a partir da métrica de *loss function* coletado para o conjunto de treino e teste está representado na Figura 17 e algumas das imagens reconstruídas na saída do *decoder* podem ser vistas na Figura 18, que mostra uma comparação entre as imagens imagens originais na primeira linha e as reconstruídas na segunda.

Analisando os resultados, vemos que para as 5 imagens de teste a resposta do autoencoder foi praticamente a mesma, ou seja, o modelo convergiu para a solução que minimiza a função de perda de erro quadrado médio para o conjunto de teste. Isto está de acordo com as curvas de perda observadas na Figura 17, pois a partir da época 60, a função de perda do conjunto de teste não diminuiu, enquanto a curva de treino continua reduzindo, o que indica que o modelo está começando a decorar as características do conjunto de treino e se especializando demais, ou seja, *overfitting*.

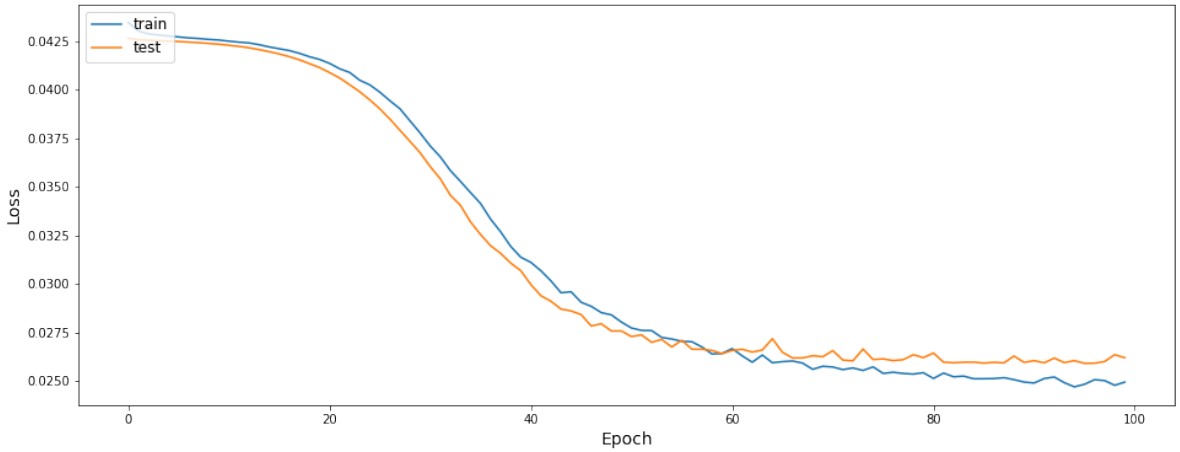

Figura 17 – Gráfico de treinamento do autoencoder completamente conectado

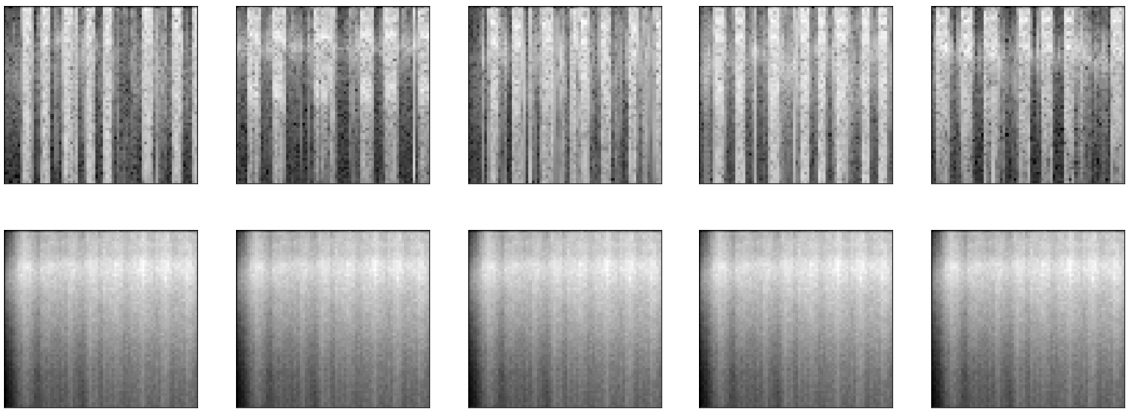

Figura 18 – Conjunto de imagens de teste (acima) e respostas de predição do autoencoder de camadas densas (abaixo)

Após a obtenção destes resultados, foram tentadas algumas outras arquiteturas utilizando apenas camadas densas no autoencoder, mas os resultados foram sempre similares, com o modelo convergindo para para a solução que minimiza o erro do treino e não generaliza o modelo. Reduzir a complexidade do modelo ao diminuir o número de camadas e/ou o número de épocas durante o treino fez com que a rede respondesse com resultados muito diferentes da entrada. Desta forma, conclui-se que uma rede neural densa não possui a capacidade para extração de *features* de maneira aceitável.

Como alternativa à rede neural densa, que analisa todas as entradas de cada camada de maneira global, optou-se por estudar o desempenho das redes neurais convolucionais para a extração de *features*, que leva em consideração informações espaciais de uma matriz bi-dimensional de dados, de forma que o posicionamento de cada valor na matriz se torna relevante.

Primeiramente, definiu-se quatro arquiteturas de autoencoders convolucionais que seriam utilizados, três deles compostos somente por camadas convolucionais e um com algumas camadas densas na parte mais interna. Do menor número de *features* para o maior, os autoencoders citados possuem um *bottleneck* de 128, 272, 544 e 1088 neurônios, cada um permitindo a extração deste

mesmo número de *features*. Para exemplificação, a arquitetura do autoencoder de 128 *features* pode ser visualizada no Listing 2, com as demais arquiteturas seguindo estruturas similares, podendo ser encontradas no repositório github do projeto (FAYAN, R, 2018).

A fim de evitar a convergência para um único resultado ótimo como no caso anterior, utilizou-se nos autoencoders convolucionais a técnica de *Denoising Autoencoder* descrita no no capítulo 2.4.2, inserindo 10% de ruído aos espectrogramas. Alguns dos hiperparâmetros escolhidos para o treinamento também mudaram, seguem eles:

- Função de perda: Erro quadrático médio
- Otimizador: Adadelta
- *Batch size*: 25
- Épocas: 1000
- *Callbacks*: *EarlyStopping* e *ModelCheckpoint*

Realizado o treinamento dos modelos, foram obtidos resultados muito superiores aos obtidos com o autoencoder completamente conectado. As Figuras 19, 20, 21 e 22 representam os resultados dos autoencoders de 128, 272, 544 e 1088 características respectivamente.

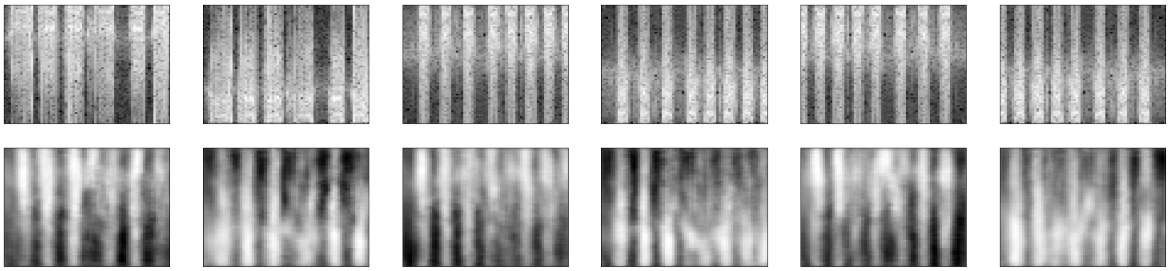

Figura 19 – Conjunto de imagens de teste e respostas de predição do autoencoder convolucional de 128 características

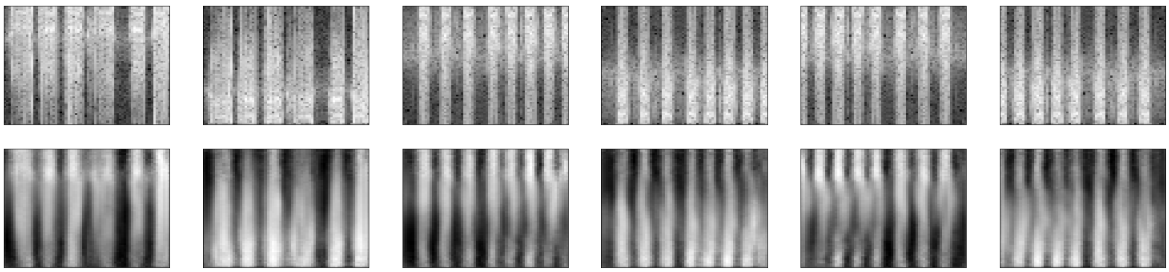

Figura 20 – Conjunto de imagens de teste e respostas de predição do autoencoder convolucional de 272 características

Por último, foram criados mais 3 autoencoders convolucionais para utilizar como entrada os espectrogramas dos sinais compostos pelos testes TUG concatenados, como na Figura 15, utilizando a mesma técnica de denoising autoencoder e mesmos hiperparâmetros de treinamento. Os *bottlenecks* destes autoencoders possuem 128, 250 e 500 neurônios para extração de *features*. As Figuras 20, 21 e 22 representam seus resultados.

Listing 2: Arquitetura do Autoencoder Convolucional com camadas densas no bottleneck

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 64, 68, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 68, 16)        416
_________________________________________________________________
batch_normalization_1 (Batch (None, 64, 68, 16)        64
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 34, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 34, 4)         1604
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 34, 4)         16
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 17, 4)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 17, 1)         101
_________________________________________________________________
flatten_1 (Flatten)          (None, 272)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               34944
_________________________________________________________________
dense_2 (Dense)              (None, 272)               35088
_________________________________________________________________
reshape_1 (Reshape)          (None, 16, 17, 1)         0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 16, 17, 4)         104
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 17, 4)         16
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 32, 34, 4)         0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 32, 34, 16)        1616
_________________________________________________________________
batch_normalization_4 (Batch (None, 32, 34, 16)        64
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 64, 68, 16)        0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 64, 68, 1)         401
=================================================================
Total params: 74,434
Trainable params: 74,354
Non-trainable params: 80
```

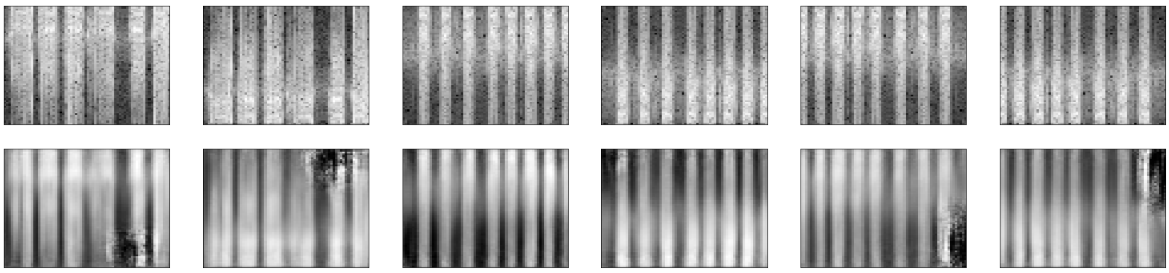

Figura 21 – Conjunto de imagens de teste e respostas de predição do autoencoder convolucional de 544 características

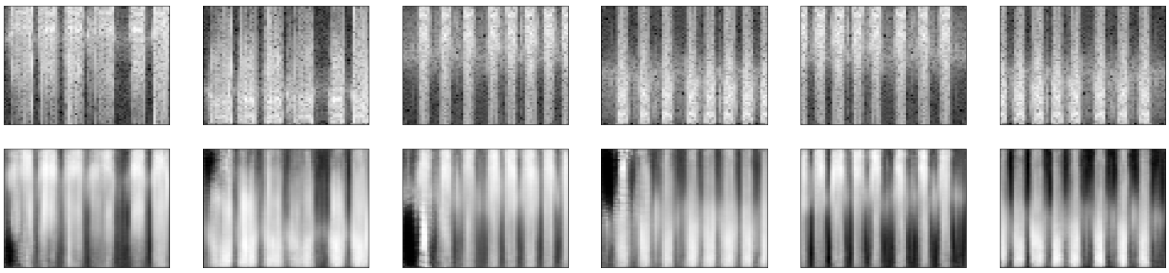

Figura 22 – Conjunto de imagens de teste e respostas de predição do autoencoder convolucional de 1088 características

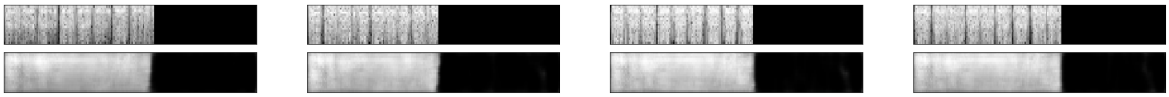

Figura 23 – Conjunto de imagens de teste e respostas de predição do autoencoder de TUGs concatenadas de 128 características

Um sumário das métricas de perda utilizando erro quadrático médio obtidas na reconstrução dos espectrogramas dos conjuntos está representado na Tabela 1.

Observando a tabela, percebemos duas coisas. Primeiro que, em geral, quanto maior o número de *features*, menor o valor do erro quadrático médio, o que é de se esperar pois a imagem tem de ser reconstruída a partir de um conjunto maior de informações. Segundo, os resultados obtidos com o autoencoder completamente conectado foram muito inferiores aos obtidos através

Tabela 1 – Sumário das métricas de perda para os autoencoders implementados

(Sinal - Características)	Loss	Validation Loss
Todo Sinal - 128 Características	0.0122	0.0170
Todo Sinal - 272 Características	0.0091	0.0108
Todo Sinal - 544 Características	0.0068	0.0073
Todo Sinal - 1088 Características	0.0060	0.0062
TUGs Concatenados - 128 Características	0.0107	0.0105
TUGs Concatenados- 250 Características	0.0065	0.0057
TUGs Concatenados - 500 Características	0.0061	0.0050
Todo Sinal (Camadas Densas) - 80 Características	0.0250	0.0262

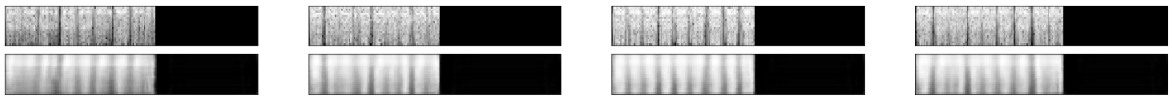

Figura 24 – Conjunto de imagens de teste e respostas de predição do autoencoder de TUGs concatenadas de 250 características

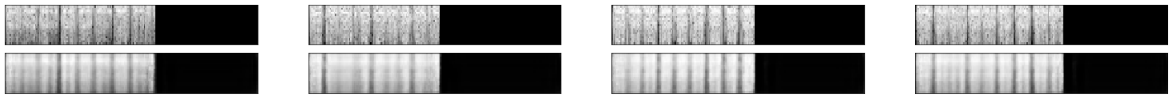

Figura 25 – Conjunto de imagens de teste e respostas de predição do autoencoder de TUGs concatenadas de 500 características

das redes convolucionais, com um valor de perda muito acima dos demais e resultados discutidos anteriormente nas Figuras 17 e 18. Por isso, decidiu-se não incluir seu *encoder* nas análises posteriores.

3.2.4 Aplicação de técnicas de aprendizado de máquina para classificação dos pacientes

Com os extratores de *features* implementados e treinados, ou seja, as camadas de encoding dos autoencoders, a próxima etapa é implementar os algoritmos de aprendizado de máquina para classificação dos pacientes entre caidores e não caidores através da *features* extraídas dos espectrogramas. Para análise e comparação dos resultados, foram selecionados quatro algoritmos: *Suport Vector Machines*, *Random Forests*, *Ada Boosting* e *GradientBoosting*. Uma descrição breve de tais algoritmos pode ser encontrada no capítulo 2.6.

Para o treino e predição dos classificadores SVM, *Ada Boosting* e *Gradient Boosting*, aplicou-se o método de *Principal Component Analysis* (PCA), descrito no capítulo 2.5, ao conjunto de características extraídas para cada paciente, reduzindo a dimensionalidade do espaço de características mas ao mesmo tempo mantendo grande parte da variância e representatividade dos dados originais. Foram escolhidos os valores de 16, 32 e 64 componentes para a aplicação do PCA a fim de analisar o impacto de diferentes valores de variância da base de dados original sob os resultados de classificação a serem obtidos.

Para a implementação do código em Python, utilizou-se a ferramenta scikit-learn (PEDREGOSA *et al.*, 2011), que implementa inúmeras funções para o desenvolvimento de algoritmos focados em aprendizado de máquina e foram essenciais para os estudos realizados.

Primeiramente, foi feita uma análise de representatividade dos dados ao aplicar o método PCA sob os diferentes espaços de *features*, desenhou-se os gráficos de variância acumulada e por componente, conhecido como *Scree Plot*. As Figuras 26, 27, 28 e 29 representam gráficos obtidos para os *datasets* com espaço de 128, 277, 544 e 1088 características, respectivamente.

Analisando os gráficos de variância, vemos que para as bases de 272, 544 e 1088

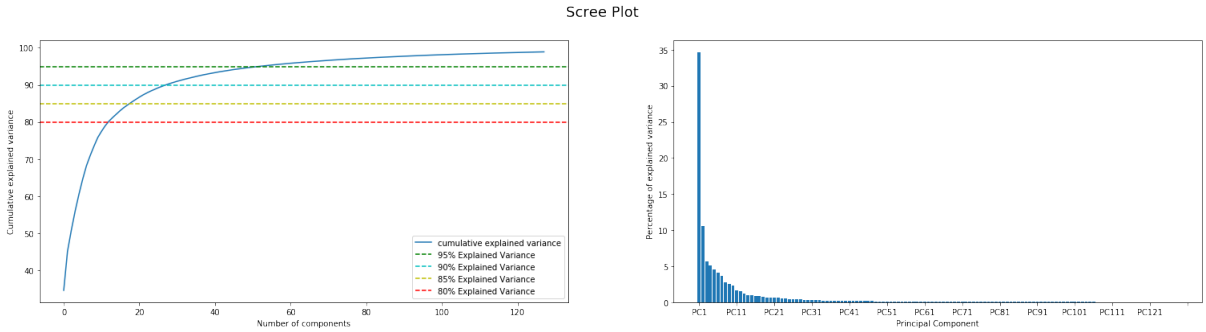

Figura 26 – Variância cumulativa e por componente da transformação PCA ao *dataset* de 128 *features*

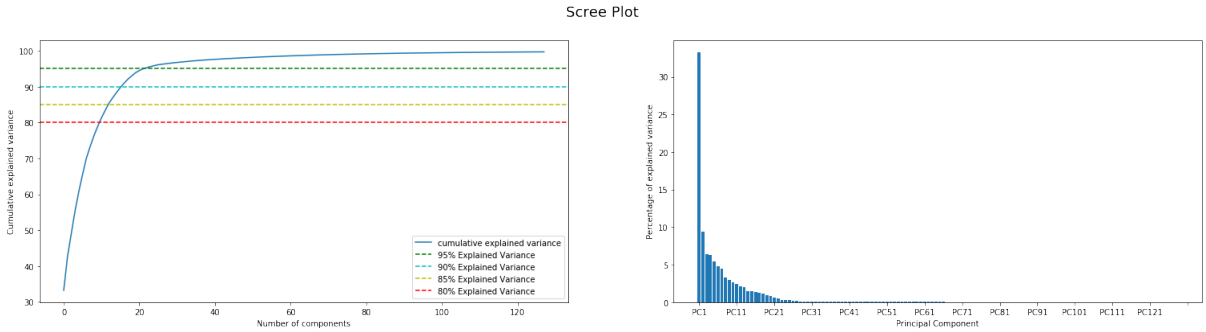

Figura 27 – Variância cumulativa e por componente da transformação PCA ao *dataset* de 272 *features*

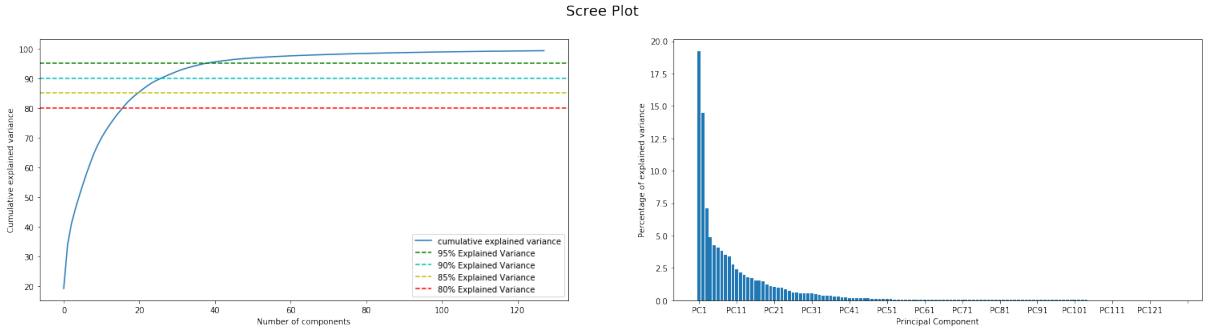

Figura 28 – Variância cumulativa e por componente da transformação PCA ao *dataset* de 544 *features*

características, a variância cumulativa representada pelos primeiros N componentes tende a diminuir. Isso condiz esperado, pois normalmente são necessários mais componentes para obter uma mesma representatividade quando se aumenta a dimensionalidade espaço de características. No entanto, vemos que o conjunto de 128 características, contrário ao esperado, é bem menos representado pelos primeiros componentes do que a base de 272 dimensões. Isso nos indica que o *encoder* de 128 features não conseguiu extrair bem as informações que melhor representam as características contidas nos espectrogramas.

Ao total, teremos uma combinação de 70 modelos a serem treinados. Para cada um dos 7 *encoders* implementados, teremos a aplicação dos classificadores SVM, *Ada Boosting* e *Gradient Boosting*, cada um desses 3 aplicados em conjunto com PCA de 16, 32 ou 64 componentes. Além destes, utilizou-se também o classificador Random Forest sem PCA, pois este método lida melhor

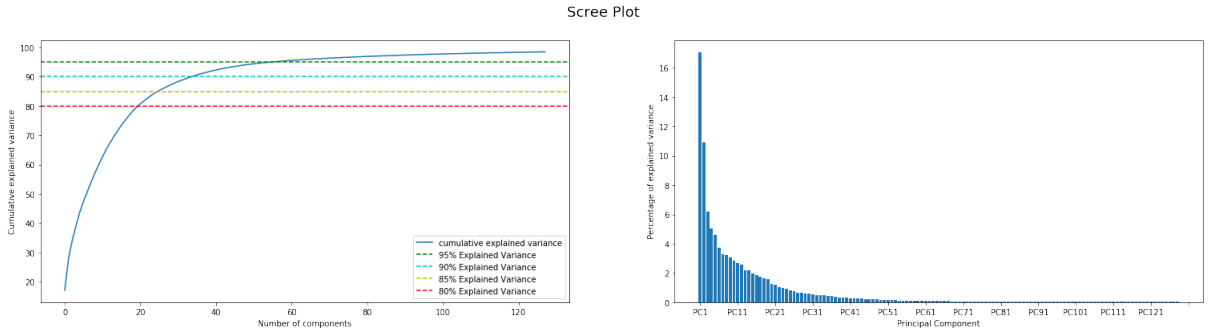

Figura 29 – Variância cumulativa e por componente da transformação PCA ao *dataset* de 1088 *features*

com dados não normalizados, totalizando 10 modelos por *encoder*. Para os classificadores *Ada Boosting* e *Gradient Boosting* escolheu-se utilizar a estrutura de árvore de decisão como seus estimadores.

Adicionalmente, para fins de comparação com o conjunto de características extraídas, utilizou-se os mesmos quatro classificadores, sem a utilização de PCA, sob o conjunto de 9 características extraídas a partir dos dados de aceleração temporais, obtidos através dos métodos descritos por (PONTI *et al.*, 2017), totalizando 74 modelos.

O código em Python implementado que realiza o treinamento dos modelos descritos e obtém as métricas de predição para cada uma das combinações de *encoders*, PCAs e classificadores, pode ser descrito de forma resumida através das seguintes etapas:

1. Separação do conjunto de espectrogramas em treino e teste
2. Aplicação de *Data Augmentation* sob o conjunto de treino
3. Extração das N *features* utilizando o autoencoder associado
4. Padronização dos dados utilizando *StandardScaler* (exceto para Random Forest)
5. Aplicação da transformação PCA para M Componentes (exceto para Random Forest)
6. Separação do conjunto de treino em 4-Fold para validação cruzada dos dados de treinamento
7. Busca dos melhores hiperparâmetros para cada classificador utilizando *GridSearch*
8. Obtenção do melhor *threshold* para classificação dos dados a partir da probabilidade de predição utilizando o índice de Youden J (YOUDEN, 1950)
9. Avaliação do classificador, coletando métricas de desempenho através de predições sob o conjunto de treino

Estes procedimentos foram implementados em oito arquivos separados, sete para as arquiteturas de *encoders* utilizadas e um para as *features* originais.

Tabela 2 – Melhores 10 resultados com base na métrica ROC AUC

	AUC_Score	F1_Score	Average_Precision	Balanced_Accuracy
GradBoosting_TUGs_250_64	0.892	0.889	0.883	0.9
SVM_TUGs_500_32	0.862	0.833	0.596	0.923
AdaBoosting_TUGs_128_32	0.831	0.727	0.805	0.823
SVM_TUGs_128_64	0.8	0.667	0.633	0.808
AdaBoosting_TUGs_250_64	0.785	0.667	0.716	0.762
GradBoosting_272_16	0.768	0.643	0.526	0.789
AdaBoosting_TUGs_128_16	0.754	0.625	0.491	0.769
GradBoosting_272_32	0.743	0.581	0.553	0.736
AdaBoosting_272_16	0.739	0.571	0.524	0.721
SVM_TUGs_128_16	0.738	0.625	0.566	0.769

Tabela 3 – Piores 10 resultados com base na métrica ROC AUC

	AUC_Score	F1_Score	Average_Precision	Balanced_Accuracy
SVM_272_64	0.504	0.4	0.326	0.593
AdaBoosting_1088_32	0.507	0.455	0.308	0.625
RandomForest_1088	0.507	0.385	0.304	0.554
SVM_TUGs_250_64	0.508	0.5	0.358	0.615
GradBoosting_TUGs_500_64	0.508	0.5	0.442	0.631
RandomForest_OriginalFeatures	0.514	0.444	0.363	0.629
SVM_1088_16	0.518	0.462	0.281	0.593
GradBoosting_128_64	0.518	0.421	0.31	0.611
AdaBoosting_TUGs_128_64	0.523	0.5	0.339	0.646
AdaBoosting_TUGs_500_64	0.523	0.533	0.326	0.669

3.3 Resultados Obtidos

Os resultados de predição para cada um dos modelos foram coletados e agregados em uma estrutura de dados *DataFrame* construída utilizando a biblioteca Pandas (MCKINNEY, 2011) e posteriormente salvos ao arquivo `<results.csv>`, que pode ser encontrado no repositório github do projeto (FAYAN, R, 2018). As Tabelas 2 e 3 representam uma parcela destes resultados, listando os 10 melhores e piores resultados com base na área abaixo da curva ROC, respectivamente, e a Tabela 4 lista os 10 melhores resultados com base na métrica F1 Score. As quatro métricas listadas na tabela estão descritas no capítulo 2.7. Cada linha da tabela representa um dos modelos treinados e seus resultados, com a primeira coluna representando os algoritmos e conjuntos de dados utilizados segundo esta nomenclatura:

`<Classificador>_[TUGs]_<CaracterísticasExtraidas>_[ComponentesPCA]`

Onde `<Classificador>` representa um dos 4 algoritmos de aprendizado de máquina utilizado, `[TUGs]` se o espectrograma foi gerado a partir dos testes TUG concatenados, <CaracterísticasExtraidas> indica quantas features foram extraídas pelo autoencoder e `[ComponentesPCA]` o número de componentes utilizados na transformação PCA, se foi utilizado. Campos em colchetes indicam que o campo pode ser nulo.

Ao analisar as tabelas, podemos observar alguns pontos:

Tabela 4 – Melhores 10 resultados com base na métrica F1 Score

	AUC_Score	F1_Score	Average_Precision	Balanced_Accuracy
GradBoosting_TUGs_250_64	0.892	0.889	0.883	0.9
SVM_TUGs_500_32	0.862	0.833	0.596	0.923
AdaBoosting_TUGs_128_32	0.831	0.727	0.805	0.823
SVM_TUGs_128_64	0.8	0.667	0.633	0.808
AdaBoosting_TUGs_250_64	0.785	0.667	0.716	0.762
AdaBoosting_TUGs_250_32	0.692	0.667	0.616	0.762
GradBoosting_272_16	0.768	0.643	0.526	0.789
GradBoosting_TUGs_128_32	0.723	0.625	0.443	0.769
SVM_TUGs_128_16	0.738	0.625	0.566	0.769
AdaBoosting_TUGs_128_16	0.754	0.625	0.491	0.769

- Os modelos que utilizam os espectrogramas gerados a partir dos testes TUG concatenados apresentaram melhores resultados e aparecem com maior frequência nas Tabelas 2 e 4. Isso está de acordo com o esperado pois o período de repouso presente entre os testes TUG contido no sinal teoricamente não são representativos do perfil de risco de queda do idoso. Esta informação deve estar contida nos dados de aceleração coletados durante a movimentação do paciente.
- Nas Tabelas 2 e 4, o algoritmo de classificação Random Forest não consta nenhuma vez. No entanto na Tabela 3 ele aparece duas vezes. Isto pode ser indício de que o algoritmo não obteve bons resultados de forma geral.
- Nove dos 10 modelos presentes na Tabela 2 estão presentes na Tabela 4. Como os modelos apresentados nas duas tabelas apresentaram métricas de classificação muito boas, é de se esperar que os melhores modelos estejam presentes em ambas as tabelas, pois conforme um modelo se aproxima do modelo perfeito (todas as classificações corretas), a variação entre as diferentes métricas diminui.
- A discrepância entre as métricas obtidas entre melhores e piores classificadores é muito grande. Enquanto os modelos presentes na Tabela 3 se aproximam de um classificador aleatório, com ROC AUC próximo a 0.5, vemos que na Tabela 2 o oposto ocorre, principalmente com o primeiro modelo, que se aproxima de um modelo quase perfeito.

A fim de analisar os resultados obtidos em um escopo mais amplo, desenhou-se o gráfico de médias e valor máximo para cada uma das quatro métricas agregando os resultados de modelos de mesmo *input*, exibido pela Figura 30.

Analisando o gráfico gerado, vemos que para as quatro métricas os modelos que utilizam os espectrogramas dos sinais de aceleração do teste TUG concatenados como entrada apresentaram melhores resultados, tanto de média quanto valor máximo, como discutido em um dos pontos observados. De forma análoga, utilizando somente os modelos que usam este tipo de entrada, desenhamos outro gráfico para visualização das métricas, mas agregando os valores por classificador. O gráfico gerado está representado na Figura 31.

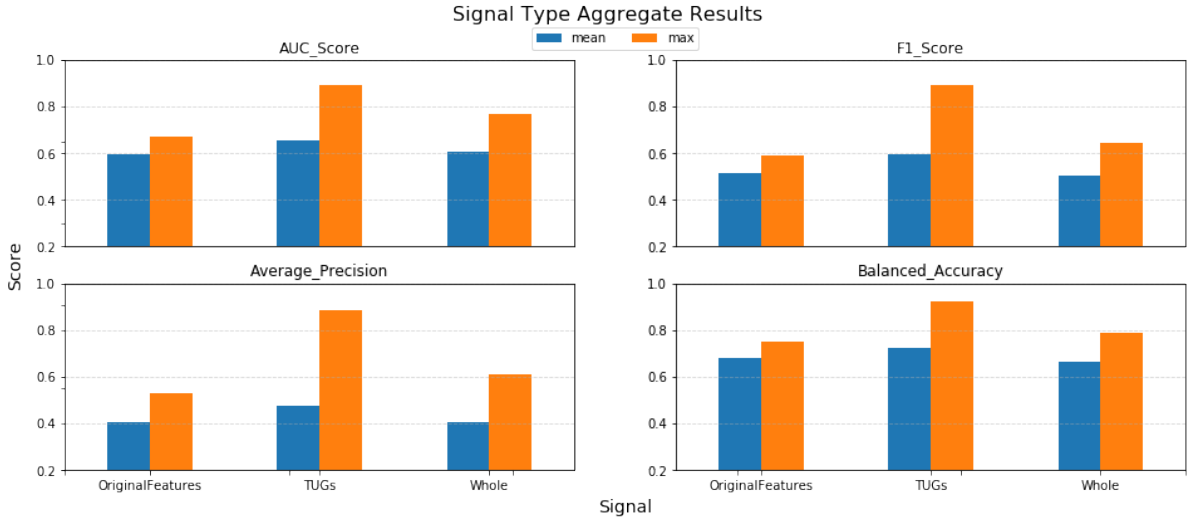

Figura 30 – Sumário das métricas obtidas, agregando os modelos por tipo de entrada

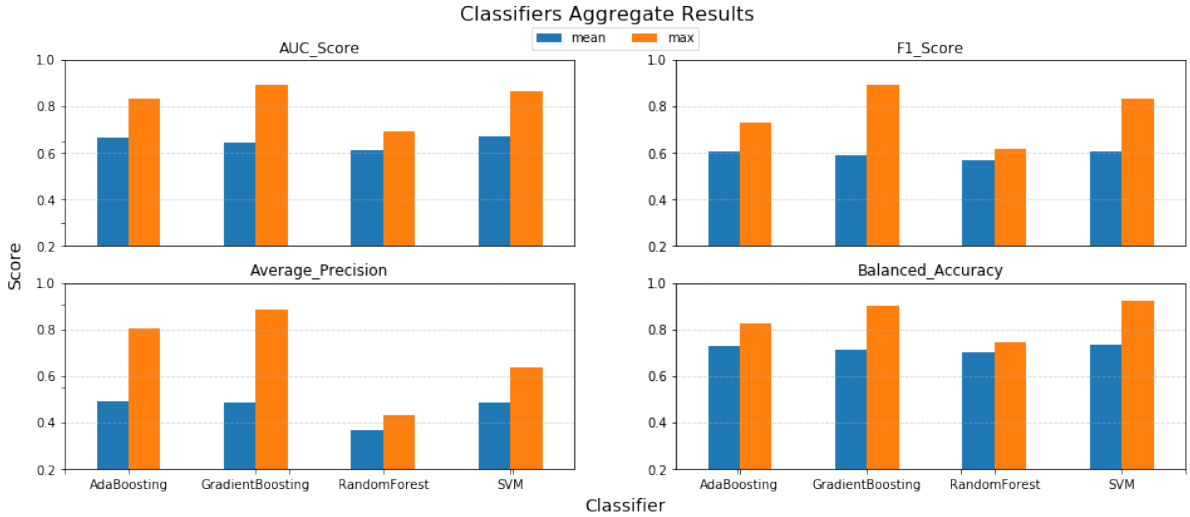

Figura 31 – Sumário das métricas obtidas para os modelos de entrada com espectrogramas dos sinais TUG, agregando por tipo de classificador

Podemos ver através do gráfico que entre os quatro classificadores, o Random Forest foi o qual apresentou os piores resultados, o que condiz com a análise das Tabelas 2, 3 e 4 feita anteriormente. Os algoritmos de *boosting* apresentaram métricas similares, um pouco melhores que o SVM por uma pequena margem. Filtrando os modelos também pelos classificadores de *boosting* e agregando por número de *features*, obtivemos a Figura 32.

Vemos no gráfico gerado que os modelos com 250 características extraídas apresentaram os melhores resultados de predição. Uma possível explicação para isso é que com espaço de 128 características os dados perdem informações relevantes para a diferenciação das classes caidores e não caidores, e com 500 características os algoritmos não possuíram complexidade o suficiente para determinar padrões adequados para classificação devido a alta dimensionalidade dos *inputs*. Por fim, gerando o gráfico filtrando por 250 características e agregando por componentes PCA, obtivemos a Figura 33.

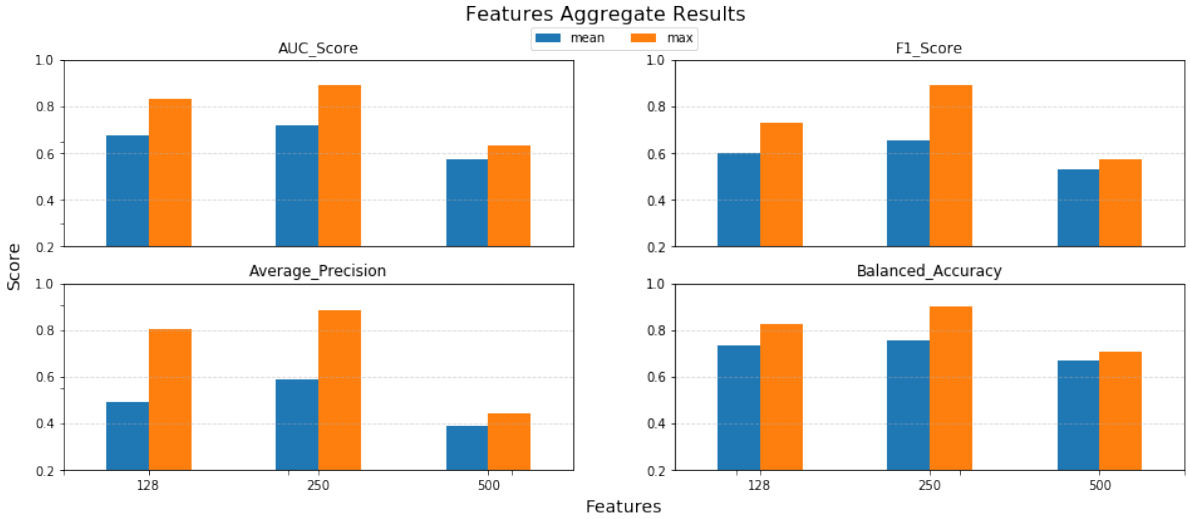

Figura 32 – Sumário das métricas obtidas para os modelos de entrada com espectrogramas dos sinais TUG e classificadores AdaBoosting ou GradientBoosting, agregando por número de features

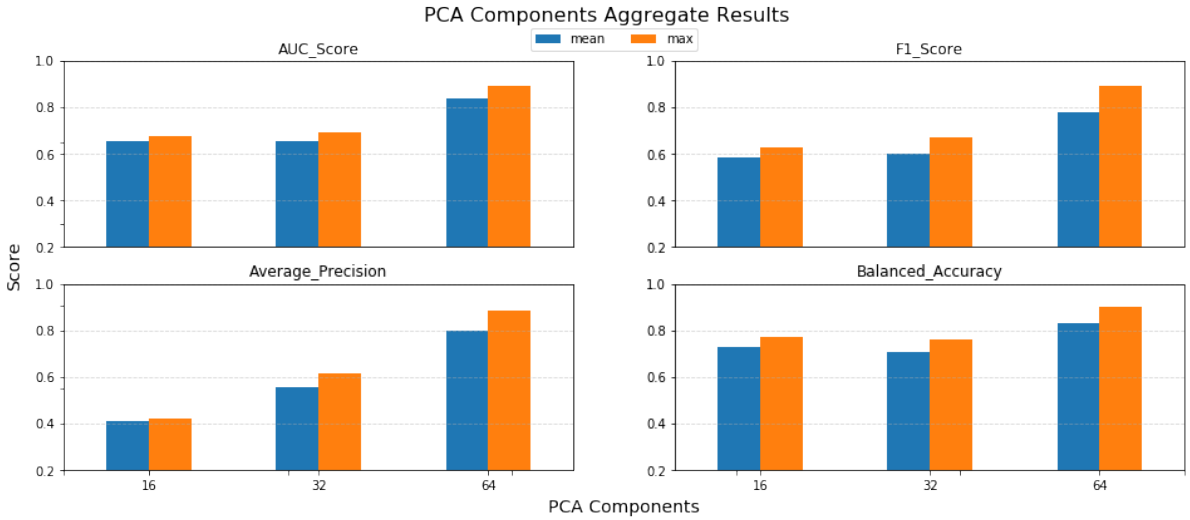

Figura 33 – Sumário das métricas obtidas para os modelos de entrada com espectrogramas dos sinais TUG, classificadores AdaBoosting ou GradientBoosting e 250 features, agregando por número de componentes PCA

Vemos que a aplicação de PCA para 64 componentes apresentou os melhores resultados, e conforme a redução para 32 e 16 componentes é feita os resultados pioram. Deste modo concluímos que, em média, os melhores modelos foram os que utilizaram os espectrogramas dos testes TUG como entrada, classificadores Ada Boosting e Grad Boosting, pois não houve grandes diferenças entre os resultados destes, encoder com extração de 250 características e aplicação de PCA para redução a 64 componentes principais. Analisando as Tabelas 2 e 4, concluímos que isso está correto pois os primeiros e quinto melhores modelos se encaixam nessa descrição.

Para finalizar, decidiu-se inspecionar mais a fundo os três melhores modelos segundo as métricas de classificação ROC AUC e F1 Score. Uma representação visual mais detalhada dos resultados de predição obtidos para estes modelos podem ser visualizados nas Figuras 34, 35 e 36.

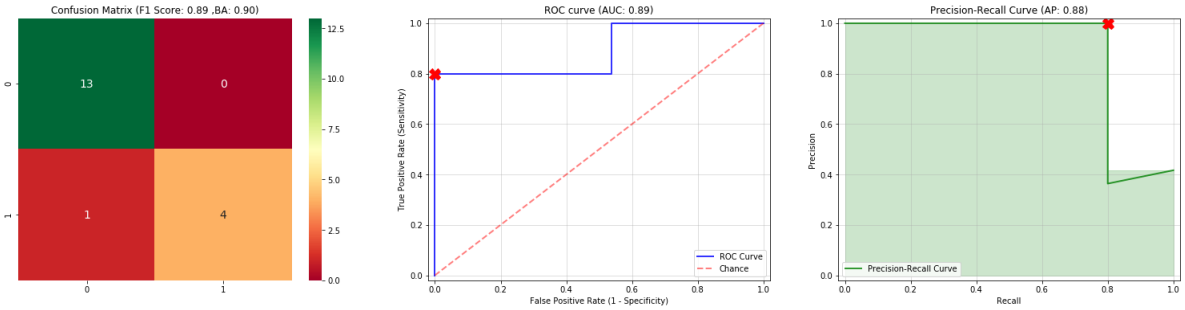

Figura 34 – Matriz de Confusão, Curva ROC e Curva de Precisão-revocação para o classificador Gradient Boosting com sinais TUG, 250 características e 64 componentes PCA

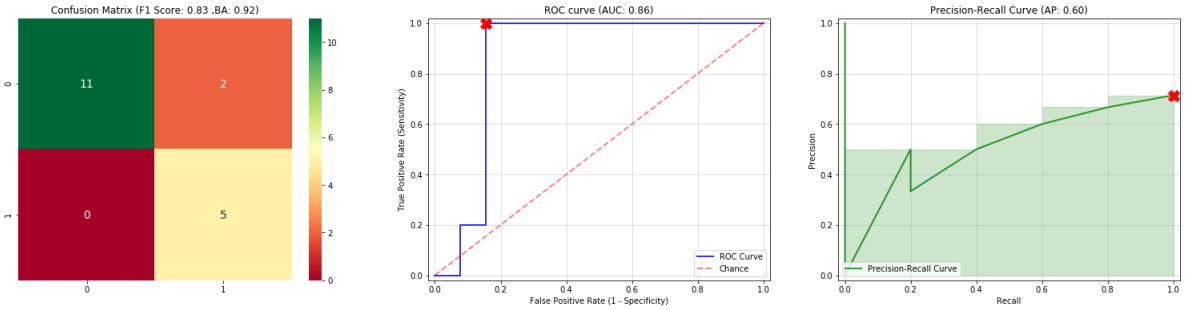

Figura 35 – Matriz de Confusão, Curva ROC e Curva de Precisão-revocação para o classificador SVM com sinais TUG, 500 características e 32 componentes PCA

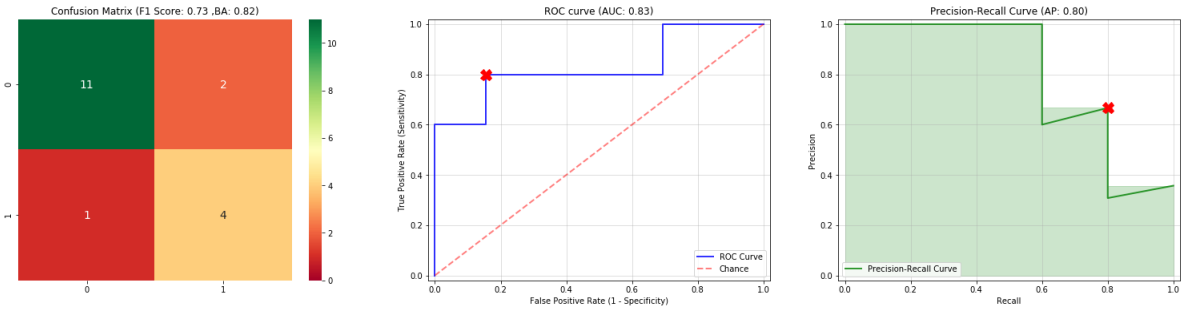

Figura 36 – Matriz de Confusão, Curva ROC e Curva de Precisão-revocação para o classificador Ada Boosting com sinais TUG, 128 características e 32 componentes PCA

Estes três melhores modelos apresentaram ótimos resultados, o primeiro deles se aproximando bastante de um modelo perfeito, com apenas uma classificação incorreta do conjunto de testes de 18 pacientes. No entanto um ponto importante a levar em consideração é justamente o fato de que o número de amostras no conjunto de testes utilizado para coleta das métricas é muito pequeno, contendo apenas 13 caidores e 5 não caidores.

Em casos como esse, quando o conjunto de teste é muito pequeno, para cada acerto ou erro de predição do modelo as métricas de classificação melhoram ou pioram muito, gerando um desvio padrão alto. Neste estudo, como foram treinados 74 modelos diferentes, não é de se surpreender se alguns poucos modelos tenham obtido melhores resultados por causa disso, principalmente quando se há apenas cinco amostras em uma das classes. Pesa ainda o fato de que são poucos exemplos de treinamento utilizados, o que reduz a generalidade das conclusões.

No entanto, dadas as limitações foram utilizadas todas as técnicas conhecidas para minimizar problemas, incluindo *data augmentation*, uso de algoritmos com viés restrito como o SVM em comparação com algoritmos de combinação de classificadores, além de validação cruzada para análise dos resultados, a qual é considerada um bom estimador do risco empírico dos classificadores (DEVROYE; GYÖRFI; LUGOSI, 2013). Ainda, os bons resultados obtidos para um subconjunto de arquiteturas e classificadores demonstra o potencial dos métodos aqui investigados, e aponta boas possibilidades de trabalhos futuros utilizando espectrogramas e autoencoders.

3.4 Dificuldades e Limitações

Como alguém com pouco conhecimento prévio sobre as diferentes áreas de processamento digitais de sinais e aprendizado de máquina, houve uma grande curva de aprendizado nas etapas iniciais e intermediárias do projeto. Foi necessário muita pesquisa sobre os diversos conceitos que não eram familiares, e como consequência houveram atrasos no cronograma planejado.

Em relação a área de *machine learning* especificamente, foi surpreendente a complexidade de muitas etapas envolvidas no treinamento de modelos de rede neurais e também nas etapas de pré-processamento dos dados e avaliação dos resultados. Embora os conceitos sejam simples de compreender, é preciso se atentar a uma série de operações como o *fine-tuning* de vários hiperparametros, padronização dos dados, separação adequada dos conjuntos de teste, escolha das métricas certas para avaliação dos modelos, etc. Foi encontrada muita dificuldade com alguns destes itens, principalmente quando era necessário algumas noções de estatística, como por exemplo, para os estudos das operações de redução de dimensionalidade como PCA.

No entanto, as maiores barreiras para a obtenção dos resultados e o desenvolvimento do projeto foi a pequena quantidade de dados fornecida e o tempo e recursos computacionais necessários para as diversas operações de *machine learning* realizadas. A pequena quantidade de dados, principalmente de idosos classificados como caidores, dificultou bastante a validação dos resultados obtidos, o que levantou o questionamento de se esses resultados e comportamentos são esperados neste caso ou se durante o desenvolvimento dos códigos foram cometidos alguns erros. E durante o treinamento dos autoencoders e algoritmos de classificação, como SVM e Random Forest, devido às restrições de pode computacional, foi necessário um pouco de conservadorismo quanto à exploração das arquiteturas e hiperâmetros, pois cada execução chegava a demorar até 1 hora, muitas vezes retornando resultados insatisfatórios.

3.5 Considerações Finais

Embora os resultados obtidos através dos melhores modelos sejam muito promissores, é necessário exercer cautela pois como o conjunto de teste onde as métricas foram obtidas é muito pequeno, a avaliação do modelo está sujeita a grandes variações. Não somente a base de dados utilizada é pouco representativa do espaço de amostras que podem ser utilizadas para aplicação na área de saúde, mas também as métricas obtidas dificilmente são representativas da eficiência real de tais classificadores com dados desconhecidos.

No entanto, acredito que muito do que foi desenvolvido possa ser reutilizado futuramente para refinação dos modelos e validação dos estudos realizados. Devido as limitações mencionadas na seção anterior e à restrição de tempo do projeto, não foi possível explorar toda gama de possibilidades que ainda não foi estudada para aplicação da predição do risco de queda. Com uma base de dados maior, acima de centenas, e mais recursos computacionais disponíveis, uma série de outras investigações pode ser feita a fim de contribuir para o desenvolvimento de pesquisas na área.

Capítulo 4

CONCLUSÃO

4.1 Contribuições

Ao término da interpretação dos resultados obtidos, concluiu-se que estes resultados, embora promissores, devem ser estudados mais a fundo e revisados para melhor validar a representatividade das métricas obtidas. As técnicas aplicadas aparentemente se mostraram adequadas como abordagem para o problema. Infelizmente, sem uma base de dados maior, a validação dos resultados e métodos aplicados se torna algo difícil de se fazer.

Através da realização do projeto, foi possível compreender e atuar em diversas etapas envolvidas no estudo e desenvolvimento de uma solução em aprendizado de máquina. Desde a obtenção e pré-processamento dos dados até a aplicação de algoritmos de classificação e avaliação das métricas, foi interessante participar de um estudo que envolvesse diferentes áreas e conhecimentos.

Devido ao número de etapas e complexidade do problema, o trabalho me exigiu uma certa organização para a aplicação dos métodos e manipulação dos dados, assim como conhecimentos de não só como realizar uma determinada operação mas compreender como ela funciona e interpretar seus resultados. A realização deste projeto despertou meu interesse pela área de *Data Science* e *Machine Learning*, não somente pela complexidade dos problemas mas também pela aplicabilidade em diversos contextos, me motivando a me aprofundar na área.

4.2 Relacionamento entre o Curso e o Projeto

Acredito que o projeto desenvolvido tenha bastante relação com o curso de Engenharia de Computação. Alta interdisciplinariedade do projeto, assim como a necessidade de uma boa base estatística e de programação e também o aspecto exploratório exigido fez muito sentido para mim como estudante de engenharia de computação. No entanto, senti em vários momentos uma falta base teórica nas áreas de ciência de dados e aprendizado de máquinas.

Em nosso curso, essas matérias não são apresentadas na grade obrigatória, cuja decisão eu concordo concordo pois são áreas específicas, no entanto não foi apresentada nenhuma matéria relacionada a ciência de dados como optativa eletiva e apenas uma relacionada a algoritmos

evolutivos na área de aprendizado de máquina. A maior parte do conhecimento específico exigido neste projeto não foi obtido durante a curso de graduação e sim por esforço próprio.

4.3 Considerações sobre o Curso de Graduação

Acredito o curso de Engenharia de Computação é um curso extremamente interdisciplinar, com uma vasta gama de áreas de atuação e especialização. No entanto, sinto que na USP - São Carlos, o curso sofre pois ele tenta inserir quase todas as matérias obrigatórias dos cursos de Engenharia de Elétrica e Ciência da Computação em um.

Atualmente o curso possui 85% do total de horas aula pertencentes à grade obrigatória, restringindo a escolha dos alunos para decidir as áreas que deseja se especializar. Como consequência, muitos alunos passam boa parte da graduação atendendo a aulas que provavelmente não serão relevantes para seu futuro acadêmico ou no mercado. Sou da opinião que aumentar o número de matérias optativas exigidas, reduzindo as obrigatórias, forneceria uma melhor formação para as áreas de interesse pessoais de cada um.

Acredito também que faltam incentivos para que os alunos do curso participem de atividades extracurriculares. Passei maior parte da minha graduação participando de secretarias acadêmicas, grupos de robótica, empresas juniores e semanas do curso. Eu acredito que essas experiencias fornecem um crescimento pessoal muito forte ao aluno, pois ele tem se adaptar a um ambiente novo e compartilhado, com metas e exigências e com problemas reais, fornecendo a oportunidade de aplicar muitos dos conhecimentos vistos somente em teoria na sala de aula.

Por fim, acho que a grande maioria dos professores fazem um trabalho excepcional em sala de aula, no entanto, ainda vejo alguns poucos professores que ano após ano apresentam as mesmas aulas ruins, o mesmo comportamento negativo e os mesmos problemas de forma geral, sem que nenhuma atitude seja tomada. Em minha opinião deveria haver um maior engajamento pelas coordenadorias e institutos em solucionar quaisquer impedimentos que possam estar afetando a formação dos alunos.

O curso de Engenharia de Computação, em conjunto com as as experiências que tive na USP - São Carlos, me fizeram a pessoa que sou hoje, e por isto eu sou muito grato. Embora ainda haja muitos pontos a ser melhorados, acredito que ele continua sendo um dos melhores.

4.4 Trabalhos Futuros

Os possíveis estudos que poderiam ser realizados para a expansão das análises de risco de queda através de dados de acelerometria são vários, entre eles:

- Revisar os métodos implementados no projeto e buscar uma melhor explicação da alta variação dos resultados obtidos, assim como uma refatoração dos algoritmos para possível

uso posterior em outras aplicações e projetos

- Estudo de diferentes arquiteturas de autoencoders para buscar melhorias na extração de características, assim como o estudo de outros algoritmos de extracão de *features* como *Recursive Feature Elimination* e métodos por seleção univariada
- Aplicar métodos similares aos realizados, mas aos sinais de aceleração originais e também ao sinal em frequência após a aplicação da transformada de Fourier. Deste modo, a dimensionalidade do problema diminuirá pois serão analisados sinais bi-dimensionais ao contrário de imagens de espectrogramas.
- Investigar a aplicabilidade de redes neurais recorrentes para análise dos dados de aceleração no tempo e, caso sejam obtidas bases de dados maiores, redes neurais adversariais.

REFERÊNCIAS

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I. J.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JÓZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANÉ, D.; MONGA, R.; MOORE, S.; MURRAY, D. G.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P. A.; VANHOUCKE, V.; VASUDEVAN, V.; VIÉGAS, F. B.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **CoRR**, abs/1603.04467, 2016. Disponível em: <http://arxiv.org/abs/1603.04467>. Citado na página 27.

ALEXANDRE, T. S.; MEIRAII, D. M.; RICOIII, N. C.; MIZUTA, S. K. Accuracy of timed up and go test for screening risk of falls among community-dwelling elderly. **Brazilian Journal of Physical Therapy**, v. 16, 2012. Citado na página 14.

ALMEIDA, S. T. de; SOLDERA, C. L. C.; CARLI, G. A. de; GOMES, I.; RESENDE, T. de L. Análise de fatores extrínsecos e intrínsecos que predispõem a quedas em idosos. **Rev. Assoc. Med. Bras**, 2012. Citado na página 11.

BET, P. **Mobile inertial sensors and fall risk prediction in community-dwelling elderly**. 2018. Repositório Github. Disponível em: <https://github.com/patriciabet/mobile-fall-screening>. Acesso em: 06 novembro 2018. Citado 2 vezes nas páginas 23 e 49.

BET, P. **Sensores inerciais moveis no rastreio e predição de risco de queda em idosos**. Dissertação (Mestrado) — Universidade Federal de São Carlos, 2018. Citado na página 23.

BREIMAN, L. Random forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001. Citado na página 20.

CAVALLARI, G. B.; RIBEIRO, L. S.; PONTI, M. A. Unsupervised representation learning using convolutional and stacked auto-encoders: a domain and cross-domain feature space analysis. In: **SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI 2018)**. [S.l.: s.n.], 2018. Citado na página 15.

CHOLLET, F. *et al.* **Keras**. 2015. <https://keras.io>. Citado na página 27.

DEVROYE, L.; GYÖRFI, L.; LUGOSI, G. **A probabilistic theory of pattern recognition**. [S.l.]: Springer Science & Business Media, 2013. v. 31. Citado na página 42.

FAYAN, R. **Mobile inertial sensors and fall risk prediction in community-dwelling elderly**. 2018. Repositório Github, *forked* de (BET, 2018a). Disponível em: <https://github.com/rfayan/mobile-fall-screening>. Acesso em: 06 novembro 2018. Citado na página 23.

FAYAN, R. **Projeto de TCC: Análise em tempo e frequência para extração de características de sinais de aceleração aplicado ao estudo da marcha**. 2018. Repositório Github.

Disponível em: <https://github.com/rfayan/TUG_Fall_Analysis>. Acesso em: 07 novembro 2018. Citado 2 vezes nas páginas 31 e 37.

FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. **Journal of computer and system sciences**, Elsevier, v. 55, n. 1, p. 119–139, 1997. Citado na página 20.

FRIEDMAN, J. H. Greedy function approximation: a gradient boosting machine. **Annals of statistics**, JSTOR, p. 1189–1232, 2001. Citado na página 20.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A.; BENGIO, Y. **Deep learning**. [S.l.]: MIT press Cambridge, 2016. v. 1. Citado na página 16.

HASTIE, T.; ROSSET, S.; ZHU, J.; ZOU, H. Multi-class adaboost. **Statistics and its Interface**, International Press of Boston, v. 2, n. 3, p. 349–360, 2009. Citado na página 20.

HEARST, M.; DUMAIS, S.; OSMAN, E.; PLATT, J.; SCHOLKOPF, B. Support vector machines. **Intelligent Systems and their Applications, IEEE**, v. 13, p. 18 – 28, 08 1998. Citado na página 19.

JONES, E.; OLIPHANT, T.; PETERSON, P. *et al.* **SciPy: Open source scientific tools for Python**. 2001–. [Online; accessed <today>]. Disponível em: <http://www.scipy.org/>. Citado na página 24.

MCKINNEY, W. pandas: a foundational python library for data analysis and statistics. 2011. Citado na página 37.

MUS, D. Audio feature extraction with restricted boltzmann machines. **Leiden Institute of Advanced Computer Science**, 2011. Citado na página 11.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011. Citado na página 34.

PHELAN, E. A.; MAHONEY, J. E.; VOIT, J. C.; STEVENS, J. A. Assessment and management of fall risk in primary care settings. **Medical Clinics of North America**, 2015. Citado na página 11.

PONTI, M. A.; BET, P.; OLIVEIRA, C. L.; CASTRO, P. C. Better than counting seconds: Identifying fallers among healthy elderly using fusion of accelerometer features and dual-task timed up and go. **PLoS one**, 2017. Citado 4 vezes nas páginas 11, 14, 23 e 36.

PONTI, M. A.; COSTA, G. B. P. da. Como funciona o deep learning. 2017. Citado 2 vezes nas páginas 5 e 18.

PONTI, M. A.; RIBEIRO, L. S. F.; NAZARE, T. S.; BUI, T.; COLLOMOSSE, J. Everything you wanted to know about deep learning for computer vision but were afraid to ask. In: IEEE. **2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)**. [S.l.], 2017. p. 17–41. Citado na página 15.

ROMANOVAS, M. **Schematic diagram of the TUG test showing the subcomponents of the test**. 2012. [Online; accessed <today>]. Disponível em: <https://www.researchgate.net/figure/Schematic-diagram-of-the-TUG-test-showing-the-subcomponents-of-the-test_fig1_261496694>. Citado 2 vezes nas páginas 5 e 13.

SHUMWAY-COOK, A.; BRAUER, S.; WOOLLACOTT, M. Predicting the probability for falls in community-dwelling older adults using the timed up go test. **Physical Therapy**, v. 80, p. 896–903, 2013. Citado na página 13.

SMITH, J. O. **The Short-Time Fourier Transform**. 2018. Disponível em: <https://ccrma.stanford.edu/~jos/sasp/Short_Time_Fourier_Transform.html>. Acesso em: <today>. Citado na página 14.

SOUZA, G. M. de. **Consequências causadas pelas quedas à qualidade de vida do idoso**. Dissertação (Mestrado) — Universidade Federal de Minas Gerais, 2011. Citado na página 11.

VINCENT, P.; LAROCHELLE, H.; LAJOIE, I.; BENGIO, Y.; MANZAGOL, P.-A. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. **Journal of Machine Learning Research**, v. 11, n. Dec, p. 3371–3408, 2010. Citado na página 18.

World Health Organization. **WHO global report on falls prevention in older age**. [S.l.], 2007. Citado na página 11.

YOUDEN, W. J. Index for rating diagnostic tests. **Cancer**, 1950. Citado na página 36.

ZAMBOM, A. Z.; DIAS, R. **A Review of Kernel Density Estimation with Applications to Econometrics**. — Universidade Estadual de Campinas, 2012. Citado na página 26.