Sistema FIEB | SENAI CIMATEC
PELO FUTURO DA INOVAÇÃO

CENTRO UNIVERSITÁRIO CIMATEC

ESPECIALIZAÇÃO EM AUTOMAÇÃO, CONTROLE E ROBÓTICA

ANÁLISE DE SINAIS EEG BCI 200 PARA CLASSIFICAÇÃO DE INTENÇÃO DE MOVIMENTOS: uma abordagem prática.

Iuri de Araujo Sampaio[1]

Oberdan Rocha Pinheiro[2]

RESUMO

A Interfaces Cérebro Computador (ICC), ou Interface Homem-Máquina (ICM), do inglês: Brain Computer Interfaces (BCI), vem se tornando um campo vital da engenharia e computação biomédica. Ela utiliza sinais elétricos obtidos a partir de exames de eletroencefalograma (EEG) para fornecer tecnologias assistivas (AT) para humanos. O objetivo da ICM é interpretar a atividade cerebral em forma digital e atuar com comandos em dispositivos eletromecânicos, para que as pessoas incapazes de produzir uma resposta motora, possam comunicar-se com um computador e obter maior auto-suficiência, inclusão social e liberdade. Um dos principais desafios na pesquisa atual da ICM é a extração de características de sinais aleatórios de EEG com variação no tempo, e sua classificação com a maior precisão possível. Técnicas de extração de dados, como: densidade espectral de potência (PSD), centróides espectrais, desvio padrão e entropia são utilizadas na filtragem e investigação dos sinais a partir de dois exercícios mentais diferentes; Os sinais selecionados são classificados utilizando algoritmos de Análise Discriminante Linear (LDA), Common Spatial Patterns (CSP). A melhor precisão foi alcançada pela

[1] Formado em Ciência da Computação pela UFBA, especialização em engenharia e gestão pela USP – atua com Desenvolvimento, Arquitetura e Engenharia de Software há 16 anos. E-mail: iuri.sampaio@aln.senaicimatec.edu.br

[2] Breve Doutorado em Modelagem Computacional e Tecnologia Industrial, SENAI - Departamento Regional da Bahia, SENAI/DR/BA, Mestrado em Modelagem Computacional e Tecnologia Industrial, SENAI - Departamento Regional da Bahia, SENAI/DR/BA, Brasil. Especialização em Especialização Avançada em Sistemas Distribuídos, Universidade Federal da Bahia, UFBA, Brasil. Graduação em Processamento de Dados, Faculdade Ruy Barbosa, FRB, Brasil, E-mail: oberdan.pinheiro@fieb.com.br.

densidade espectral de potência. As precisões deste recurso vão desde 75% a 99%, dependendo da quantidade e qualidade das amostras de dados coletados. Por fim, o algoritmo de tradução será construído usando recursos de EEG selecionados e classificados para controlar os dispositivos ICM. Assim, este artigo apresenta os resultados da análise de sinais de EEG de intenção de movimentos específicos para extrair os recursos de sinais adequados, utilizando técnicas e algoritmos de classificação estatística que podem ser empregados para controlar dispositivos ICM que podem ser usados por pessoas com deficiência ou paralisadas.

Palavras-chave: Interface Cérebro Computador. Redes Neurais. Sinais Eletroencefalograma. Aprendizado de Máquina;

Sistema FIEB | SENAI CIMATEC
PELO FUTURO DA INOVAÇÃO

CENTRO UNIVERSITÁRIO CIMATEC

ESPECIALIZAÇÃO EM AUTOMAÇÃO, CONTROLE E ROBÓTICA

ABSTRACT

Brain Computer Interfaces (BCI) has become a vital field of biomedical engineering and computation, which uses electrical signals obtained from electroencephalogram exams (EEG) to provide assistive technologies (AT) for humans. The goal of BCI is to interpret brain activity in digital format and act with commands on electromechanical devices, so that people with limitations to produce a motor response can communicate with a computer device and achieve greater self-sufficiency, social inclusion and freedom. One of the main challenges in current BCI research is extracting features from time-varying random EEG signals and classifying them as accurately as possible. Feature extraction techniques are used to extract features that represent a unique property obtained from the brain signal pattern. Thus, this article presents the results of analyzing specific movement intent of EEG signals to extract the appropriate features using statistical classification techniques and algorithms, which can be employed to control BCI devices, so they can be used by people with temporary disabilities, disabled or paralyzed.

The EEG characteristics in terms of power spectral density (PSD), spectral centroids, standard deviation and entropy techniques were selected and investigated from two different mental exercises; The selected signals are classified using Linear Discriminant Analysis (LDA), Common Spatial Patterns (CSP). The best accuracy was achieved by the power spectral density. The accuracies of this feature are 75% to 99%, depending on the quantity and quality of the data samples collected. Finally, the translation algorithm will be built using selected and classified EEG resources to control the ICM devices.

Keywords: Brain Computer Interface. Neural networks. Electroencephalogram Signs. Machine Learning;

1 INTRODUÇÃO

Neste artigo, foi realizado um estudo de conjuntos de dados dos sinais EEG adquiridos de exames Eletroencefalogramas e aplicados à redes neurais para aprendizado de máquina e criação de um modelo computacional de decisão, para controle de dispositivos eletromecânicos, como por exemplo: uma cadeira de rodas.

Em particular, foram analisados conjuntos de dados respectivos a experimentos de imaginação, ou intenção, de movimento dos punhos esquerdo e direito. A atividade cerebral foi registrada no córtex pré-frontal, região do córtex cerebral responsável pela imaginação, ou planejamento, dos sinais motores do organismo humano, onde existem oscilações, crescente e decrescente, de atividade elétrica, entre 8 Hz e 12 Hz, que pode ser observada através dos sinais elétricos obtidos por exames eletroencefalograma.

> *"O córtex cerebral humano possui uma dinâmica espaço-temporal tremenda e próspera que é particularmente exclusiva do ser humano. Milhões de neurônios no cérebro comunicam-se através de sinais químicos e elétricos (potenciais de ação)". (SUBASI, 2005, p. 701-11)*

Especificamente quando um movimento é realizado, é possível identificar uma redução da atividade elétrica, ou diminuição de banda "mu", em regiões específicas do cérebro que lidam com a parte do corpo humano que está sendo movimentada naquele exato momento, como também, num intervalo de tempo anterior ao movimento.

A redução da banda "mu" do sinal EEG é chamada de Dessincronização Evento Relacionada, do inglês: event-related-dessincronization (ERD). Ao medir a atividade em diferentes locais do córtex motor cerebral, é possível determinar qual membro do corpo o indivíduo está se movimentando, exatamente pela diminuição da frequência da banda "mu" do sinal EEG, em determinada região do córtex.

Através dos conceitos de rede neural e neurônio espelho, este efeito também ocorre quando o indivíduo imagina o movimento, sem ainda o realizar. Isto é uma descoberta muito importante para a aplicação em dispositivos ICM.

Portanto, sempre que um movimento é imaginado, é possível observar uma diminuição da atividade de banda “mu” nos sinais EEG, por volta de 8 Hz a 12 Hz, incluindo frequências Alfa e Beta, de 8 Hz a 30 Hz, transformá-la em dados para identificar uma intenção de movimento e aplicá-la no controle de dispositivos ICM através das etapas de aquisição, pré-processamento, extração das características e classificação do sinal.

Os conjuntos de dados foram analisados utilizando algoritmos, técnicas e métodos que serão demonstrados neste artigo, aplicados na extração de características e classificação dos sinais EEG, como também, no treinamento de modelos matemáticos para alcançar resultados de alto desempenho. Também foram apresentados os resultados e os próximos passos na construção do modelo computacional para controle de uma Interface Cérebro Máquina.

Este artigo apresenta os resultados da análise de sinais de EEG de intenção de movimentos específicos para extrair os sinais EEG adequados, utilizando técnicas e algoritmos de classificação estatística que podem ser empregados para controlar dispositivos ICM que podem ser usados por pessoas com deficiência ou paralisadas.

2 METODOLOGIA

A interface Cérebro Máquina (ICM) sugere uma inovadora alternativa de comunicação entre humanos e máquinas. Ela oferece ao indivíduo uma maneira de controlar dispositivos eletrônicos com o pensamento, sem envolver qualquer movimento muscular.

Um sistema ICM pode ser dividido em cinco etapas: aquisição do sinal, pré-processamento, extração de características, classificação, e controle do dispositivo. A Figura 1 descreve a estrutura genérica de uma ICM com todas suas etapas.

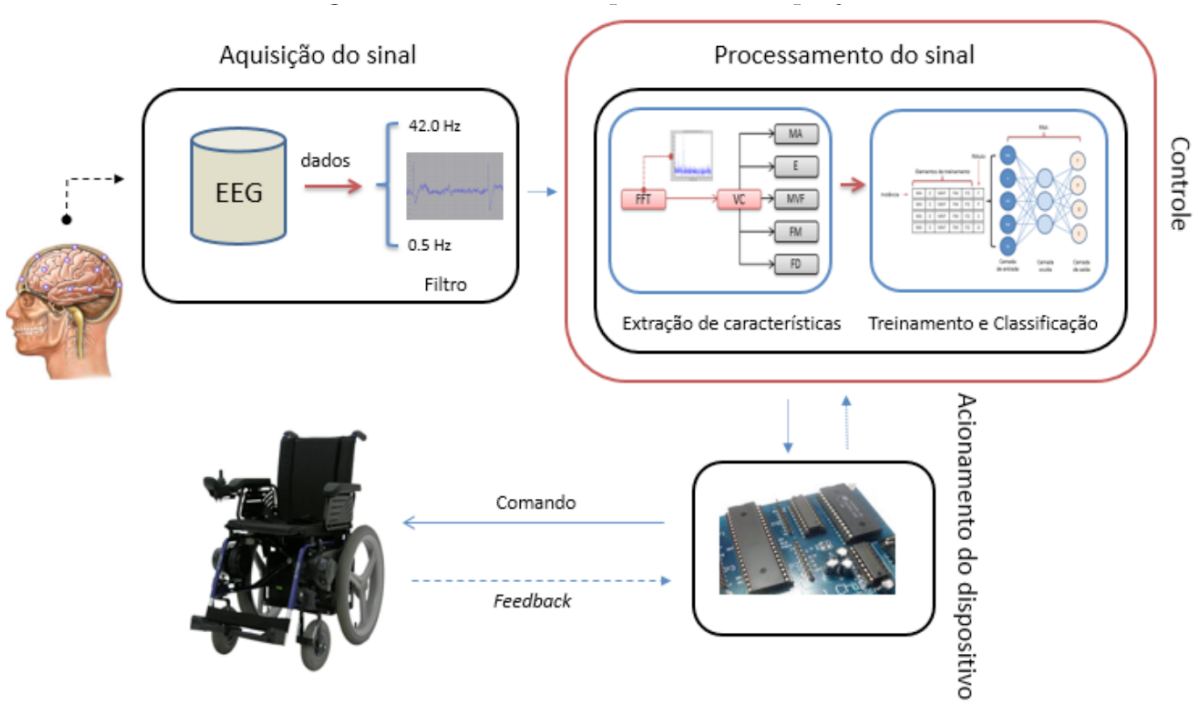

Figura 1. Descrição das etapas do sistema BCI. (Fonte: PINHEIRO, 2016)

2.1. Aquisição do sinal e protocolo de experimento

A etapa de aquisição do sinal consiste na obtenção dos dados através de exames Eletroencefalograma (EEG), onde a ocorre a leitura dos sinais elétricos cerebrais pelos sensores, localizados em regiões predeterminadas do cérebro, ou do escalpo, quando invasivos ou não invasivos respectivamente. Os sinais elétricos extraídos são digitalizados e gravados em arquivos de dados.

Os dados utilizados, neste artigo, foram adquiridos de pesquisas anteriores, publicadas pelos institutos: (i) PhysioNet, o Centro de Recursos de Pesquisa para Sinais Fisiológicos Complexos (PHYSIONET, 2022); e, (ii) Berlin BCI Group: Berlin Institute of Technology (BERLIN BCI, 2022).

Ambos já consagrados e certificados mundialmente na área de saúde, pela qualidade e segurança dos trabalhos de pesquisa e suas publicações, respectivamente os artigos: (i) BCI2000: a general-purpose brain-computer interface

(BCI) system, SCHALK, G., et al, 2004; e (ii) The non-invasive Berlin Brain–Computer Interface: Fast acquisition of effective performance in untrained subjects, BLANKERTZ, B., et al, 2007.

Os arquivos, contendo os dados, amostras de sinais elétricos EEG digitalizados, são de formato EDF (European Data Format), padrão internacional e universal criado para armazenar dados médicos de séries temporais (KEMP, B. et al, 1992).

A base de dados consiste em mais de mil e quinhentas (1500) gravações de um (1) e dois (2) minutos de sinais EEG, obtidas de cento e nove (109) voluntários, obedecendo um protocolo de experimento. Os indivíduos desempenharam atividades motoras e de imaginação, enquanto os sinais elétricos cerebrais foram coletados e gravados, em sessenta e quatro (64) canais EEG, de diferentes regiões do cérebro. (GOLDBERGER, A., 2000)

A figura 2. ilustra o equipamento utilizado para exames eletroencefalograma, bem como os sensores localizados em regiões pré-determinadas.

Figura 2. Eletroencefalograma não invasivo (Fonte: PINHEIRO apud WLADIMIR, 2013)

Os experimentos consistem nos respectivos estados (execução de tarefas): (i) olhos abertos e (ii) olhos fechados, que definem os sinais padrão para configuração base e definição de padrões de sinais cerebrais de cada indivíduo; (iii) a tarefa 1,

que define os movimentos abrir e fechar dos punhos esquerdo e direito; (iv) a tarefa 2, que define a imaginação dos movimentos abrir e fechar dos punhos esquerdo e direito; (v) a tarefa 3, que define os movimentos abrir e fechar dos punhos e pés, e; (vi) a tarefa 4, que define a imaginação dos movimentos abrir e fechar dos punhos e pés.

Neste trabalho, foi realizado o estudo das tarefas 1 e 2 apenas, e desta forma, possibilitou uma análise mais detalhada dos métodos matemáticos de extração de características e classificação de sinais EEG, que serão descritos ao longo do texto.

2.2. Pré-processamento e filtragem dos sinais

As etapas de pré-processamento e filtragem dos sinais consistem na seleção dos canais correspondentes às regiões do cérebro, áreas onde há maior incidência e intensidade dos sinais que interessam a esta análise. Particularmente, os sinais extraídos dos canais Fp1 e Fp2, respectivos à imaginação de movimentos do punho esquerdo e direito; e os canais C3 e C4, respectivos à execução motora do punho esquerdo e direito. Como ilustrado na Figura 3, onde os quatro canais foram destacados na cor rosa.

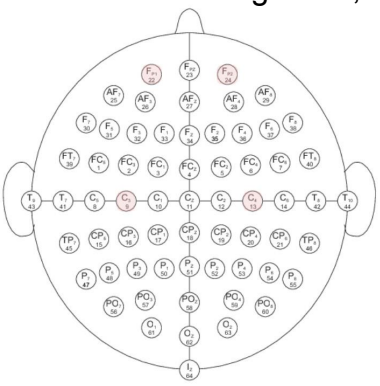

Figura 3. Distribuição dos eletrodos nas regiões do cérebro pré-determinadas.
(PINHEIRO apud GOLDBERGER et al., 2000)

Além da diferenciação de canais, entre áreas de captação de sinais EEG, também foram filtradas as frequências destes mesmos sinais.

Geralmente, existem cinco bandas de frequência para cada canal de EEG, que são: delta (0,5–4 Hz), teta (4–8 Hz), alfa (8–13 Hz), beta (14–30 Hz) e gama (30–30 Hz). 45Hz) [24]. Nesta pesquisa, os dados de EEG foram filtrados apenas em bandas alfa e beta. As faixas de frequência e as atividades de banda alfa e beta do EEG foram mostradas na Tabela 1.

Band	Frequency	Activity
Delta	0.5-4Hz	Deep sleep
Theta	4-8Hz	Drowsiness, light sleep
Alpha	8-13Hz	Relaxed
Beta	13-30Hz	Active thinking, alert
Gamma	More than 30Hz	Hyperactivity

Tabela 1. Bandas de frequências dos sinais EEG. (HASHID et al apud ATKINSON, 2016, pág 35-41)

2.3. Extração de Características

A extração de características é o processo de obtenção de valores numéricos ou nominais dos padrões cerebrais usados na comunicação com uma ICM.

As razões para realizá-la vão desde análises subsequentes mais fáceis à melhorias do desempenho das próximas etapas de classificação dos dados e treinamento dos modelos. Isto ocorre através de uma representação mais estável ou da remoção de informação redundante ou irrelevante (PFURTSCHELLER; GRAIMANN; NEUPER, 2006).

Algumas das técnicas utilizadas na extração de características aplicadas neste trabalho foram: a densidade espectral de potência média (PSD); padrão comum espacial (CSPs); análise linear de discriminante (LDA); centróide espectral; e, entropia de energia das bandas alfa e beta. Todas elas foram peças chave, muito importantes para alcançar uma melhor acuracidade e treinamento dos modelos matemáticos criados nesta pesquisa.

Além destas, os métodos de Transformação Rápida de Fourier (FFT), densidade espectral de potência, aplicados aos dados EEG sustentaram os resultados obtidos. A equação (1) da FFT (OTSUKA, T., 2009) e a equação (2) da PSD são mostrados a seguir.

$$X_{(k)} = \sum_{k=1}^{N-1} X_{(n)} W_N^{kn}; K = 0 \ldots N - 1 \quad (1)$$

$$W_N = e^{-j\frac{2\pi}{N}}$$

$$PSD = \left|X_{(k)}\right|^2 = \left|\sum_{n=0}^{N-1} x(nTs) e^{\frac{-j2\pi k}{N}}\right|^2 \quad (2)$$

Para o cálculo da PSD, de cada amostra, extraiu-se a frequência característica do sinal, nos canais Fp1 e Fp2, córtex pré-frontal, onde observou-se com sucesso a diminuição da atividade cerebral (μ), que significa a existência do movimento, e/ou imaginação do mesmo.

A figura 4, ilustra graficamente, o PSD das amostras calculados de maneira semelhante aos dados do SSVEP. Em neurociência, os potenciais evocados visualmente em estado estacionário (SSVEP) são sinais que são respostas naturais à estimulação visual em frequências específicas. Quando a retina é excitada por um estímulo visual variando de 3,5 Hz a 75 Hz (DING, J., 2006), o cérebro gera atividade elétrica na mesma frequência (ou múltiplos de) do estímulo visual.

A função [PSD] da biblioteca [MLAB] escrita em Python foi utilizada no cálculo do PSD, no intervalo de [0,5-2,5s] dos testes, em duas classes, esquerda e direita, feitos logo após o início da sugestão de imaginação do movimento dos punhos esquerdo e direito.

Python: $mlab.psd(trials[ch,:,trial],\ NFFT = int(nsamples),\ Fs = sample\_rate)$

Todos os algoritmos utilizados neste trabalho estão disponíveis no item Anexo de artigo.

```
plot_psd(
    trials_PSD,
    freqs,
    [channel_names.index(ch) for ch in ['Fp1', 'Fpz', 'Fp2']],
    chan_lab=['left', 'center', 'right'],
)
```

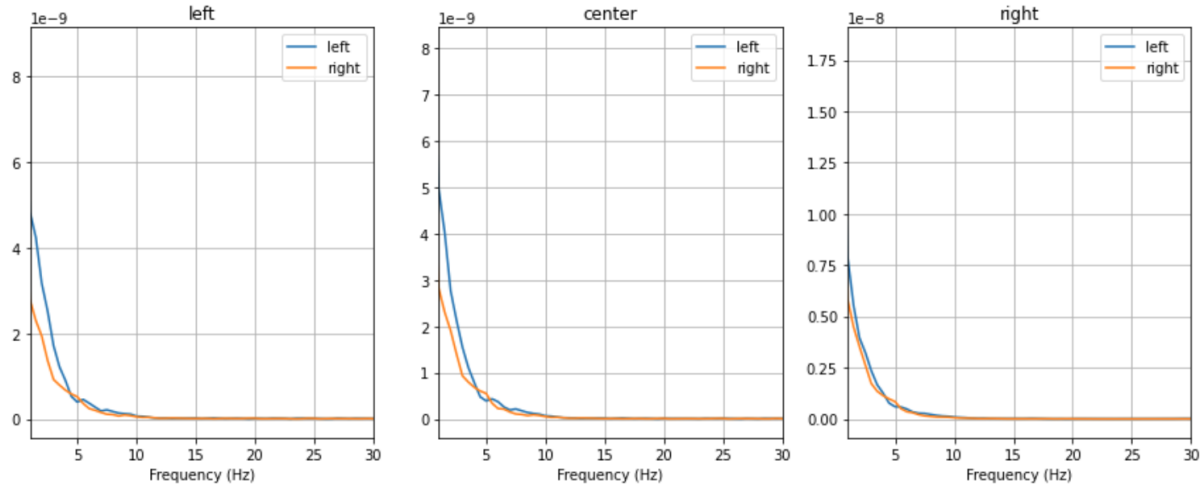

Figura 4. Representação gráfica dos dados calculados PSD; Fonte: autor

Este artigo consiste na análise de movimentos controlados no córtex pré-frontal, onde há um aumento da atividade “mu” (8-12 Hz) no instante em que os movimentos foram imaginados. Acompanhado por uma redução dessa atividade “mu” em regiões específicas que lidam com o membro que está se movendo no momento.

Essa diminuição é chamada de Dessincronização Relacionada a Eventos (ERD). Ao medir a quantidade de atividade “mu” em diferentes locais do córtex pré-frontal, canais Fp1 e Fp2, podemos determinar qual membro o indivíduo imaginou. Através dos neurônios-espelho, esse efeito também ocorre quando o sujeito está realmente movendo seus membros.

Um pico de atividade "mu" pode ser observado em cada canal para ambas as classes. No hemisfério direito, a atividade "mu" para o movimento do punho esquerdo é menor do que o direito devido à ERD.

No eletrodo esquerdo, a "mu" para o movimento da mão direita é reduzida e no eletrodo central a atividade “mu” é aproximadamente igual para ambas as classes.

Isso está de acordo com a teoria de que o punho esquerdo é controlado pelo hemisfério direito e o punho direito é controlado pelo hemisfério esquerdo.

2.4. Classificação de dados e Treinamento

Além da etapa de extração de características, a classificação de sinais e treinamento de modelos matemáticos, desempenham um papel fundamental para projetar qualquer sistema ICM porque a seleção adequada de técnicas e representações matemáticas aumenta a precisão do classificador e o desempenho do dispositivo ICM.

O algoritmo de aprendizado de máquina, descrito a seguir, foi criado para construir um modelo que classifica/distingue os movimentos do punho esquerdo e direito em duas classes respectivas.

1. Encontrar uma maneira de quantificar a atividade "mu" presente numa amostra;
2. Criar um modelo que descreva os valores esperados da atividade "mu" para cada classe;
3. Testar o modelo com novos dados para confirmar a capacidade preditiva dos algoritmos em determinar as classes corretamente.

Seguindo o desenho clássico de uma ICM, descrito por Blankertz em BLANKERTZ, B. et al., 2007, utilizou-se, em todas as amostras, o logaritmo da variância do sinal numa determinada banda de frequência, como característica para o classificador, para projetar um filtro “passa-banda” que utiliza a biblioteca

[scipy.signal.irrfilter], escrita em Python, para remover as frequências fora da janela de [8 Hz, 15 Hz].

A função [iirfilter()] utiliza a ordem do filtro: números mais altos significam um corte de frequência mais nítido, e o sinal resultante pode ser deslocado no tempo. Números mais baixos significam um corte de frequência suave, e o sinal resultante menos distorcido no tempo.

Ela também trata os limites de frequência inferior e superior, divididos pela frequência NYQUIST, que é a metade da taxa de amostragem (i.e. “sample_rate” dividida por 2).

$$a,\ b\ =\ scipy.signal.iirfilter(6,\ [lo/(sample\_rate/2.0),\ hi/(sample\_rate/2.0)])$$

Todos os algoritmos utilizados neste trabalho estão disponíveis no item Anexo deste artigo.

A Figura 5, a seguir, mostra os resultados dos cálculos após a classificação e a supressão de frequências do filtro de banda passante.

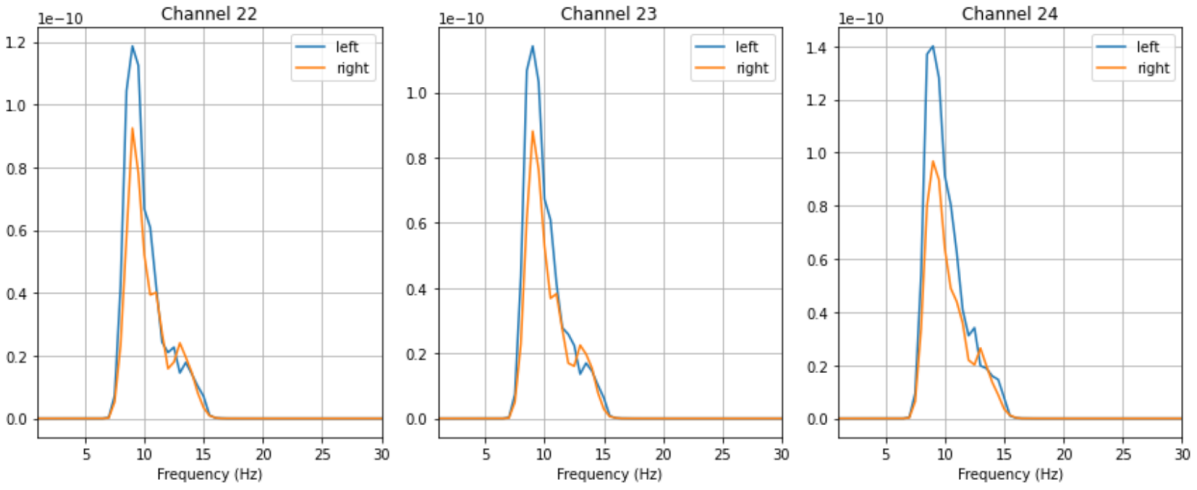

Figura 5. PSD resultante com a supressão de frequências do filtro de banda passante. Fonte: próprio autor;

A maioria dos canais mostra uma pequena diferença no log-var do sinal entre as duas classes: esquerda e direita. O próximo passo foi reduzir de 118 canais para apenas algumas combinações de canais.

O algoritmo CSP calcula as combinações de canais (W), que são projetados para maximizar a diferença de variação entre duas classes. Essas combinações são chamadas de filtros espaciais, que também calcula a covariância para cada amostra e retorna as médias respectivas, ao aplicar uma matriz de combinação que basicamente multiplica o W (de cada amostra) com a matriz de sinais EEG. Como mostrado na Figura 6 a seguir.

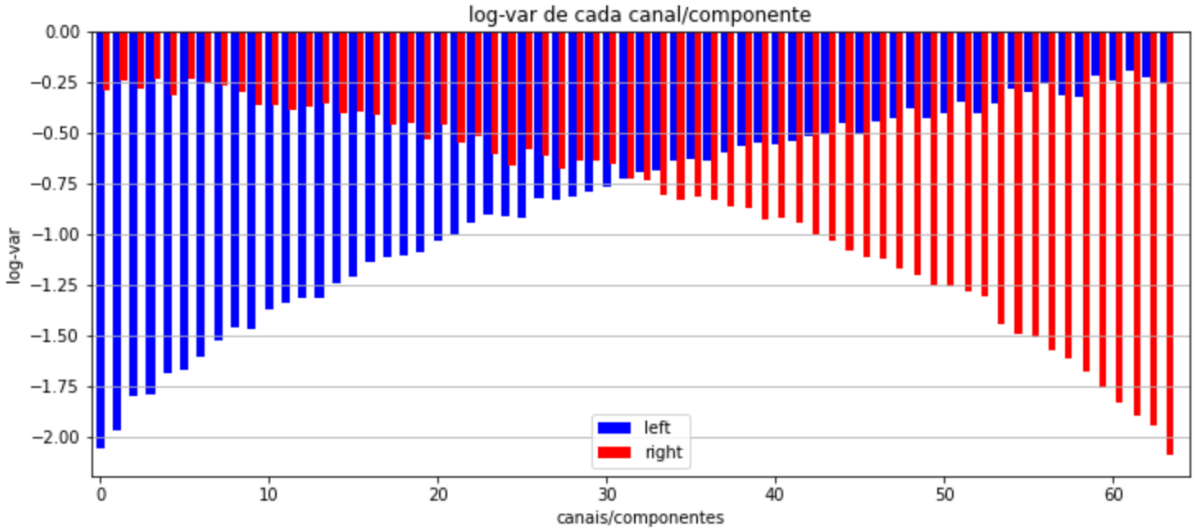

Figura 6. Resultados do algoritmo CSP, representados no gráfico log-var. Fonte: próprio autor

Ao invés de 118 canais, agora temos 118 combinações de canais, chamados de componentes, que são o resultado de 118 filtros espaciais aplicados aos dados.

Os primeiros filtros maximizam a variação da primeira classe, enquanto minimizam a variação da segunda. Os últimos filtros maximizam a variação da segunda classe, enquanto minimizam a variação da primeira.

O gráfico seguinte, da Figura 7, mostra o resultado do PSD após o filtro, aplicado aos três componentes.

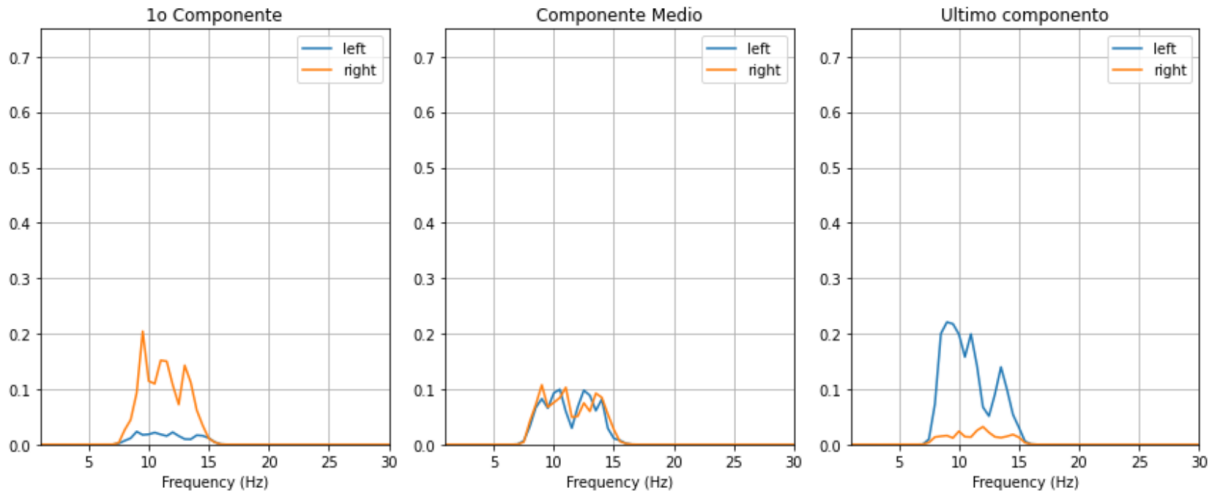

Figura 7. Combinações de canais, chamados de componentes; Fonte: proprio autor.

É possível diferenciar bem as duas classes: esquerda e direita, no gráfico de dispersão. Ele é uma ferramenta útil para visualizar essa diferença. Ambas as classes são mostradas num plano bidimensional: o eixo x é o primeiro componente CSP, o eixo y é o último.

Em seguida, o algoritmo de classificação linear foi aplicado para desenhar uma linha no gráfico acima para separar as duas classes. Para determinar a classe da nova amostra, verifica-se em qual lado da linha ela está apresentada.

Como informativo, foi recriado o gráfico de dispersão e sobreposto o limite de decisão conforme determinado pelo classificador LDA. O limite de decisão é a linha para a qual a saída do classificador é exatamente zero (0).

Primeiramente, traça-se o limite de decisão com os dados de treinamento usados para calculá-lo. O gráfico a seguir mostra o limite dos dados da amostra

teste, nos quais aplica-se o classificador. Nele, evidencia-se que o classificador comete alguns erros.

Conforme mostrado na Figura 8 a seguir, o limite dos dados de teste ao aplicar o classificador, contendo dados sobrepostos, muito próximos ao limite (linha tracejada)

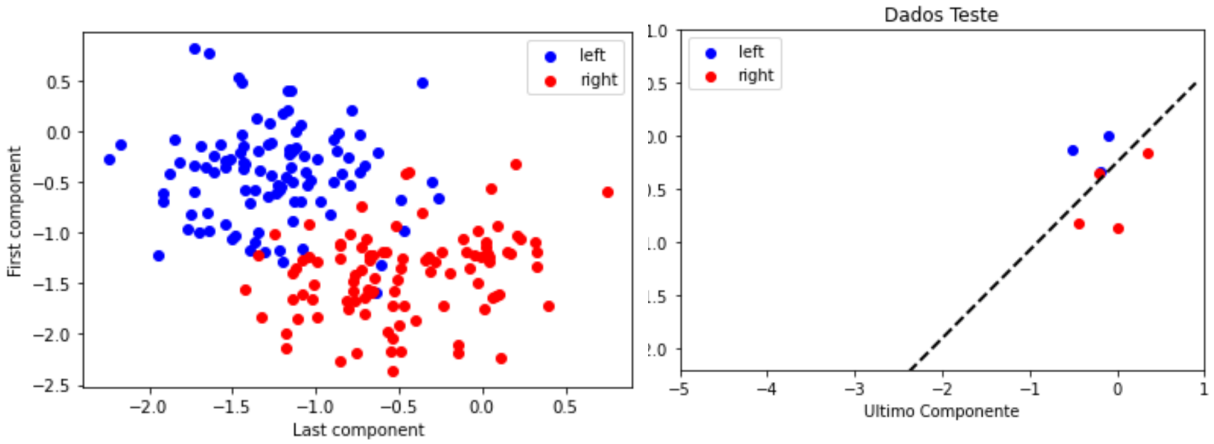

Figura 8. Gráfico de dispersão das amostras nas classes esquerda e direita. Fonte: próprio autor.

Além da divisão de classes, conforme mencionado anteriormente, os dados são divididos em amostras de treinamento e um conjunto de teste, cuja porcentagem de amostras para utilizar no treinamento segue a divisão 50-50.

O classificador ajusta um modelo (neste caso, uma linha reta) no conjunto de treinamento e utiliza este modelo para fazer previsões sobre o conjunto de teste, ou seja, define em qual lado da linha cada amostra do conjunto de teste se apresenta.

O algoritmo CSP faz parte do modelo, portanto, deve ser calculado utilizando apenas os dados de treinamento.

Para o classificador, também foi utilizado o algoritmo de Análise Discriminante Linear (LDA), que ajusta uma distribuição Gaussiana para cada classe, caracterizada pela média e covariância. Ele determina um plano de separação ótimo

para dividir as duas classes. Com isso, é possível calcular o resultado, positivo ou negativo, para classificar corretamente os dados, ou seja, determinar a classe de uma nova amostra. Este plano é definido como:

$r = W_0 * X_0 + W_1 * X_1 + \ldots + W_n * X_n - b$, onde: r é a saída do classificador; W_n são os pesos das características; X_n são as características das amostras; n é a dimensão dos dados, e; b é o deslocamento, ou desvio padrão.

Neste caso, já com dados bidimensionais, o plano de separação foi então definido como a linha reta: $r = W_0 * X_0 + W_1 * X_1 - b$. O gráfico de dispersão visto na Figura 8. usou X_0 como eixo x e X_1 como eixo y.

Para encontrar a função $y = f(x)$ que descreve o limite de decisão, defini-se r para 0 e resolve-se para y na equação do plano de separação.

Ao treinar o LDA utilizando os dados de treinamento, o algoritmo retorna W: [24.0296518, -28.98866899] e b: 7.121690208432298. Conforme o calculo mostrado na equação abaixo:

$W_0 * X_0 + W_1 * X_1 - b = r$ (1) equação original

$W_0 * x + W_1 * y - b = 0$ (2) substituindo X_0 por x, X_1 por y e $r = 0$

$W_1 * y = b - W_0 * x$ (3) isolando y

$y = \frac{b - W_0 * x}{W_1}$ (4) resolvendo y

O algoritmo LDA foi construído e ajustado aos dados de treinamento para ser aplicado aos dados de teste, cujos resultados foram representados numa matriz de confusão, onde o número na diagonal representam as amostras que foram classificadas corretamente, e quaisquer amostras classificadas incorretamente,

como por exemplo: um falso positivo ou falso negativo; foram representados nos cantos. A precisão foi de 0,857.

Matriz de Confusão: `[[3 1]`

`[0 3]]`

A matriz de confusão mostra que uma (1) das três (3) amostras resultantes de imaginação do movimento do punho esquerdo, foram classificadas incorretamente como imaginação do movimento do punho direito, e não houve erros na classificação das amostras de imaginação do movimento do punho direito, zero (0) das três (3) amostras resultantes. No total, 85,7% das amostras foram classificadas corretamente.

3 CONSIDERAÇÕES FINAIS

Nesta pesquisa, duas classes de exercício mental foram classificadas com os quatro métodos de classificação e treinamento de RNAs: Densidade Espectral de Potência, Centróide Espectral, Desvio Padrão e Entropia Energética.

Os resultados obtidos, a partir das bases de dados Physionet e BCI V, alcançaram as taxas de sucesso esperadas chegando até 91% de acuracidade, dependendo da quantidade e qualidade das amostras testadas.

Para classificar os sinais foram usadas as plataformas nas plataformas de software: Google Colab e Weka; e os algoritmos: PSD, LDA, Random Forest, Dl4jMlpClassifier, SVM e K-NN, validação cruzada, perceptron multicamadas, entre outros. Todos com o objetivo de alcançar melhor desempenho e acuracidade nos resultados.

A melhor precisão de classificação foi obtida com a combinação dos algoritmos PSD, CSP e LDA, com 91%, e, portanto, aplicar o modelo e seus métodos ao algoritmo de tradução para executar a ICM para controlar o dispositivo eletromecânico, cadeira de rodas.

Os próximos passos consistem no aprofundamento desta pesquisa, e na exploração de novos métodos. Não somente na aplicação de ICM em dispositivos eletromecânicos, como também, no auxílio à medicina e tratamento diagnóstico, nas diversas variações de tecnologias assistivas, e dispositivos de controle e monitoramento, aplicados na área da saúde.

Sempre com o intuito de melhorar a qualidade de vida e inclusão dos indivíduos.

O autor agradece o grande apoio de sua equipe de pós-graduação: o supervisor mestre Dr. Oberdan Pinheiro, autor do artigo que fundamenta e motiva esta pesquisa - “SmartChair : Cadeira de Rodas Inteligente com Interface Flexíıvel; aos colegas das disciplinas que contribuíram de alguma forma para construção deste estudo, no curso de especialização em Automação e Robótica Industrial, SENAI CIMATEC BA.

REFERÊNCIAS

PINHEIRO, O., SmartChair: Cadeira de Rodas Inteligente com Interface Flexível, 2016;

Artigo digital Electroencephalogram (EEG), website Jhon Hpkins Medicine, acessado em 07 de abril de 2022, https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/electroencephalogram-eeg#:~:text=An%20EEG%20is%20a%20test,activity%20of%20your%20brain%20cells.

Artigo Digital Brain-computer interface (BCI), website TechTarget, acessado em 07 de abril de 2022, https://www.techtarget.com/whatis/definition/brain-computer-interface-BCI
CHEN, J. Artigo digital Neural Network, publicado em Dezembro de 2021, acessado em 07 de abril de 2022

Website Investopedia, acessado em 07 de abril de 2022, https://www.investopedia.com/terms/n/neuralnetwork.asp#:~:text=A%20neural%20network%20is%20a,organic%20or%20artificial%20in%20nature.

Website Sales Brain, acesso em 07 de abril de 2022, https://www.salesbrain.com/neuroresearch-portuguese/eeg-and-eye-tracking-portuguese/?lang=pt-br

SUBASI A., Automatic recognition of alertness level from EEG by using neural network and wavelet coefficients. Expert Syst Appl 2005;28:701-11

CHO, H. et al., A Step-by-Step Tutorial for a Motor Imagery–Based BCI, Jan. 2018

RASHID M.,, The Classification of EEG Signal Using Different Machine Learning Techniques for BCI Application, 2019, acessado em Maio 2022, https://link.springer.com/chapter/10.1007/978-981-13-7780-8_17

SCHALCK, G., McFarland, D.J., Hinterberger, T., Birbaumer, N., Wolpaw, J.R. BCI2000: A General-Purpose Brain-Computer Interface (BCI) System. IEEE Transactions on Biomedical Engineering 51(6):1034-1043, 2004.

GOLDBERGER, A., Amaral, L., Glass, L., Hausdorff, J., Ivanov, P. C., Mark, R., ... & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation [Online]. 101 (23), pp. e215–e220.

KEMP, B et al. A simple format for exchange of digitized polygraphic recordings, 1992

PHYSIONNET, o Centro de Recursos de Pesquisa para Sinais Fisiológicos Complexos, acessado em 13 de junho de 2022, disponível em: <https://physionet.org/content/eegmmidb/1.0.0/>

BERLIN BCI GROUP, Berlin Institute of Technology, acessado em 15 de junho de 2022, disponvel em: <https://www.bbci.de/competition/iv/desc_1.html>

BLANKERTZ, B., Dornhege, G., Krauledat, M., Müller, K.-R., & Curio, G. (2007). The non-invasive Berlin Brain-Computer Interface: fast acquisition of effective performance in untrained subjects. *NeuroImage*, 37(2), 539–550. doi:10.1016/j.neuroimage.2007.01.051

PFURTSCHELLER, G.; GRAIMANN, B.; NEUPER, C., EEGBased BrainComputer Interface System, University of Technology Graz, BCILab , Graz, Austria, 2006.

OTSUKA, T., et al.: Effects of mandibular deviation on brain activation during clenching: an fMRI preliminary study. Cranio 27, 88–93 (2009)

DING J, Sperling G, Srinivasan R (2006). "Attentional modulation of SSVEP power depends on the network tagged by the flicker frequency". *Cereb. Cortex*. **16** (7): 1016–29, 2006;

CENTRO UNIVERSITÁRIO CIMATEC

ESPECIALIZAÇÃO EM AUTOMAÇÃO, CONTROLE E ROBÓTICA

ANEXO

```
"""Copy of EEG_BCI200_Motor_Imagery_Analysis_Classification_Dataset_1_EDF.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1s3gEViaS5tWi22n09eB81VA0ZfClzmAk
"""

# To install mne library
!pip install mne

# %%
import numpy as np
import scipy.io
import mne

"""Descricao do Dataset https://physionet.org/content/eegmmidb/1.0.0/

"""

# Connect to GDrive
from google.colab import drive
drive.mount('gdrive')

# To read EDF
raw = mne.io.read_raw_edf('gdrive/MyDrive/TCC/Datasets/eegmmidb/S002/S002R03.edf',
preload=True)

"""Adicionar este trecho de codigo para isolar as variaveis e dados de analise"""

raw.info

"""Each annotation includes one of three codes (T0, T1, or T2):

T0 corresponds to rest

T1 corresponds to onset of motion (real or imagined) of:
  the left fist (in runs 3, 4, 7, 8, 11, and 12)
  both fists (in runs 5, 6, 9, 10, 13, and 14)

T2 corresponds to onset of motion (real or imagined) of
the right fist (in runs 3, 4, 7, 8, 11, and 12)
both feet (in runs 5, 6, 9, 10, 13, and 14)
"""

events=mne.events_from_annotations(raw)

event_dict={
    'left':2,
    'right':3
```

```
}

events

events[0][0:20]

events[0][1]

fig=mne.viz.plot_events(events[0],event_id=event_dict,sfreq=raw.info['sfreq'],first_samp=raw.first_s
amp)

epoch=mne.Epochs(raw,events[0],event_id=[2,3],tmin=-0.2,tmax=1.9)

epoch.get_data().shape

labels=epoch.events[:-1]

labels

labels.shape

# evoked_1=epoch['1'].average()
evoked_2=epoch['2'].average()
evoked_3=epoch['3'].average()

dicts={
    'left':evoked_2,
    'right':evoked_3
}

mne.viz.plot_compare_evokeds(dicts)

def read_data(path):
  raw = mne.io.read_raw_edf(path, preload=True)
  raw.set_eeg_reference()
  events=mne.events_from_annotations(raw)
  epoch=mne.Epochs(raw,events[0],event_id=[2,3], tmin=-0.2,tmax=1.9)
  labels=epoch.events[:,-1]
  features=epoch.get_data()
  return labels,features,epoch

path='gdrive/MyDrive/TCC/Datasets/eegmmidb/S015/S015R03.edf'
label,features,epoch=read_data(path)

features.shape, labels.shape

# evoked_1=epoch['1'].average()
evoked_2=epoch['2'].average()
evoked_3=epoch['3'].average()

mne.viz.plot_compare_evokeds(dicts)

# To read EDF
```

```
raw = mne.io.read_raw_edf('gdrive/MyDrive/TCC/Datasets/eegmmidb/S003/S003R03.edf',
preload=True)
raw.info

events=mne.events_from_annotations(raw)

sample_rate = raw.info["sfreq"]
EEG = raw.get_data() # EEG = m['cnt'].T
# nchannels=len(raw.ch_names)
# nsamples=raw.n_times
nchannels, nsamples = EEG.shape

raw.rename_channels({ch: ch.replace('.', '') for ch in raw.ch_names})
channel_names = raw.ch_names

# event_onsets = m['mrk'][0][0][0]
# event_codes = m['mrk'][0][0][1]
event_onsets=[]
event_codes=[]
for elem in events[0]:
  # print(elem[2])
  if elem[2]!=1:
    event_onsets.append(elem[0])
    event_codes.append(elem[2])
event_onsets = np.array([event_onsets], dtype=np.int16)
event_codes = np.array([event_codes], dtype=np.int16)

labels = np.zeros((1, nsamples), int)
labels[0, event_onsets] = event_codes

cl_lab = ['left', 'right']
cl1 = cl_lab[0] # esquerda
cl2 = cl_lab[1] # direita

nclasses = len(cl_lab)
nevents = len(event_onsets)

print('Forma do EEG', EEG.shape)
print('Taxa de amostragem', sample_rate)
print('Quantidade de canais', nchannels)
print('Nomes dos Canais', channel_names)
print('Quantidade de eventos', len(event_onsets))
print('Codigos de Eventos', np.unique(event_codes)) # esquerda -1 e direita 1
print('Classe de rotulos', cl_lab)
print('Quantidade de classes', nclasses)

"""Os dados sao extraidos a partir de uma grande amostra: 64 eletrodos foram usados, espalhados
por todo o couro cabeludo.

O sujeito recebeu uma sugestão/estimulo externo e, em seguida, imaginou o movimento da mão
direita ou o movimento dos pés.

Como pode ser visto no Homunculus, o movimento do pé é controlado no centro do córtex motor (o
```

```
que torna difícil distinguir o pé esquerdo do direito), enquanto o movimento da mão é controlado
mais lateralmente.

# Plotando os dados

O código abaixo corta os testes em duas classes. # Os cortes sao feitos no intervalo [0,5-2,5 s]
após o início da sugestão.
"""

# Dicionário para armazenar as tentativas, cada classe recebe uma entrada
trials = {}

# A janela de tempo (de amostras) para extrair cada tentativa (trial), 0.5 -- 2.5 segundos
win = np.arange(int(0.5*sample_rate), int(2.5*sample_rate))

# Tamanho da janela de tempo
nsamples = len(win)

# Iteracao  nas classes (direita, esquerda)
for cl, code in zip(cl_lab,np.unique(event_codes)):
  # Extrair eventos onsets das classes
  cl_onsets = event_onsets[event_codes == code]

  # alocar memoria para as amostras
  trials[cl] = np.zeros((nchannels, nsamples, len(cl_onsets)))

  # Extrair cada amostra
  for i, onset in enumerate(cl_onsets):
    trials[cl][:,:,i] = EEG[:, win+onset]

# Info aobre as dimensoes dos dados (canais X tempo X amostras - channels X time X trials )
print('Forma das amostras triasl[cl1]', trials[cl1].shape)  # esquerda
print('Forma das amostras triasl[cl2]', trials[cl2].shape)  # direita

"""A caracteristica a ser buscada é frequência (uma diminuição na atividade), vamos plotar o PSD
das tentativas de maneira semelhante aos dados do SSVEP.

O código abaixo define uma função que calcula o PSD (Power Spectral Density) para cada amostra
teste (que sera utilizada posteriormente no codigo):
"""

from matplotlib import mlab

def psd(trials):
  '''
  Calcula o PSD de cada amostra
  Parametros:
  ----------
  amostras: 3d-array ( canais X PSD X amostras )
  Sinal EEG

  Retorna:
  ----------
```

```
  amostras_PSD: 3d-array ( canais X PSD X trials)
    O PSD de cada amostra
  freqs: lista de floats
    Frequencias correspondentes aos PSD que foram processados (necessario para plotar o grafico)
  '''

  ntrials = trials.shape[2]
  trials_PSD = np.zeros((nchannels, 161, ntrials))

  # iteracao sobre amostras e canais
  for trial in range(ntrials):
   for ch in range(nchannels):
      # Calcular PSD
      (PSD, freqs) = mlab.psd(trials[ch,:,trial], NFFT=int(nsamples), Fs=sample_rate)
      trials_PSD[ch, :, trial] = PSD.ravel()
  return trials_PSD, freqs

psd_l, freqs = psd(trials[cl1])
psd_r, freqs = psd(trials[cl2])
trials_PSD = {cl1: psd_l, cl2: psd_r}

import matplotlib.pyplot as plt

def plot_psd(trials_PSD, freqs, chan_ind, chan_lab=None, maxy=None):
  '''
  Plota dados PSD calculados com a funcao psd()

  Parametros:
  ----------------
  trials: 3d-array
    Dados PSD, retornados pela funcao psd()
  freqs: lista de floats
    Lista de frequencias correspondentes aos PSD que foram definidos
  chan_ind: list de inteiros
    Indices de canais para plotar o grafico
  chan_lab: lista de strings
    (opcional) lista de nomes de cada canal
  maxy: float
    (opcional) limite no eixo y
  '''
  plt.figure(figsize=(12,5))
  nchans = len(chan_ind)

  # Maximo de 3 plotagens por linha
  nrows = int(np.ceil(nchans / 3))
  ncols = min(3, nchans)

  # Iteracao sobre canais
  for i,ch in enumerate(chan_ind):
    # Definir qual suplot para desenhar
    plt.subplot(nrows,ncols,i+1)

    # Plotar PSD para cada classe
```

```
    for c1 in trials.keys():
      plt.plot(freqs, np.mean(trials_PSD[c1][ch,:,:], axis=1), label=c1)

    # Ajutar decoracao do grafico
    plt.xlim(1,30)
    if maxy != None:
      plt.ylim(0,maxy)

    plt.grid()
    plt.xlabel('Frequency (Hz)')
    if chan_lab == None:
      plt.title('Channel %d' % (ch+1))
    else:
      plt.title(chan_lab[i])
    plt.legend()
  plt.tight_layout()

"""Um pico de atividade "mu" pode ser visto em cada canal para ambas as classes.

No hemisfério direito, o "mu" para o movimento da mão esquerda é menor do que para o
movimento da mão direita devido ao ERD.

No eletrodo esquerdo, o "mu" para o movimento da mão direita é reduzido e no eletrodo central a
atividade mu é aproximadamente igual para ambas as classes.

Isso está de acordo com a teoria de que a mão esquerda é controlada pelo hemisfério direito e os
pés são controlados centralmente.
"""

plot_psd(
    trials_PSD,
    freqs,
    [channel_names.index(ch) for ch in ['Fp1', 'Fpz', 'Fp2']],
    chan_lab=['left', 'center', 'right'],
)

"""# Classificar os dados
Foi utilizado um algoritmo de aprendizado de máquina para construir um modelo distingue entre o
movimento da mão direita e do pé desse sujeito. Para fazer isso precisamos:

Encontre uma maneira de quantificar a quantidade de atividade "mu" presente numa amostra
Criar um modelo que descreva os valores esperados da atividade "mu" para cada classe
Finalmente teste este modelo em alguns dados não vistos para ver se ele pode prever o rótulo de
classe correto

Seguindo um desenho clássico do BCI de Blankertz et al. [1] onde utilizam o logaritmo da variância
do sinal numa determinada banda de frequência como característica para o classificador.

[1] Blankertz, B., Dornhege, G., Krauledat, M., Müller, K.-R., & Curio, G. (2007). A Interface
Cérebro-Computador de Berlim não invasiva: aquisição rápida de desempenho efetivo em
indivíduos não treinados. NeuroImage, 37(2), 539-550. doi:10.1016/j.neuroimage.2007.01.051

O script abaixo projeta um filtro passa-banda usando scipy.signal.irrfilter que removerá as
```

frequências fora da janela de 8-15Hz. O filtro é aplicado a todas as amostras:

O objetivo e construir um modulo para classificar/distinguir movimentos da mao esquerda e direita
1. aplicar etapas para quantificar a quantidade de atividades que estao presentes em cada amostra.
2. criar um modulo de descreve os valores respetivos de atividade de cada classe
3. testar o modelo com outros/novos sinais/dados e confirmar se o modelo consegue prever as classe corretamente

A função iirfilter() usa a ordem do filtro: números mais altos significam um corte de frequência mais nítido, mas o sinal resultante pode ser deslocado no tempo, números mais baixos significam um corte de frequência suave, mas o sinal resultante menos distorcido no tempo.

**Também* leva os limites de frequência inferior e superior para passar, dividido pela frequência niquist, que é a taxa de amostragem dividida por 2:*

"""

```
import scipy.signal
def bandpass(trials, lo, hi, sample_rate):
  '''
    Estrutura e aplica um filter de banda de passada ao signal.
    Parameters
    ----------
    trials: 3d-array ( canais X amostras X testes)
      Sinal EEG
    lo: float
      Limite de frequência inferior (Hz)
    hi: float
      Limite de frequência superior (Hz)
    sample_rate: float\
      Taxa de amostragem do sinal EEG

    Retorno:
    ----------
    trials_filt: 3d-array (canais X amostras X testes)
      Sinal EEG de banda de passagem
  '''
  a, b = scipy.signal.iirfilter(6, [lo/(sample_rate/2.0), hi/(sample_rate/2.0)])

  # Aplicando o filtro a cada tentativa
  ntrials = trials.shape[2]
  trials_filt = np.zeros((nchannels, nsamples, ntrials))
  for i in range(ntrials):
    trials_filt[:, :, i] = scipy.signal.filtfilt(a, b, trials[:, :, i], axis=1)
  return trials_filt

# Aplicar a funcao
trials_filt = {cl1: bandpass(trials[cl1], 8, 15, sample_rate),
               cl2: bandpass(trials[cl2], 8, 15, sample_rate)}
```

```
psd_l, freqs = psd(trials_filt[cl1])
psd_r, freqs = psd(trials_filt[cl2])
trials_PSD = {cl1: psd_l, cl2: psd_r}

plot_psd(
    trials_PSD,
    freqs,
    [channel_names.index(ch) for ch in ['Fp1', 'Fpz', 'Fp2']],
)

#  Como recurso para o classificador, usaremos o logaritmo da variância de cada canal. A função abaixo calcula isso:
# Calculate the log(var) of the trials
def logvar(trials):
    '''
    Calcular log-var de cada canal.

    Parametros
    ----------
    trials : 3d-array (channels x samples x trials)
        Sinal EEG .

    Retorno
    -------
    logvar - 2d-array (canais x amostras)
        Para cada canal o logvar do sinal
    '''
    return np.log(np.var(trials, axis=1))


# Aplicar a funcao
trials_logvar = {cl1: logvar(trials_filt[cl1]),
                 cl2: logvar(trials_filt[cl2])}

# Abaixo está uma função para visualizar o logvar de cada canal como um gráfico de barras:
def plot_logvar(trials):
    '''
    Plota a log-var de cada canal/componente.
    arguments:
        trials - Dicionario contendo as amostras (log-vars x trials) para 2 classes.
    '''
    plt.figure(figsize=(12,5))

    x0 = np.arange(nchannels)
    x1 = np.arange(nchannels) + 0.4

    y0 = np.mean(trials[cl1], axis=1)
    y1 = np.mean(trials[cl2], axis=1)

    plt.bar(x0, y0, width=0.5, color='b')
    plt.bar(x1, y1, width=0.4, color='r')
```

```
    plt.xlim(-0.5, nchannels+0.5)

    plt.gca().yaxis.grid(True)
    plt.title('log-var de cada canal/componente')
    plt.xlabel('canais/componentes')
    plt.ylabel('log-var')
    plt.legend(cl_lab)

"""A maioria dos canais mostra uma pequena diferença no log-var do sinal entre as duas classes.

O próximo passo é passar de 118 canais para apenas algumas combinacoes de canais.

O algoritmo CSP calcula combinacoes de canais que são projetados para maximizar a diferença de variação entre duas classes.

Essas combinacoes são chamadas de filtros espaciais.
"""

# Plotar log-vars
plot_logvar(trials_logvar)

from numpy import linalg
def cov(trials):
  ''' Calcula a covariancia para cada amostra e retorna as medias respoctivas '''
  ntrials = trials.shape[2]
  covs = [ trials[:,:,i].dot(trials[:,:,i].T) / nsamples for i in range(ntrials) ]
  return np.mean(covs, axis=0)

def whitening(sigma):
  ''' Calcula a matriz "whitening" para a matriz sigma de covariancia. '''
  U, l, _ = linalg.svd(sigma)
  return U.dot( np.diag(l ** -0.5) )

def csp(trials_l, trials_r):
  ''' Calcula a matriz W de transformacao CSP
    argumentos:
    ----------
      trials_l: array ( canais X testes X amostras ) contendo as amostras de movimento do punho esquerdo
      trials_r: ( canais X testes X amostras ) contendo as amostras de movimento do punho direito
    retorna:
      Matriz W de combinacao
  '''
  cov_l = cov(trials_l)
  cov_r = cov(trials_r)
  P = whitening(cov_l + cov_r)
  B, _, _ = linalg.svd( P.T.dot(cov_r).dot(P) )
  W = P.dot(B)
  return W

def apply_mix(W, trials):
  ''' Aplica uma matriz de combinacao a cada amostra ( basicamente multiplica W com a matriz de sinais EEG ) '''
```

```
  ntrials = trials.shape[2]
  trials_csp = np.zeros((nchannels, nsamples, ntrials))
  for i in range(ntrials):
    trials_csp[:,:,i] = W.T.dot(trials[:,:,i])
  return trials_csp

# Aplicar as funcoes
W = csp(trials_filt[cl1], trials_filt[cl2])
trials_csp = {cl1: apply_mix(W, trials_filt[cl1]),
              cl2: apply_mix(W, trials_filt[cl2])}

# Para visualizar os resultados do algoritimo CSP, segue o grafico log-var plotado abaixo
trials_logvar = {cl1: logvar(trials_csp[cl1]),
                 cl2: logvar(trials_csp[cl2])}
plot_logvar(trials_logvar)

"""Ao invez  de 118 canais, agora temos 118 combinacoes de canais, chamados de componentes,
que são o resultado de 118 filtros espaciais aplicados aos dados.

Os primeiros filtros maximizam a variação da primeira classe, enquanto minimizam a variação da
segunda. Os últimos filtros maximizam a variação da segunda classe, enquanto minimizam a
variação da primeira.

Isso também é visível no  gráfico PSD. O código abaixo plota o PSD para o primeiro e o último
componente, bem como um no meio:

"""

psd_l, freqs = psd(trials_csp[cl1])
psd_r, freqs = psd(trials_csp[cl2])
trials_PSD = {cl1: psd_l, cl2: psd_r}

plot_psd(trials_PSD, freqs, [0,28,-1], chan_lab=['1o Componente', 'Componente Medio', 'Ultimo
componento'], maxy=0.75 )

"""E possivel diferenciar bem as duas classes, o gráfico de dispersão é uma ferramenta útil para
visualizar essa diferenca.

Aqui ambas as classes são plotadas em um plano bidimensional: o eixo x é o primeiro componente
CSP, o eixo y é o último.

"""

def plot_scatter(left, right):
  plt.figure()
  plt.scatter(left[0,:], left[-1,:], color='b')
  plt.scatter(right[0,:], right[-1,:], color='r')
  plt.xlabel('Ultimo Componente')
  plt.ylabel('1o Componente')
  plt.legend(cl_lab)

plot_scatter(trials_logvar[cl1], trials_logvar[cl2])
```

```
"""Aplicar um classificador linear a esses dados.

Um classificador linear pode ser pensado como desenhar uma linha no gráfico acima para separar
as duas classes. Para determinar a classe para uma nova tentativa, apenas verifica-se em qual
lado da linha a tentativa estaria se plotada como acima.

Os dados são divididos em amostras de treinamento e um conjunto de teste.
O classificador ajustará um modelo (neste caso, uma linha reta) no conjunto de treinamento e
usará esse modelo para fazer previsões sobre o conjunto de teste (veja em qual lado da linha cada
tentativa do conjunto de teste cai).

Observe que o algoritmo CSP faz parte do modelo, portanto, por uma questão de justiça, ele deve
ser calculado usando apenas os dados de treinamento.
"""

# Porcentagem de amostras para utilizar no treinamento (divisao 50-50 )
train_percentage = 0.5

# Calcular o numero de amostras para cada classe em que a porcentagem acima sirva
ntrain_l = int(trials_filt[cl1].shape[2] * train_percentage)
ntrain_r = int(trials_filt[cl2].shape[2] * train_percentage)
ntrain_l = trials_filt[cl1].shape[2] - ntrain_l
ntrain_r = trials_filt[cl2].shape[2] - ntrain_r

# Dividindo o sinal filtrado de frequencia num conjunto de treinamento e testes
train = {cl1: trials_filt[cl1][:,:,:ntrain_l],
        cl2: trials_filt[cl2][:,:,:ntrain_r]}

test = {cl1: trials_filt[cl1][:,:,ntrain_l:],
        cl2: trials_filt[cl2][:,:,ntrain_r:]}


# Treinar a CSP com  o conjunto de treinamento somente
W = csp(train[cl1], train[cl2])

# Aplicar a CSP em ambos conjuntos: treinamento e testes
train[cl1] = apply_mix(W, train[cl1])
train[cl2] = apply_mix(W, train[cl2])
test[cl1] = apply_mix(W, test[cl1])
test[cl2] = apply_mix(W, test[cl2])

# Selecionar somente o primeiro e ultimo para classificacao
comp = np.array([0, -1])
train[cl1] = train[cl1][comp,:,:]
train[cl2] = train[cl2][comp,:,:]
test[cl1] = test[cl1][comp,:,:]
test[cl2] = test[cl2][comp,:,:]

# Calcular log-var
train[cl1] = logvar(train[cl1])
train[cl2] = logvar(train[cl2])
test[cl1] = logvar(test[cl1])
test[cl2] = logvar(test[cl2])
```

CENTRO UNIVERSITÁRIO CIMATEC

ESPECIALIZAÇÃO EM AUTOMAÇÃO, CONTROLE E ROBÓTICA

```
"""Para o classificador será utilizado o algoritmo de Análise Discriminante Linear (LDA).
Ele ajusta uma distribuição gaussiana para cada classe, caracterizada pela média e covariância, e
determina um plano de separação ótimo para dividir as duas. Este plano é definido como:
   r = W0*X0 + W1*X1 +...+ WnXn - b, onde:
r é a saída do classificador;
Wn são os pesos das caracteristicas;
Xn são as caracteristicas das amostras;
n é a dimensao dos dados e;
b é o deslocamento.

Neste caso, temos dados bidimensionais, então o plano de separação será uma linha:
  r = W0*X0 + W1*X1
Para determinar um rótulo de classe para uma amostra não vista, podemos calcular se o resultado
é positivo ou negativo.

"""

def train_lda(class1, class2):
  ''' Treinar o algoritiom LDA
  argumentos:
    class1: array (observacoes X caracteristicas) da classe1
    class2: array (observacoes X caracteristicas) da classe2
  retorna:
    Matriz de projecao W
    Deslocamento b
  '''
  nclasses = 2
  nclass1 = class1.shape[0]
  nclass2 = class2.shape[0]

  # Classe anterior: nester caso, as amostras tem numeros iguais de exemplos para cada classe,
  # logo ambas as calsses anteriores sao 0.5

  prior1 = nclass1 / float(nclass1 + nclass2)
  prior2 = nclass2 / float(nclass1 + nclass1)

  mean1 = np.mean(class1, axis=0)
  mean2 = np.mean(class2, axis=0)

  class1_centered = class1 - mean1
  class2_centered = class2 - mean2

  # Calcular a covariancia entre as caracteristicas
  cov1 = class1_centered.T.dot(class1_centered) / (nclass1 - nclasses)
  cov2 = class2_centered.T.dot(class2_centered) / (nclass2 - nclasses)

  W = (mean2 - mean1).dot(np.linalg.pinv(prior1*cov1 + prior2*cov2))
  b = (prior1 * mean1 + prior2 * mean2).dot(W)

  return(W,b)
```

```
def apply_lda(test, W, b):
  '''Aplicar a LDA previamenter treinada ao novo dado
  argumentos:
    test: array (caracteristicas X amostras) contendo os dados
    W: a matriz projetada W conforme calculada pela funcao tain_lda()
    b: o deslocamento b confirme calculado pela funcao train_lda()
  Retorna:
    Uma lista contendo rotulos de classe para cada amostra
  '''
  ntrials = test.shape[1]
  prediction = []
  for i in  range(ntrials):
    # A linha abaixo e uma generalizacao para:
    # result = W[0]*test[0,i] + W[1]*test[1,i] - b
    result = W.dot(test[:,i]) - b
    if result <= 0:
      prediction.append(1)
    else:
      prediction.append(2)
  return np.array(prediction)

"""Ao treinar a LDA utilizando os dados de treinamento, retorna W e b"""

W,b = train_lda(train[cl1].T, train[cl2].T)
print('W:', W)
print('b:', b)

"""Como informativo, foi recriado o gráfico de dispersão e sobreposto o limite de decisão conforme
determinado pelo classificador LDA.
O limite de decisão é a linha para a qual a saída do classificador é exatamente zero (0).
O gráfico de dispersão usou X0 como eixo x e X1 como eixo y. Para encontrar a função y=f(x) que
descreve o limite de decisão, definimos r para 0 e resolvemos para y na equação do plano de
separação:

Primeiro traçamos o limite de decisão com os dados de treinamento usados para calculá-lo:
"""

# Scatterplot like before
plot_scatter(train[cl1], train[cl2])
plt.title('Dados treinados')

# Calculate decision boundary (x,y)
x = np.arange(-5, 1, 0.1)
y = (b - W[0]*x) / W[1]

# Plot the decision boundary
plt.plot(x,y, linestyle='--', linewidth=2, color='k')
plt.xlim(-5, 1)
plt.ylim(-2.2, 1)

"""O código abaixo traça o limite com os dados de teste nos quais aplicaremos o classificador. Você
verá que o classificador cometerá alguns erros."""
```

```
plot_scatter(test[cl1], test[cl2])
plt.title('Dados Teste')
plt.plot(x,y, linestyle='--', linewidth=2, color='k')
plt.xlim(-5, 1)
plt.ylim(-2.2, 1)

"""Agora o LDA é construído e ajustado aos dados de treinamento. Agora podemos aplicá-lo aos
dados de teste. Os resultados são apresentados como uma matriz de confusão:

O número na diagonal serão as amostras que foram classificadas corretamente, quaisquer
amostras classificadas incorretamente (um falso positivo ou falso negativo) estarão nos cantos.
"""

# Apresentar matriz de confusao
conf = np.array([
    [(apply_lda(test[cl1], W, b) == 1).sum(), (apply_lda(test[cl2], W, b) == 1).sum()],
    [(apply_lda(test[cl1], W, b) == 2).sum(), (apply_lda(test[cl2], W, b) == 2).sum()],
])

print('Matriz de Confusao:')
print(conf)
print()
print('Precisao: %.3f' % (np.sum(np.diag(conf)) / float(np.sum(conf))))
```