PEDRO HENRIQUE GALDINO BOUZON

AUTOMAÇÃO DO ISOLAMENTO DE FALTAS EM LINHAS DE TRANSMISSÃO

LAVRAS – MG

2019

PEDRO HENRIQUE GALDINO BOUZON

AUTOMAÇÃO DO ISOLAMENTO DE FALTAS EM LINHAS DE TRANSMISSÃO

Tese apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Engenharia de Controle e Automação, para a obtenção do título de Bacharel.

Prof. Dr. Danton Diego Ferreira
Orientador

Prof. Dr. Belisário Nina Huallpa
Coorientador

LAVRAS – MG
2019

PEDRO HENRIQUE GALDINO BOUZON

AUTOMAÇÃO DO ISOLAMENTO DE FALTAS EM LINHAS DE TRANSMISSÃO
AUTOMATION OF FAULT ISOLATION ON POWER TRANSMISSION LINES

Tese apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Engenharia de Controle e Automação, para a obtenção do título de Bacharel.

APROVADA em 13 de Novembro de 2019.

Dr. Danton Diego Ferreira	UFLA
Dr. Belisário Nina Huallpa	UFLA
Gabriel Aparecido Fonseca	UFLA

Prof. Dr. Danton Diego Ferreira
Orientador

Prof. Dr. Belisário Nina Huallpa
Co-Orientador

LAVRAS – MG
2019

AGRADECIMENTOS

O autor gostaria de agradecer à Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) pelo suporte.

A definição de insanidade é fazer a mesma coisa repetidamente e esperar resultados diferentes.
(Albert Einstein)

RESUMO

As linhas de transmissão de energia são elementos expostos a eventos de causa natural e humana que podem levar ao seu mal funcionamento. Uma falta no sistema elétrico de potência é definida como qualquer falha que interfira no fluxo normal de corrente. Faltas acontecem principalmente quando descargas elétricas danificam os isoladores de uma linha. Este estudo provê uma abordagem para o isolamento automático das linhas contendo faltas, utilizando processamento de sinais e aprendizado de máquina. Os tipos de falta analisados são: AT, BT, CT, AB, AC, BC, ABT, ACT, BCT e ABT. O método implementado utiliza Estatísticas de Ordem Superior e Redes Neurais para classificar o tipo de falta com base nos sinais de corrente e tensão. A partir daí, uma falta no sistema é isolada pela correta atuação de relés de proteção ultrarrápidos. O método proposto necessita de cerca de 0,5 ms para determinar quais linhas devem ser isoladas. Desta forma, foi possível classificar 10 tipos de faltas com eficiência média de 97% para o conjunto de sinais analisados.

Palavras-chave: Estatística de Ordem Superior. Classificação de Faltas. Linhas de Transmissão.

ABSTRACT

Power transmission lines are elements exposed to natural and human events that can lead to malfunctions. A power system fault is defined as any failure that interferes with normal current flow. Faults occur mainly when electrical discharges damage the insulators of a line. This study provides an approach for the automatic isolation of fault-containing lines using signal processing and machine learning. The types of fault analyzed are: AT, BT, CT, AB, AC, BC, ABT, ACT, BCT and ABT. The implemented method uses Higher Order Statistics and Neural Networks to classify the fault type based on the current and voltage signals. From there, a fault in the system is isolated by the correct actuation of ultrafast protective relays. The proposed method needs about 0.5 ms to determine which lines should be isolated. Thus, it was possible to classify 10 types of faults with an average efficiency of 97% for the set of signals analyzed.

Keywords: Higher Order Statistics. Fault Classification. Power Transmission Lines.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

As linhas de transmissão de energia são elementos expostos a eventos de causa natural e humana que podem levar ao seu mal funcionamento. Uma falta é definida como qualquer falha que interfira com o fluxo normal de corrente (GRAINGER; STEVENSON, 1994). Faltas acontecem principalmente quando descargas elétricas danificam os isoladores de uma linha. Neste contexto, é importante detectar e classificar o tipo de falta, para que relés ultrarrápidos possam ser acionados e as cargas ligadas ao sistema não sejam afetadas.

Várias metodologias de detecção e classificação de faltas são encontradas na literatura. Em geral, elas diferem na técnica de extração de características e classificador adotado. Métodos baseados no domínio da frequência são os mais difundidos. Em (SILVA; SOUZA; BRITO, 2006) a Transformada Wavelet (WT) foi utilizada juntamente com uma Rede Neural Artificial (RNA). A Transformada de Fourier Discreta de Meio Ciclo (TFDMC) é adotada nos trabalhos de (JAMEHBOZORG; SHAHRTASH, 2010b) e (JAMEHBOZORG; SHAHRTASH, 2010a) em conjunto com uma árvore de decisão para classificar as faltas. O conceito da Transformada Stockwell (TS) foi utilizado em (SAMANTARAY; DASH; PANDA, 2006), (SAMANTARAY; DASH, 2008) e (SAMANTARAY, 2013). Ainda são encontrados métodos ligados ao domínio do tempo, como em (THUKARAM; KHINCHA; VIJAYNARASIMHA, 2005), onde a Análise de Componentes Principais (PCA) foi aplicada diretamente aos sinais de corrente e tensão e em (CHENG; WANG; GAO, 2015), onde foi proposto um método de extração de características baseado em Projeções Aleatórias.

Neste estudo, sinais de tensão são continuamente monitorados por um detector baseado na distância euclidiana (RIBEIRO et al., 2018). Caso uma falta seja detectada, a Estatística de Ordem Superior (EOS) é utilizada para extrair características dos sinais de corrente e tensão, na forma de cumulantes. A classificação da falta é feita por meio de uma Rede Neural Multicamadas. As novidades do

método são o uso dos cumulantes na forma proposta por (FERREIRA et al., 2011) para classificação de faltas em linhas de transmissão. A vantagem desta abordagem está no fato da EOS ser imune a ruídos gaussianos e possuírem uma boa capacidade de representação de processos não lineares.

O referencial teórico é abordado no próximo capítulo. O capítulo 3 descreve o método proposto. No capítulo 4 detalha-se a base de dados utilizada. No capítulo 5 os resultados são apresentados e discutidos. Finalmente, são apresentadas as conclusões no capítulo 6.

2 REFERENCIAL TEÓRICO

2.1 Estatísticas de Ordem Superior

As Estatísticas de Ordem Superior podem ser descritas por meio de cumulantes ou momentos. O primeiro caso se aplica a sinais aleatórios, enquanto o segundo a sinais determinísticos. Esta técnica é melhor explorada quando utilizada em processos de distribuição não gaussiana e sistemas não lineares (FERREIRA et al., 2011), (RIBEIRO et al., 2007).

As expressões que descrevem os cumulantes de segunda, terceira e quarta ordens de um sinal aleatório $x[n]$ com $E\{x[n]\} = 0$ são representadas pelas Equações (2.1), (2.2) e (2.3) (MENDEL, 1991).

$$c_{2,x}[i] = E\{x[n]x[n+1]\} \tag{2.1}$$

$$c_{3,x}[i] = E\{x[n]x^2[n+1]\} \tag{2.2}$$

$$c_{4,x}[i] = E\{x[n]x^3[n+1]\} - 3c_{2,x}[i]c_{2,x}[0] \tag{2.3}$$

Para um vetor finito de tamanho N, aproximações estocásticas (KUSHNER; YIN, 2003) resultam nas expressões

$$\hat{c}_{2,x}[i] = \frac{2}{N}\sum_{n=0}^{N/2-1} x[n]x[n+i], \tag{2.4}$$

$$\hat{c}_{3,x}[i] = \frac{2}{N}\sum_{n=0}^{N/2-1} x[n]x^2[n+i] \tag{2.5}$$

e

$$\hat{c}_{4,x}[i] = \frac{2}{N}\sum_{n=0}^{N/2-1} x[n]x^3[n+i] - \frac{2}{N^2}\sum_{n=0}^{N/2-1} x[n]x[n+i]\sum_{n=0}^{N/2-1} x^2[n], \tag{2.6}$$

onde $i = 0, 1, \cdots, \frac{N}{2} - 1$. Infelizmente, as Equações (2.4), (2.5) e (2.6) não podem ser usadas para $i > \frac{N}{2} - 1$ porque o índice $n+i$ ultrapassa o tamanho do vetor. Devido a esta limitação, aproximações alternativas foram introduzidas por (RIBEIRO et al., 2007). Nesta nova abordagem, os cumulantes são estimados pelas seguintes equações

$$\hat{c}_{2,x}[i] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]x[mod(n+i,N)], \tag{2.7}$$

$$\hat{c}_{3,x}[i] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]x^2[mod(n+i,N)] \tag{2.8}$$

e

$$\hat{c}_{4,x}[i] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]x^3[mod(n+i,N)] - \frac{3}{N^2}\sum_{n=0}^{N-1} x[n]x[mod(n+i,N)]\sum_{n=0}^{N-1} x^2[n], \tag{2.9}$$

onde $mod(n+1,N)$ é o resto inteiro da divisão de $n+i$ por N. O uso do operador *mod* implica na assunção de que o vetor $x[n]$ é periódico.

2.2 Razão Discriminante de Fisher

A Razão Discriminante de Fisher é um critério de seleção de características baseado na forma como as amostras vetor de características se distribuem no espaço l-dimensional, onde l representa o tamanho deste vetor. A Equação 2.10

mostra a Razão Discriminante de Fisher para o caso bidimensional, onde μ_1, σ_1, μ_2 e σ_2 são a média e variância das classes *1* e *2* respectivamente (THEODORIDIS et al., 2010). Nota-se que as melhores características são aquelas cujas médias das duas classes são as mais distantes possíveis, levando em conta também a variância destas.

$$\mathbf{J} = \frac{(\mu_1 - \mu_2)^2}{(\sigma_1^2 + \sigma_2^2)} \tag{2.10}$$

Para o caso multidimensional, formas baseadas na médias podem ser utilizadas (THEODORIDIS et al., 2010). A Equação 2.11 mostra a Razão Discriminante de Fisher para o caso multidimensional, onde μ_i, σ_i, μ_j e σ_j são a média e variância das classes i e j respectivamente. Nesta situação, as características são analisadas duas a duas e a média da importância destas é utilizada.

$$\mathbf{J} = \sum_{\mathbf{i}}^{\mathbf{N}} \sum_{\mathbf{j} \neq \mathbf{i}}^{\mathbf{N}} \frac{(\mu_\mathbf{i} - \mu_\mathbf{j})^2}{(\sigma_\mathbf{i}^2 + \sigma_\mathbf{j}^2)} \tag{2.11}$$

2.3 Coeficiente de Correlação de Pearson

O Coeficiente de Correlação de Pearson é uma medida do grau de correlação entre duas variáveis X e Y. Possui valores entre +1 e -1, onde +1 representa correlação total positiva, 0 se refere a nenhuma correlação e -1 à correlação total negativa (FISZ; BARTOSZYŃSKI, 2018).

O Coeficiente de Pearson é dado pela covariância das duas variáveis dividido pelo produto dos seus desvios padrão. A Equação 2.12 descreve matematicamente as operações efetuadas para o cálculo deste (FISZ; BARTOSZYŃSKI, 2018).

$$\mathbf{R_{ij}} = \frac{\sigma_{ij}}{\sigma_{ii} * \sigma_{jj}} \tag{2.12}$$

2.4 Perceptron

O Perceptron é um algoritmo de aprendizado de máquina para determinar uma função que classifica um valor de entrada como pertencente à uma determinada classe (DUDA; HART; STORK, 2012). A Equação 2.13 mostra a função discriminante do Perceptron, onde w e b representam os pesos e viés do classificador respectivamente.

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{caso contrário} \end{cases} \tag{2.13}$$

Para determinar os valores de w e b na Equação 2.13 é necessário um conjunto de dados de treinamento. Este dados são utilizados em conjunto com um algoritmo de otimização para determinar as regiões no espaço de entrada onde o classificador acusa a ocorrência de uma classe. A função de custo de Perceptron, a ser minimizada durante o treinamento, é dada pela Equação 2.15, onde w representa um vetor de pesos e χ o conjunto de amostras que foram classificadas de forma errada.

$$J_p(\mathbf{w}) = \sum_{\mathbf{x} \in \chi} (\mathbf{w}^{\mathbf{t}} \mathbf{x}) \tag{2.14}$$

O algoritmo de otimização mais comumente utilizado para treinamento do Perceptron é o Grandiente Descendente. Neste método, o vetor de pesos é atualizado na direção do vetor gradiente da função de custo em relação ao vetor de pesos w (DUDA; HART; STORK, 2012). Assim, tem-se que

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \sum_{x \in \chi} (\mathbf{x}) \tag{2.15}$$

onde η representa a taxa de aprendizado.

2.5 Perceptron Multicamadas

O Perceptron Multicamadas é uma classe de Redes Neurais Artificias que consiste em pelo menos três camadas: de entrada, escondida e de saída. Com excessão da camada de entrada, todas as outras possuem neurônios com função de ativação não linear. O treinamento da rede é feito por uma técnica chamada de Backpropagation (DUDA; HART; STORK, 2012).

As funções de ativação mais utilizadas são $\tanh(x)$ e $(1+e^{-x})^{-1}$. Estas funções tem um formato interessante para o treinamento porque possuem assíntotas horizontais que limitam valores muito grandes. A primeira, chamada tangente hiperbólica, tem valores limitados entre -1 e 1. A segunda, chamada função logistica se limita a 0 e 1.

Uma Rede Perceptron de Multiplas Camadas é definida pela dimensão do vetor de entrada, número de camadas escondidas, número de classes de saída, quantidade de neurônios em cada camada e funções de ativação utilizadas. Além disso, o processo de treinamento depende da taxa de aprendizado e do otimizador utilizado.

A configuração ideal da rede depende muito do problema a ser resolvido, mas, de forma geral, quanto maior for a razão do número de características de entrada pelo número de parâmetros livres do classificador, melhor será a generalização para dados desconhecidos (DUDA; HART; STORK, 2012).

2.6 Python

A linguagem de programação Python vem se tornando atrativa para a comunidade científica devido à sua filosofia de código aberto e à grande quantidade de pessoas disponibilizam bibliotecas de forma gratuita. Atualmente, existem inúmeros pacotes para ciência de dados em Python mas os mais utilizados são: Numpy, Scikit-Learn e Keras.

Numpy é um pacote para computação científica de operações elementares com alta performace que fornece um objeto do tipo array N-dimensional, funções de algebra linear, transformadas de fourier, geração de números aleatórios e funções estatísticas. O projeto Numpy é mantido pela Universidade de Berkeley e pela empresa Quansight.

Criado em 2006, o Scikit-Learn é um pacote que traz ferramentas para análise de dados e foi construído utilizando o Numpy. Seu conjunto de funções abrange áreas como: classificação, regressão, agrupamento de dados, seleção de características, escolha de modelos e preprocessamento.

Keras é uma biblioteca para Deep Learning que permite a execução do código na GPU, o que acelera muito o treinamento. Com suas funções, é possível criar redes com múltiplas camadas utilizando diferentes configurações de números de nós, funções de ativação e algoritmos de otimização. Entre as camadas disponíveis nesta biblioteca estão as: Convolucionais, Densas, Recorrentes e de Normalização.

3 MÉTODO PROPOSTO

O método proposto é organizado como se segue. Sinais de tensão são monitorados em tempo real por um detector baseado na distância euclidiana (RIBEIRO et al., 2018). Se uma condição anormal for encontrada, os sinais de tensão e corrente são segmentados e características são extraídas destes na forma de cumulantes, com base nas Equações (2.7), (2.8) e (2.9). Finalmente, um Rede Neural Multicamadas realiza a classificação em uma das seguintes faltas: AT, BT, CT, AB, AC, BC, ABT, ACT, BCT e ABT.

3.1 Detecção

O detector utilizado neste trabalho foi proposto por (RIBEIRO et al., 2018) onde o monitoramento se dá por meio de janelas deslizantes (amostra a amostra) com tamanho igual a um ciclo do sinal fundamental. Neste método, considera-se o sinal de tensão como um ponto d-dimensional, onde d é o tamanho da janela monitorada. Calcula-se então a distância euclidiana deste ponto ao centro do espaço, definido como $\mathbf{c} = [0\,0\,0\cdots 0_d]$. A Equação (3.1) descreve a operação de extração de características, onde $\mathbf{v}$ representa a janela monitorada e $\mathbf{c}$ o centro do espaço. A partir desta análise, é possível obter uma região hiperesférica no espaço d-dimensional que modela sinais sem distúrbios (RIBEIRO et al., 2018).

$$r = \|\mathbf{v} - \mathbf{c}\|^2 \tag{3.1}$$

Com objetivo de definir a região onde não há presença de falta, sinais nominais contendo fases de -180º a 180º e SNR (*signal-to-noise ratio*) de 60dB, são utilizados para determinar os valores de r para esta situação. A partir destes valores, os limites inferior e superior são adotados como

$$r_{max} = \bar{r} + 3 * \sigma \tag{3.2}$$

e

$$r_{min} = \bar{r} - 3 * \sigma \tag{3.3}$$

onde $\bar{r}$ é o valor médio e σ o desvio padrão de r.

Os limites encontrados durante a fase de projeto definem uma região que modela os sinais nominais e a qualquer evento fora desta é associado uma pertubação.

Caso o valor de r para um sinal seja maior que r_{max} ou menor que r_{min}, janelas com sessenta e quatro pontos pré-falta (um quarto de ciclo) e oito pontos pós-falta (um trinta e dois avos de ciclo) dos sinais de corrente e tensão são enviadas para a etapa de extração de características.

3.2 Extração de Características

O uso da Estatística de Ordem Superior para obter informações relevantes sobre sinais elétricos em sistemas de potência tem se mostrado uma boa alternativa para extração de características (ROSA; MORENO-MUNOZ, 2009), (NIKIAS; MENDEL, 1993). Para um vetor de características finito, tais estatísticas podem ser aproximadas pelas Equações (2.7), (2.8) e (2.9).

Como o conjunto de características gerado por esta abordagem é muito grande, o Discriminante Linear de Fisher (DLF) foi utilizado para selecionar os cumulantes que melhor discriminam entre as classes. O DLF vem sendo aplicado em problemas de classificação para selecionar e reduzir os dados de entrada (BARBOSA et al., 2016), (NAVES; BARBOSA; FERREIRA, 2016), (GUEDES et al., 2015). A Equação (3.4) apresenta a função de custo do Discriminante Linear de Fisher para um problema com N classes (THEODORIDIS et al., 2010), onde μ_i, σ_i, μ_j e σ_j são a média e variância das classes i e j respectivamente.

$$\mathbf{J} = \sum_{\mathbf{i}}^{\mathbf{N}} \sum_{\mathbf{j} \neq \mathbf{i}}^{\mathbf{N}} \frac{(\mu_\mathbf{i} - \mu_\mathbf{j})^\mathbf{2}}{(\sigma_\mathbf{i}^\mathbf{2} + \sigma_\mathbf{j}^\mathbf{2})} \tag{3.4}$$

Uma vez que o Discriminante Linear de Fisher não remove redundância, outra técnica deve ser utilizada para este fim. Neste trabalho, um filtro baseado no Coeficiente de Pearson foi implementado para limitar a correlação entre as características selecionadas. A Equação (3.5) apresenta o coeficiente de Pearson calculado para as características c_i e c_j com matrizes de autocorrelação e correlação cruzada iguais a σ_{ii}, σ_{jj} e σ_{ij}. Quanto mais próximo da unidade for o valor absoluto de R_{ij}, maior é a correlação as características.

$$\mathbf{R_{ij}} = \frac{\sigma_{ij}}{\sigma_{ii} * \sigma_{jj}} \tag{3.5}$$

3.3 Classificação

Nesta etapa, os cumulantes selecionados servem de base para o treinamento de uma Rede Neural Multicamadas. A arquitetura do classificador é apresentada na Tabela 3.1. Foi adotada uma arquitetura simples para melhorar a capacidade de generalização do classificador (DUDA; HART; STORK, 2012). O classificador conta com duas camadas. A camada escondida possui oito neurônios com função de ativação Tangente Hiperbólica Sigmoidal, descrita pela Equação (3.6). A camada de saída possui dez neurônios, representando as possíveis faltas, com função de ativação Softmax, descrita pela Equação (3.7).

Tabela 3.1 – Arquitetura do classificador.

Camada	Neurônios	Função de Ativação
Camada escondida	8	Tangente Hiperbólica
Camada de saída	10	Softmax

Fonte: do Autor (2019)

$$g_1(x) = \frac{2}{1+e^{-2x}} - 1 \tag{3.6}$$

$$g_2(x_i) = \frac{e^{-x_i}}{\sum_{j=1}^{10} e^{-x_j}} \tag{3.7}$$

Para treinamento da rede, aplicou-se o otimizador Adam, proposto por (KINGMA; BA, 2014). Os hiper parâmetros deste algoritmo foram mantidos nos valores sugeridos pelos autores do método. A taxa de aprendizado utilizada foi de 0.01.

4 BASE DE DADOS

O sistema de transmissão utilizado para simular os dados conta com 2 linhas de 300km cada, tensão de 500 kV e SNR (signal-noise-ratio) de 60dB. A frequência fundamental adotada foi de 60Hz e frequência de amostragem de 15360Hz, resultando em 256 pontos por ciclo. Foram simulados dez tipos de falta: AT, BT, CT, AB, AC, BC, ABT, ACT, BCT e ABT. A simulação foi feita por meio do software Simulink.

Ao todo, 950 sinais contendo falta foram gerados. A Tabela 4.1 descreve as 5 bases utilizadas na simulação. Em cada base foram gerados 19 sinais de cada tipo de falta com ângulos de incidência de 0 à 180º em passos de 10º.

Tabela 4.1 – Parâmetros utilizados para simular as faltas.

Resistência (Ohms)	Distância à Barra Local(Km)
1	20
1	150
1	280
50	150
100	150

Fonte: do Autor (2019)

5 RESULTADOS E DISCUSSÃO

5.1 Detecção

Nesta etapa, os limiares inferior e superior encontrados para definir a região que modela os sinais nominais foram $r_{max} = 2.642 * 10^{13}$ e $r_{min} = 2.637 * 10^{13}$.

A Figura 5.1 mostra um exemplo dos sinais de tensão e corrente segmentados pelo detector na ocorrência de uma falta. A linha preta tracejada indica o ponto em que foi detectado a falta. A partir deste ponto, um quarto de ciclo pré-falta e um trinta e dois avos de ciclo pós-falta formam a janela a ser enviada para a etapa de extração de características. Todos os 950 segmentos contendo falta foram detectados.

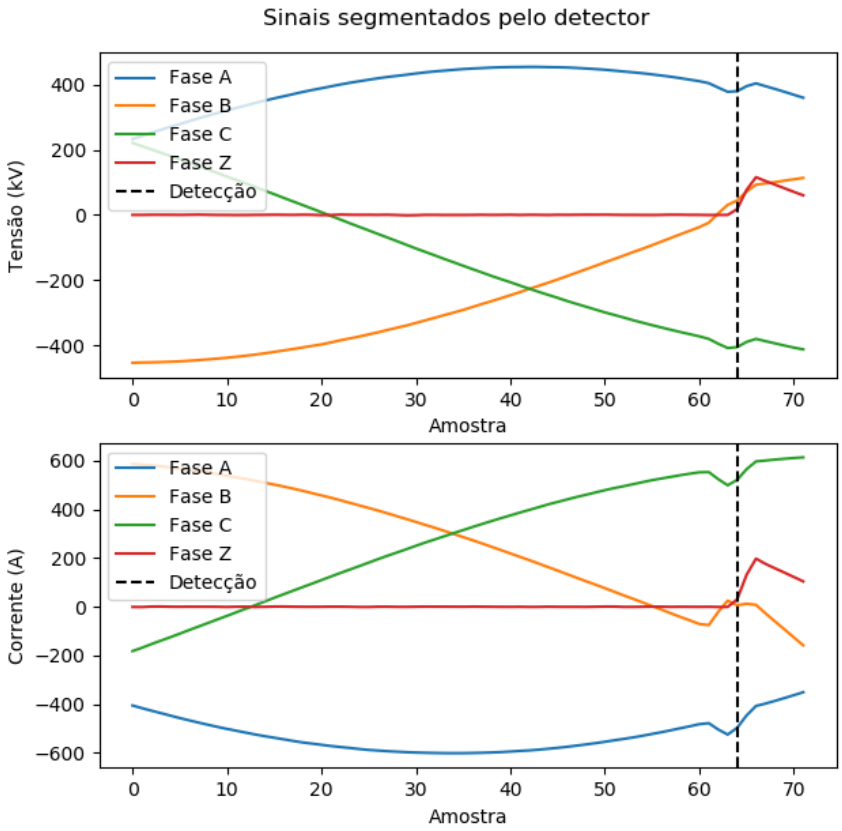
O detector apresentou um atraso de detecção igual ao tempo necessário para que a deformação no sinal de tensão gere uma distância maior que os limiares estabelecidos. Ao ser aplicado no banco de dados, o atraso do instante de detecção em relação ao valor real foi de 4.13 ± 0.95 amostras. Em termos de fração de ciclo, este valor representa um erro médio de sessenta e quatro avos com desvio padrão de um duzentos e cinquenta e seis avos de ciclo.

5.2 Extração de Características

Os cumulantes de segunda, terceira e quarta ordem foram extraídos das fases A, B, C e Z dos sinais de tensão e corrente. Para cada cumulante foi gerado um conjunto de *N* características, onde N é o tamanho da janela enviada pelo detector. Devido à simetria dos cumulantes de segunda ordem, a primeira metade destes foi descartada. Foi ainda extraído o valor médio quadrático e o valor máximo absoluto dos sinais. Desta forma, foram obtidas 1456 características.

Para reduzir o número de características e assim a complexidade computacional do método, o Discriminante Linear de Fisher foi aplicado ao conjunto total de características. A Figura 5.2 mostra o resultado deste método. Cada ponto

Figura 5.1 – Sinais de tensão e corrente segmentados pelo detector.



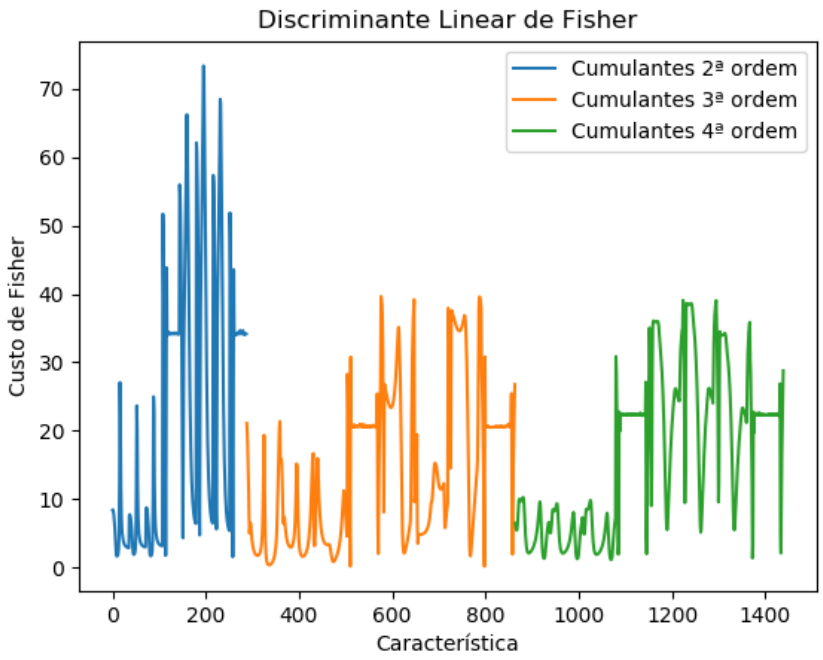
Fonte: Do Autor (2019)

nesta figura representa o valor de importância dada pelo critério de Fisher a uma característica.

A partir do DLF e do coeficiente de correlação de Pearson, foram selecionadas vinte e cinco características com correlação máxima de cinquenta porcento entre elas. A Figura 5.3 mostra a quantidade de características selecionadas de cada tipo.

Para verificar se os cumulantes trazem informação útil quanto à presença de falta, a distribuição amostral de um cumulante foi aproximada por uma gaussiana. A Figura 5.4 mostra o resultado. Este cumulante foi extraído da fase C do

Figura 5.2 – Função custo do Discriminante Linear de Fisher

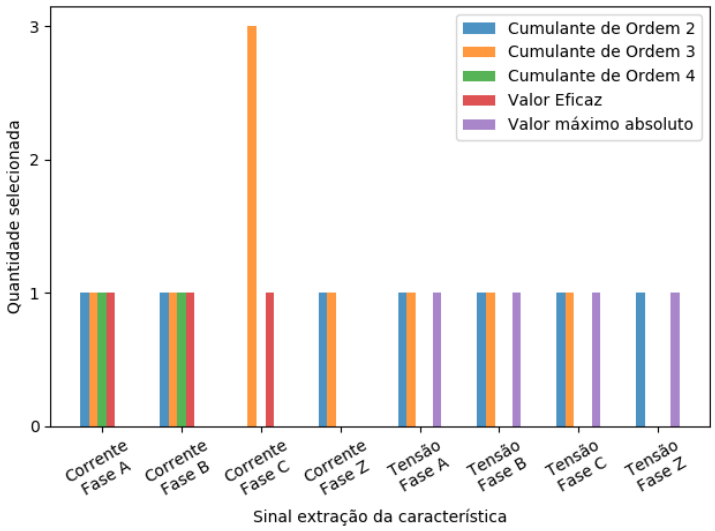


Fonte: Do Autor (2019)

sinal de corrente. Nota-se que as distribuições das classes que contém a fase C estão situadas mais à esquerda, enquanto as demais à direita. Estas diferenças são interessantes no problema de classificação.

A Figura 5.5 mostra o espaço gerado pelas três melhores características selecionadas (C1, C2 e C3) para diferentes quantidades de dados pós falta. Com 1 ciclo pós falta, há uma separação linear entre as classes. Utilizando pelo menos ⅛ ciclo há uma boa separação. Nos demais casos, as sobreposições se tornam mais evidentes. Apesar disso, quando todas as 25 características foram utilizadas, o classificador apresentou bons resultados em todos os casos.

Figura 5.3 – Sinal de origem, ordem e quantidade dos cumulantes selecionados.



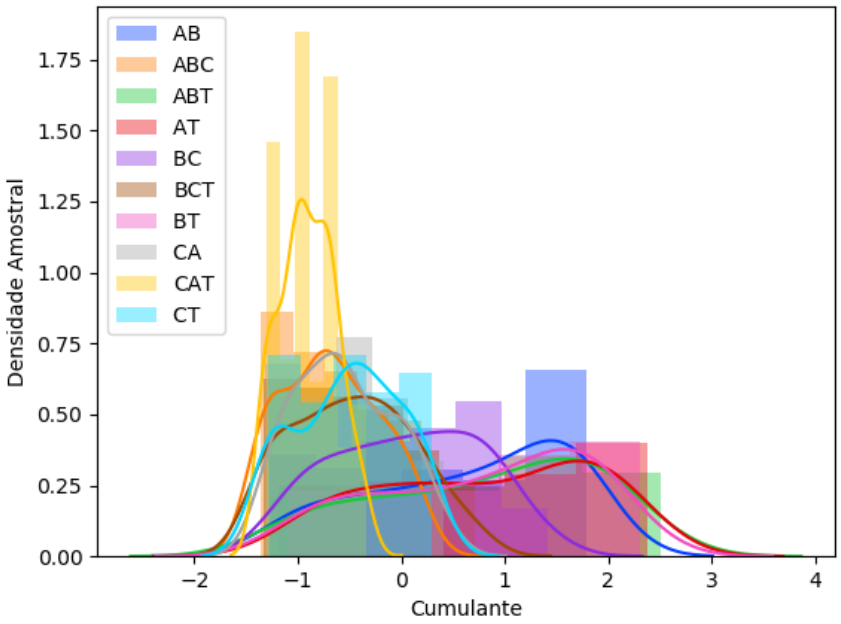
Fonte: Do Autor (2019)

5.3 Classificação

Para validação do classificador foi utilizado a metodologia K-Fold com 10 divisões. A Figura 5.6 mostra a porcentagem de acerto para diferentes quantidades de pontos pós falta na janela segmentada pelo detector. Como não se sabia inicialmente quantos ciclos seriam necessários, este número foi diminuído gradativamente de 1 até 1/32 ciclos. Foi obtido 100% (0,00%) de acerto para 1/2 e 1/8 ciclo pós falta. 99,78% (0,43%) para 1, 1/4 e 1/16 ciclo pós falta e 97,66% (1,15%) para 1/32 ciclo pós falta.

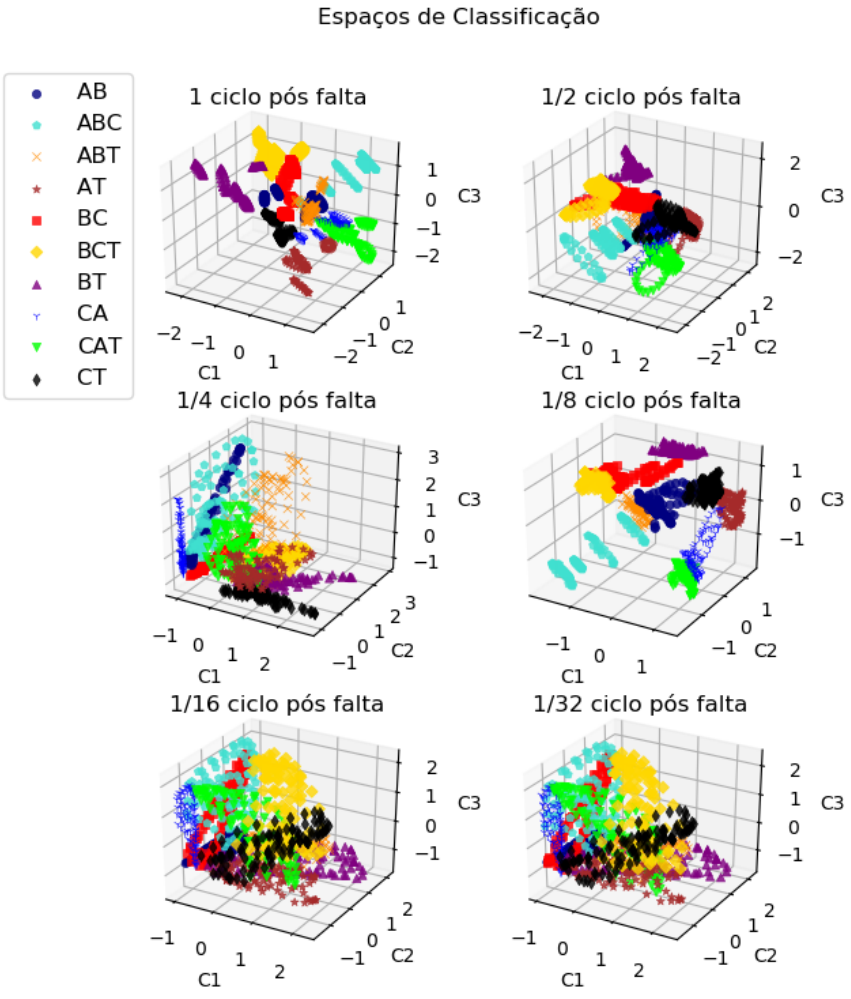
A partir destes resultados, foi escolhido utilizar 1/32 ciclos de dados após a detecção. Para este caso, a validação também foi feita por matriz de confusão. O conjunto de teste foi formado por 30% dos dados. O resultado é apresentado na Figura 5.7. Para as faltas monofásicas e as faltas CA e ABT o resultado da classificação foi de 100%. As demais faltas apresentaram porcentagens de acerto entre 96% e 97%.

Figura 5.4 – Densidade amostral de um cumulante extraído da fase C do sinal de corrente.



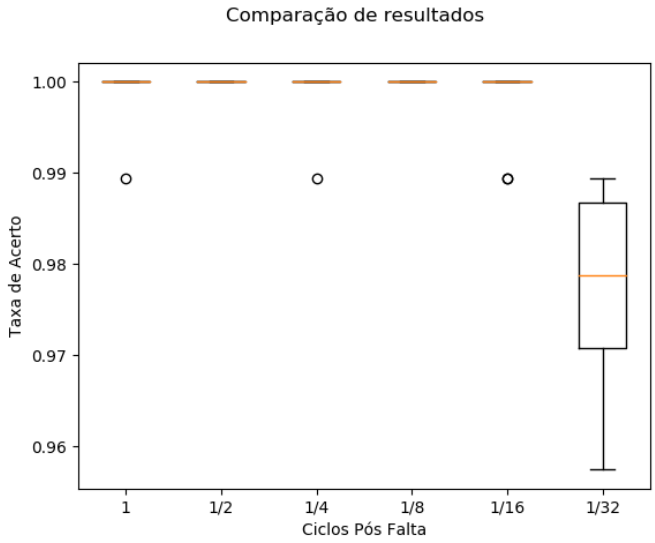
Fonte: Do Autor (2019)

Figura 5.5 – Espaços de classificação para diferentes quantidades de ciclos pós falta.



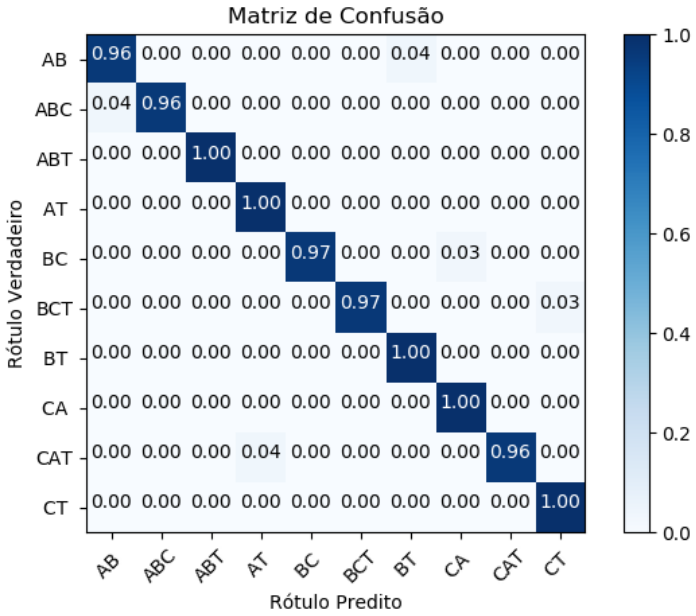
Fonte: Do Autor (2019)

Figura 5.6 – Variação da porcentagem de acerto com o número de ciclos pós falta utilizado.



Fonte: Do Autor (2019)

Figura 5.7 – Matriz de confusão.



Fonte: Do Autor (2019)

6 CONSIDERAÇÕES FINAIS

Este trabalho propôs um método de classificação baseado em Estatística de Ordem Superior que utiliza apenas ½ ciclos pós falta. Neste sentido, foi possível classificar 10 tipos de falta, incluindo as monofásicas, bifásicas e trifásicas. Para reduzir a complexidade do classificador, um detector baseado na distancia euclidiana foi implementado.

Deve ser mencionado que os cumulantes de ordem três e quatro, utilizados como entrada para o classificador, são imunes a ruídos gaussianos, o que confere robustez ao método.

Em trabalhos futuros, pretende-se analisar a complexidade computacional do método e implementar um classificador que dê mais prioridade para as faltas monofásicas.

REFERÊNCIAS

BARBOSA, T. S. et al. Fault detection and classification in cantilever beams through vibration signal analysis and higher-order statistics. **Journal of Control, Automation and Electrical Systems**, Springer, v. 27, n. 5, p. 535–541, 2016.

CHENG, L.; WANG, L.; GAO, F. Power system fault classification method based on sparse representation and random dimensionality reduction projection. In: IEEE. **2015 IEEE Power & Energy Society General Meeting**. [S.l.], 2015. p. 1–5.

DUDA, R. O.; HART, P. E.; STORK, D. G. **Pattern classification**. [S.l.]: John Wiley & Sons, 2012.

FERREIRA, D. D. et al. Exploiting higher-order statistics information for power quality monitoring. **Power Quality: Intech Open Access Publisher**, p. 345–362, 2011.

FISZ, M.; BARTOSZYŃSKI, R. **Probability theory and mathematical statistics**. [S.l.]: J. wiley, 2018. v. 3.

GRAINGER, J.; STEVENSON, W. **Power System Analysis**. [S.l.]: McGraw-Hill, 1994. (Electrical engineering series). ISBN 9780071133388.

GUEDES, J. D. S. et al. Non-intrusive appliance load identification based on higher-order statistics. **IEEE Latin America Transactions**, IEEE, v. 13, n. 10, p. 3343–3349, 2015.

JAMEHBOZORG, A.; SHAHRTASH, S. A decision tree-based method for fault classification in double-circuit transmission lines. **IEEE Transactions on Power Delivery**, IEEE, v. 25, n. 4, p. 2184–2189, 2010.

JAMEHBOZORG, A.; SHAHRTASH, S. M. A decision-tree-based method for fault classification in single-circuit transmission lines. **IEEE Transactions on Power Delivery**, IEEE, v. 25, n. 4, p. 2190–2196, 2010.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014.

KUSHNER, H.; YIN, G. G. **Stochastic approximation and recursive algorithms and applications**. [S.l.]: Springer Science & Business Media, 2003. v. 35.

MENDEL, J. M. Tutorial on higher-order statistics (spectra) in signal processing and system theory: Theoretical results and some applications. **Proceedings of the IEEE**, Ieee, v. 79, n. 3, p. 278–305, 1991.

NAVES, R.; BARBOSA, B. H.; FERREIRA, D. D. Classification of lung sounds using higher-order statistics: A divide-and-conquer approach. **Computer methods and programs in biomedicine**, Elsevier, v. 129, p. 12–20, 2016.

NIKIAS, C. L.; MENDEL, J. M. Signal processing with higher-order spectra. **IEEE Signal processing magazine**, IEEE, v. 10, n. 3, p. 10–37, 1993.

RIBEIRO, E. G. et al. Real-time system for automatic detection and classification of single and multiple power quality disturbances. **Measurement**, Elsevier, v. 128, p. 276–283, 2018.

RIBEIRO, M. V. et al. Detection of disturbances in voltage signals for power quality analysis using hos. **EURASIP Journal on Applied Signal Processing**, Hindawi Publishing Corp., v. 2007, n. 1, p. 177–177, 2007.

ROSA, J. J. G. de la; MORENO-MUNOZ, A. Higher-order characterization of power quality transients and their classification using competitive layers. In: IEEE. **2009 Compatibility and Power Electronics**. [S.l.], 2009. p. 390–395.

SAMANTARAY, S. A systematic fuzzy rule based approach for fault classification in transmission lines. **Applied soft computing**, Elsevier, v. 13, n. 2, p. 928–938, 2013.

SAMANTARAY, S.; DASH, P. Pattern recognition based digital relaying for advanced series compensated line. **International Journal of Electrical Power & Energy Systems**, Elsevier, v. 30, n. 2, p. 102–112, 2008.

SAMANTARAY, S.; DASH, P.; PANDA, G. Fault classification and location using hs-transform and radial basis function neural network. **Electric Power Systems Research**, Elsevier, v. 76, n. 9-10, p. 897–905, 2006.

SILVA, K.; SOUZA, B. A.; BRITO, N. S. Fault detection and classification in transmission lines based on wavelet transform and ann. **IEEE Transactions on Power Delivery**, IEEE, v. 21, n. 4, p. 2058–2063, 2006.

THEODORIDIS, S. et al. **Introduction to pattern recognition: a matlab approach**. [S.l.]: Academic Press, 2010.

THUKARAM, D.; KHINCHA, H.; VIJAYNARASIMHA, H. Artificial neural network and support vector machine approach for locating faults in radial distribution systems. **IEEE Transactions on Power Delivery**, IEEE, v. 20, n. 2, p. 710–721, 2005.

APÊNDICE A – Artigo

Automação do Isolamento de Faltas em Linhas de Transmissão

Pedro H.G. Bouzon
Departamento de Automática
Universidade Federal de Lavras
Lavras, Minas Gerais, Brasil, 37200-000
Email: peubouzon@gmail.com

Danton Diego Ferreira
Departamento de Automática
Universidade Federal de Lavras
Lavras, Minas Gerais, Brasil, 37200-000
Email: danton@ufla.br

José Manuel de Seixas
Departamento de Engenharia
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Rio de Janeiro,
Brasil, 21941-901
Email: seixas@lps.ufrj.br

Flávio Bezerra Costa
Escola de Ciência e Tecnologia
Universidade Federal do Rio Grande do Norte
Natal, Rio Grande do Norte, Brasil, 59064-741
Email: flaviocosta@ect.ufrn.br

Mônica Maria Leal
Escola de Ciência e Tecnologia
Universidade Federal do Rio Grande do Norte
Natal, Rio Grande do Norte, Brasil, 59064-741
Email: monicamleal@gmail.com

***Resumo*—It is known that the study of power transmission lines has been the subject of several researches aiming to provide relevant information to users of electrical systems. This study provides an approach for fault classification using higher order statistics and a neural network based classifier. A detector based on euclidean distance was implemented to reduce the complexity of the classifier. This implementation enables real-time execution because only ½₃₂ cycles of post-fault data are used in feature extraction. It was able to classify 10 classes of faults with global efficiency upper to 97%.**

***Keywords*—Estatística de Ordem Superior; Classificação de Faltas; Linhas de Transmissão**

I. Introdução

As linhas de transmissão de energia são elementos expostos a eventos de causa natural e humana que podem levar ao seu mal funcionamento. Uma falta é definida como qualquer falha que interfira com o fluxo normal de corrente [1]. Faltas acontecem principalmente quando descargas elétricas danificam os isoladores de uma linha. Neste contexto, é importante detectar e classificar o tipo de falta, para que relés ultrarrápidos possam ser acionados e as cargas ligadas ao sistema não sejam afetadas.

Várias metodologias de detecção e classificação de faltas são encontradas na literatura. Em geral, elas diferem na técnica de extração de características e classificador adotado. Métodos baseados no domínio da frequência são os mais difundidos. Em [2] a Transformada Wavelet (WT) foi utilizada juntamente com uma Rede Neural Artificial (RNA). A Transformada de Fourier Discreta de Meio Ciclo (TFDMC) é adotada nos trabalhos de [3] e [4] em conjunto com uma árvore de decisão para classificar as faltas. O conceito da Transformada Stockwell (TS) foi utilizado em [5], [6] e [7]. Ainda são encontrados métodos ligados ao domínio do tempo, como em [8], onde a Análise de Componentes Principais (PCA) foi aplicada diretamente aos sinais de corrente e tensão e em [9], onde foi proposto um método de extração de características baseado em Projeções Aleatórias.

Neste estudo, sinais de tensão são continuamente monitorados por um detector baseado na distância euclidiana [10]. Caso uma falta seja detectada, a Estatística de Ordem Superior (EOS) é utilizada para extrair características dos sinais de corrente e tensão, na forma de cumulantes. A classificação da falta é feita por meio de uma Rede Neural Multicamadas. As novidades do método são o uso dos cumulantes na forma proposta por [11] para classificação de faltas em linhas de transmissão. A vantagem desta abordagem está no fato da EOS ser imune a ruídos gaussianos e possuírem uma boa capacidade de representação de processos não lineares.

A Estatística de Ordem Superior é explicada na próxima seção. A seção III descreve o método proposto. Na seção IV detalha-se a base de dados utilizada. Na seção V os resultados são apresentados e discutidos. Finalmente, são apresentadas as conclusões na seção VI.

II. Estatística de Ordem Superior

As Estatísticas de Ordem Superior podem ser descritas por meio de cumulantes ou momentos. O primeiro caso se aplica a sinais aleatórios, enquanto o segundo a sinais determinísticos. Esta técnica é melhor explorada quando utilizada em processos de distribuição não gaussiana e sistemas não lineares [11], [12].

As expressões que descrevem os cumulantes de segunda, terceira e quarta ordens de um sinal aleatório $x[n]$ com $E\{x[n]\} = 0$ são representadas pelas Equações (1), (2) e (3) [13].

$$c_{2,x}[i] = E\{x[n]x[n+1]\} \tag{1}$$

$$c_{3,x}[i] = E\{x[n]x^2[n+1]\} \tag{2}$$

$$c_{4,x}[i] = E\{x[n]x^3[n+1]\} - 3c_{2,x}[i]c_{2,x}[0] \quad (3)$$

Para um vetor finito de tamanho N, aproximações estocásticas [14] resultam nas expressões

$$\hat{c}_{2,x}[i] = \frac{2}{N}\sum_{n=0}^{N/2-1} x[n]x[n+i], \quad (4)$$

$$\hat{c}_{3,x}[i] = \frac{2}{N}\sum_{n=0}^{N/2-1} x[n]x^2[n+i] \quad (5)$$

e

$$\hat{c}_{4,x}[i] = \frac{2}{N}\sum_{n=0}^{N/2-1} x[n]x^3[n+i] - \frac{2}{N^2}\sum_{n=0}^{N/2-1} x[n]x[n+i]\sum_{n=0}^{N/2-1} x^2[n], \quad (6)$$

onde $i = 0, 1, \cdots, \frac{N}{2} - 1$. Infelizmente, as Equações (4), (5) e (6) não podem ser usadas para $i > \frac{N}{2} - 1$ porque o índice $n + i$ ultrapassa o tamanho do vetor. Devido a esta limitação, aproximações alternativas foram introduzidas por [12]. Nesta nova abordagem, os cumulantes são estimados pelas seguintes equações

$$\hat{c}_{2,x}[i] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]x[mod(n+i,N)], \quad (7)$$

$$\hat{c}_{3,x}[i] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]x^2[mod(n+i,N)] \quad (8)$$

e

$$\hat{c}_{4,x}[i] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]x^3[mod(n+i,N)] - \frac{3}{N^2}\sum_{n=0}^{N-1} x[n]x[mod(n+i,N)]\sum_{n=0}^{N-1} x^2[n], \quad (9)$$

onde $mod(n+1, N)$ é o resto inteiro da divisão de $n + i$ por N. O uso do operador mod implica na assunção de que o vetor $x[n]$ é periódico.

III. Método Proposto

O método proposto é organizado como se segue. Sinais de tensão são monitorados em tempo real por um detector baseado na distância euclidiana [10]. Se uma condição anormal for encontrada, os sinais de tensão e corrente são segmentados e características são extraídas destes na forma de cumulantes, com base nas Equações (7), (8) e (9). Finalmente, um Rede Neural Multicamadas realiza a classificação em uma das seguintes faltas: AT, BT, CT, AB, AC, BC, ABT, ACT, BCT e ABT.

A. Detecção

O detector utilizado neste trabalho foi proposto por [10] onde o monitoramento se dá por meio de janelas deslizantes (amostra a amostra) com tamanho igual a um ciclo do sinal fundamental. Neste método, considera-se o sinal de tensão como um ponto d-dimensional, onde d é o tamanho da janela monitorada. Calcula-se então a distância euclidiana deste ponto ao centro do espaço, definido como $\mathbf{c} = [0\,0\,0\cdots 0_d]$. A Equação (10) descreve a operação de extração de características, onde **v** representa a janela monitorada e **c** o centro do espaço. A partir desta análise, é possível obter uma região hiperesférica no espaço d-dimensional que modela sinais sem distúrbios [10].

$$r = \|\mathbf{v} - \mathbf{c}\|^2 \quad (10)$$

Com objetivo de definir a região onde não há presença de falta, sinais nominais contendo fases de -180º a 180º e SNR (*signal-to-noise ratio*) de 60dB, são utilizados para determinar os valores de r para esta situação. A partir destes valores, os limites inferior e superior são adotados como

$$r_{max} = \bar{r} + 3 * \sigma \quad (11)$$

e

$$r_{min} = \bar{r} - 3 * \sigma \quad (12)$$

onde $\bar{r}$ é o valor médio e σ o desvio padrão de r.

Os limites encontrados durante a fase de projeto definem uma região que modela os sinais nominais e a qualquer evento fora desta é associado uma pertubação.

Caso o valor de r para um sinal seja maior que r_{max} ou menor que r_{min}, janelas com sessenta e quatro pontos pré-falta (um quarto de ciclo) e oito pontos pós-falta (um trinta e dois avos de ciclo) dos sinais de corrente e tensão são enviadas para a etapa de extração de características.

B. Extração de Características

O uso da Estatística de Ordem Superior para obter informações relevantes sobre sinais elétricos em sistemas de potência tem se mostrado uma boa alternativa para extração de características [15], [16]. Para um vetor de características finito, tais estatísticas podem ser aproximadas pelas Equações (7), (8) e (9).

Como o conjunto de características gerado por esta abordagem é muito grande, o Discriminante Linear de Fisher (DLF) foi utilizado para selecionar os cumulantes que melhor discriminam entre as classes. O DLF vem sendo aplicado em problemas de classificação para selecionar e reduzir os dados de entrada [17], [18], [19]. A Equação (13) apresenta a função de custo do Discriminante Linear de Fisher para um problema com *N* classes [20], onde μ_i, σ_i, μ_j e σ_j são a média e variância das classes *i* e *j* respectivamente.

$$\mathbf{J} = \sum_{\mathbf{i}}^{\mathbf{N}}\sum_{\mathbf{j}\neq\mathbf{i}}^{\mathbf{N}} \frac{(\mu_\mathbf{i} - \mu_\mathbf{j})^\mathbf{2}}{(\sigma_\mathbf{i}^\mathbf{2} + \sigma_\mathbf{j}^\mathbf{2})} \quad (13)$$

Uma vez que o Discriminante Linear de Fisher não remove redundância, outra técnica deve ser utilizada para este fim.

Neste trabalho, um filtro baseado no Coeficiente de Pearson foi implementado para limitar a correlação entre as características selecionadas. A Equação (14) apresenta o coeficiente de Pearson calculado para as características c_i e c_j com matrizes de autocorrelação e correlação cruzada iguais a σ_{ii}, σ_{jj} e σ_{ij}. Quanto mais próximo da unidade for o valor absoluto de R_{ij}, maior é a correlação as características.

$$\mathbf{R_{ij}} = \frac{\sigma_{ij}}{\sigma_{ii} * \sigma_{jj}} \tag{14}$$

C. Classificação

Nesta etapa, os cumulantes selecionados servem de base para o treinamento de uma Rede Neural Multicamadas. A arquitetura do classificador é apresentada na Tabela I. Foi adotada uma arquitetura simples para melhorar a capacidade de generalização do classificador [21]. O classificador conta com duas camadas. A camada escondida possui oito neurônios com função de ativação Tangente Hiperbólica Sigmoidal, descrita pela Equação (15). A camada de saída possui dez neurônios, representando as possíveis faltas, com função de ativação Softmax, descrita pela Equação (16).

Tabela I
ARQUITETURA DO CLASSIFICADOR.

Camada	Neurônios	Função de Ativação
Camada escondida	8	Tangente Hiperbólica
Camada de saída	10	Softmax

$$g_1(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{15}$$

$$g_2(x_i) = \frac{e^{-x_i}}{\sum_{j=1}^{10} e^{-x_j}} \tag{16}$$

Para treinamento da rede, aplicou-se o otimizador Adam, proposto por [22]. Os hiper parâmetros deste algoritmo foram mantidos nos valores sugeridos pelos autores do método. A taxa de aprendizado utilizada foi de 0.01.

IV. Base de Dados

O sistema de transmissão utilizado para simular os dados conta com 2 linhas de 300km cada, tensão de 500 kV e SNR (signal-noise-ratio) de 60dB. A frequência fundamental adotada foi de 60Hz e frequência de amostragem de 15360Hz, resultando em 256 pontos por ciclo. Foram simulados dez tipos de falta: AT, BT, CT, AB, AC, BC, ABT, ACT, BCT e ABT. A simulação foi feita por meio do software Simulink.

Ao todo, 950 sinais contendo falta foram gerados. A Tabela II descreve as 5 bases utilizadas na simulação. Em cada base foram gerados 19 sinais de cada tipo de falta com ângulos de incidência de 0 à 180º em passos de 10º.

Tabela II
PARÂMETROS UTILIZADOS PARA SIMULAR AS FALTAS.

Resistência (Ohms)	Distância à Barra Local(Km)
1	20
1	150
1	280
50	150
100	150

V. Resultados e Discussão

A. Detecção

Nesta etapa, os limiares inferior e superior encontrados para definir a região que modela os sinais nominais foram $r_{max} = 2.642 * 10^{13}$ e $r_{min} = 2.637 * 10^{13}$.

A Figura 1 mostra um exemplo dos sinais de tensão e corrente segmentados pelo detector na ocorrência de uma falta. A linha preta tracejada indica o ponto em que foi detectado a falta. A partir deste ponto, um quarto de ciclo pré-falta e um trinta e dois avos de ciclo pós-falta formam a janela a ser enviada para a etapa de extração de características. Todos os 950 segmentos contendo falta foram detectados.

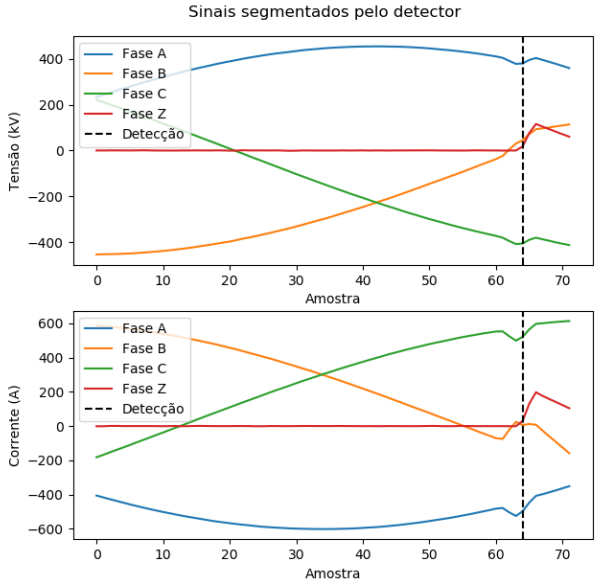


Figura 1. Sinais de tensão e corrente segmentados pelo detector.

O detector apresentou um atraso de detecção igual ao tempo necessário para que a deformação no sinal de tensão gere uma distância maior que os limiares estabelecidos. Ao ser aplicado no banco de dados, o atraso do instante de detecção em relação ao valor real foi de 4.13±0.95 amostras. Em termos de fração de ciclo, este valor representa um erro médio de sessenta e quatro avos com desvio padrão de um duzentos e cinquenta e seis avos de ciclo.

B. Extração de Características

Os cumulantes de segunda, terceira e quarta ordem foram extraídos das fases A, B, C e Z dos sinais de tensão e

corrente. Para cada cumulante foi gerado um conjunto de N características, onde N é o tamanho da janela enviada pelo detector. Devido à simetria dos cumulantes de segunda ordem, a primeira metade destes foi descartada. Foi ainda extraído o valor médio quadrático e o valor máximo absoluto dos sinais. Desta forma, foram obtidas 1456 características.

Para reduzir o número de características e assim a complexidade computacional do método, o Discriminante Linear de Fisher foi aplicado ao conjunto total de características. A Figura 2 mostra o resultado deste método. Cada ponto nesta figura representa o valor de importância dada pelo critério de Fisher a uma característica.

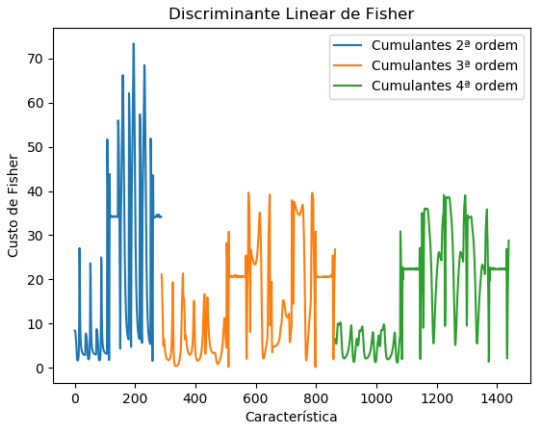


Figura 2. Função custo do Discriminante Linear de Fisher

A partir do DLF e do coeficiente de correlação de Pearson, foram selecionadas vinte e cinco características com correlação máxima de cinquenta porcento entre elas. A Figura 3 mostra a quantidade de características selecionadas de cada tipo.

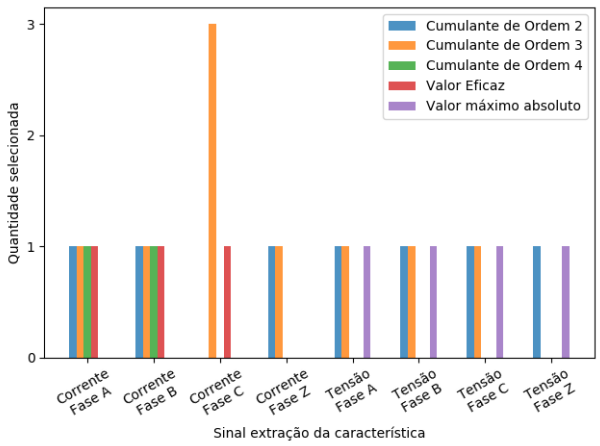


Figura 3. Sinal de origem, ordem e quantidade dos cumulantes selecionados.

Para verificar se os cumulantes trazem informação útil quanto à presença de falta, a distribuição amostral de um cumulante foi aproximada por uma gaussiana. A Figura 4 mostra o resultado. Este cumulante foi extraído da fase C do sinal de corrente. Nota-se que as distribuições das classes que contém a fase C estão situadas mais à esquerda, enquanto as demais à direita. Estas diferenças são interessantes no problema de classificação.

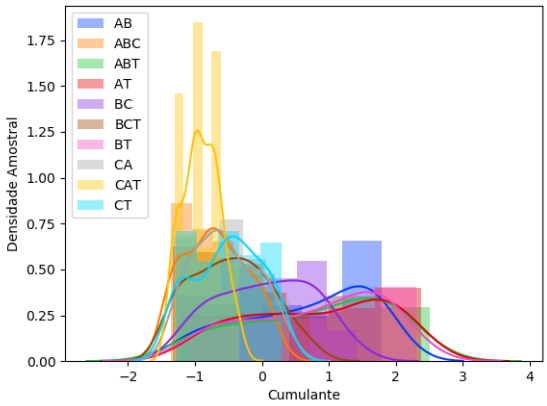


Figura 4. Densidade amostral de um cumulante extraído da fase C do sinal de corrente.

A Figura 5 mostra o espaço gerado pelas três melhores características selecionadas (C1, C2 e C3) para diferentes quantidades de dados pós falta. Com 1 ciclo pós falta, há uma separação linear entre as classes. Utilizando pelo menos ⅛ ciclo há uma boa separação. Nos demais casos, as sobreposições se tornam mais evidentes. Apesar disso, quando todas as 25 características foram utilizadas, o classificador apresentou bons resultados em todos os casos.

C. *Classificação*

Para validação do classificador foi utilizado a metodologia K-Fold com 10 divisões. A Figura 6 mostra a porcentagem de acerto para diferentes quantidades de pontos pós falta na janela segmentada pelo detector. Como não se sabia inicialmente quantos ciclos seriam necessários, este número foi diminuído gradativamente de 1 até 1/32 ciclos. Foi obtido 100% (0,00%) de acerto para ½ e ⅛ ciclo pós falta. 99,78% (0,43%) para 1, ¼ e 1/16 ciclo pós falta e 97,66% (1,15%) para 1/32 ciclo pós falta.

A partir destes resultados, foi escolhido utilizar 1/32 ciclos de dados após a detecção. Para este caso, a validação também foi feita por matriz de confusão. O conjunto de teste foi formado por 30% dos dados. O resultado é apresentado na Figura 7. Para as faltas monofásicas e as faltas CA e ABT o resultado da classificação foi de 100%. As demais faltas apresentaram porcentagens de acerto entre 96% e 97%.

VI. Conclusão

Este trabalho propôs um método de classificação baseado em Estatística de Ordem Superior que utiliza apenas 1/32 ciclos pós falta. Neste sentido, foi possível classificar 10 tipos de falta, incluindo as monofásicas, bifásicas e trifásicas. Para

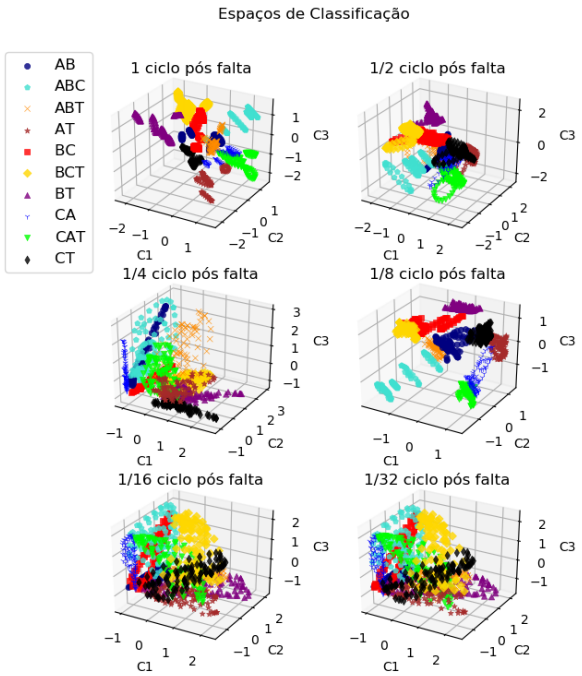


Figura 5. Espaços de classificação para diferentes quantidades de ciclos pós falta.

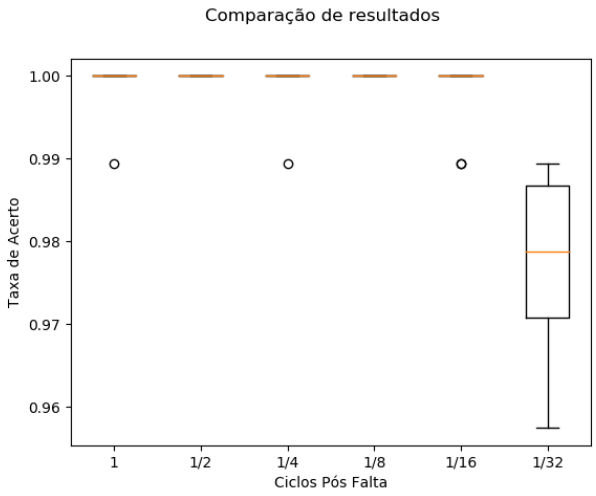


Figura 6. Variação da porcentagem de acerto com o número de ciclos pós falta utilizado.

reduzir a complexidade do classificador, um detector baseado na distancia euclidiana foi implementado.

Deve ser mencionado que os cumulantes de ordem três e quatro, utilizados como entrada para o classificador, são imunes a ruídos gaussianos, o que confere robustez ao método.

Em trabalhos futuros, pretende-se analisar a complexidade computacional do método e implementar um classificador que dê mais prioridade para as faltas monofásicas.

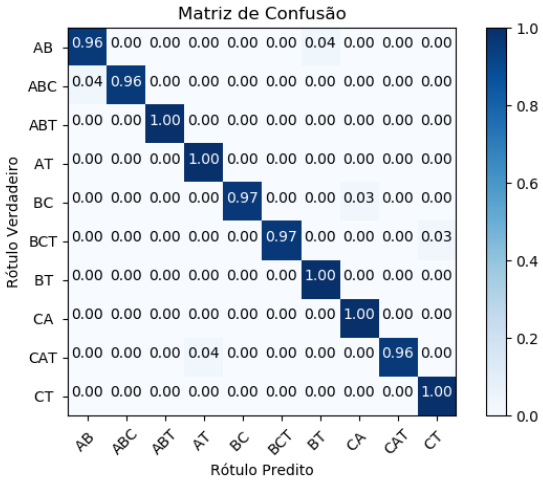


Figura 7. Matriz de confusão.

AGRADECIMENTOS

Os autores gostariam de agradecer à Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) pelo suporte.

REFERÊNCIAS

[1] J. Grainger and W. Stevenson, *Power System Analysis*, ser. Electrical engineering series. McGraw-Hill, 1994.

[2] K. Silva, B. A. Souza, and N. S. Brito, "Fault detection and classification in transmission lines based on wavelet transform and ann," *IEEE Transactions on Power Delivery*, vol. 21, no. 4, pp. 2058–2063, 2006.

[3] A. Jamehbozorg and S. M. Shahrtash, "A decision-tree-based method for fault classification in single-circuit transmission lines," *IEEE Transactions on Power Delivery*, vol. 25, no. 4, pp. 2190–2196, 2010.

[4] A. Jamehbozorg and S. Shahrtash, "A decision tree-based method for fault classification in double-circuit transmission lines," *IEEE Transactions on Power Delivery*, vol. 25, no. 4, pp. 2184–2189, 2010.

[5] S. Samantaray, P. Dash, and G. Panda, "Fault classification and location using hs-transform and radial basis function neural network," *Electric Power Systems Research*, vol. 76, no. 9-10, pp. 897–905, 2006.

[6] S. Samantaray and P. Dash, "Pattern recognition based digital relaying for advanced series compensated line," *International Journal of Electrical Power & Energy Systems*, vol. 30, no. 2, pp. 102–112, 2008.

[7] S. Samantaray, "A systematic fuzzy rule based approach for fault classification in transmission lines," *Applied soft computing*, vol. 13, no. 2, pp. 928–938, 2013.

[8] D. Thukaram, H. Khincha, and H. Vijaynarasimha, "Artificial neural network and support vector machine approach for locating faults in radial distribution systems," *IEEE Transactions on Power Delivery*, vol. 20, no. 2, pp. 710–721, 2005.

[9] L. Cheng, L. Wang, and F. Gao, "Power system fault classification method based on sparse representation and random dimensionality reduction projection," in *2015 IEEE Power & Energy Society General Meeting*. IEEE, 2015, pp. 1–5.

[10] E. G. Ribeiro, T. M. Mendes, G. L. Dias, E. R. Faria, F. M. Viana, B. H. Barbosa, and D. D. Ferreira, "Real-time system for automatic detection and classification of single and multiple power quality disturbances," *Measurement*, vol. 128, pp. 276–283, 2018.

[11] D. D. Ferreira, C. A. Marques, J. M. de Seixas, A. S. Cerqueira, M. V. Ribeiro, and C. A. Duque, "Exploiting higher-order statistics information for power quality monitoring," *Power Quality: Intech Open Access Publisher*, pp. 345–362, 2011.

[12] M. V. Ribeiro, C. A. G. Marques, C. A. Duque, A. S. Cerqueira, and J. L. R. Pereira, "Detection of disturbances in voltage signals for power quality analysis using hos," *EURASIP Journal on Applied Signal Processing*, vol. 2007, no. 1, pp. 177–177, 2007.

[13] J. M. Mendel, "Tutorial on higher-order statistics (spectra) in signal processing and system theory: Theoretical results and some applications," *Proceedings of the IEEE*, vol. 79, no. 3, pp. 278–305, 1991.

[14] H. Kushner and G. G. Yin, *Stochastic approximation and recursive algorithms and applications*. Springer Science & Business Media, 2003, vol. 35.

[15] J. J. G. de la Rosa and A. Moreno-Munoz, "Higher-order characterization of power quality transients and their classification using competitive layers," in *2009 Compatibility and Power Electronics*. IEEE, 2009, pp. 390–395.

[16] C. L. Nikias and J. M. Mendel, "Signal processing with higher-order spectra," *IEEE Signal processing magazine*, vol. 10, no. 3, pp. 10–37, 1993.

[17] T. S. Barbosa, D. D. Ferreira, D. A. Pereira, R. R. Magalhães, and B. H. Barbosa, "Fault detection and classification in cantilever beams through vibration signal analysis and higher-order statistics," *Journal of Control, Automation and Electrical Systems*, vol. 27, no. 5, pp. 535–541, 2016.

[18] R. Naves, B. H. Barbosa, and D. D. Ferreira, "Classification of lung sounds using higher-order statistics: A divide-and-conquer approach," *Computer methods and programs in biomedicine*, vol. 129, pp. 12–20, 2016.

[19] J. D. S. Guedes, D. D. Ferreira, B. H. G. Barbosa, C. A. Duque, and A. S. Cerqueira, "Non-intrusive appliance load identification based on higher-order statistics," *IEEE Latin America Transactions*, vol. 13, no. 10, pp. 3343–3349, 2015.

[20] S. Theodoridis, A. Pikrakis, K. Koutroumbas, and D. Cavouras, *Introduction to pattern recognition: a matlab approach*. Academic Press, 2010.

[21] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

APÊNDICE B – Classe para Extração de Características

```
import numpy as np

class FeatureExtractionAIA:
    def __init__(self):
        pass

    def cum(self, vetEntrada, nEvents, order):

        def cum2Calc(vetMediaZero, nPoints, ii):
            return np.matmul(vetMediaZero.T, np.roll(
                vetMediaZero, ii))/nPoints

        def cum3Calc(vetMediaZero, nPoints, ii):
            return np.matmul(vetMediaZero.T, np.roll(
                vetMediaZero, ii)**2)/nPoints

        def cum4Calc(vetMediaZero, nPoints, ii):
            sumOfSquares = np.dot(vetMediaZero.T,
                vetMediaZero)
            part1 = np.matmul(vetMediaZero.T, np.roll(
                 vetMediaZero, ii)**3)/nPoints
            part2 = 3*np.matmul(vetMediaZero.T, np.roll(
                vetMediaZero,
                                                        ii)
                                                        )
                                                        *
                                                        sumOfSquares
                                                        /(
                                                        nPoints
                                                        **2)
```

```
    cum4 = part1 - part2
    return cum4

if (order == 2):
    functionCalled = cum2Calc
elif (order == 3):
    functionCalled = cum3Calc
elif (order == 4):
    functionCalled = cum4Calc
else:
    print("The order must be in [2,4]")
    return

# Transformando vetEntrada em um vetor coluna:
dimVet = vetEntrada.shape
if(dimVet[0] == nEvents):
    vetEntrada = vetEntrada.T

nPoints = vetEntrada[:, 0].size  # number of points
    per column

# Pre - allocating space
cum = np.zeros([nEvents, nPoints])

for i in range(nEvents):
    # Transformando vetEntrada em um vetor de media
        nula:
    media = np.mean(vetEntrada[:, i])
    vetMediaZero = vetEntrada[:, i] - media

    for ii in range(nPoints):
```

```
            cum[i, ii] = functionCalled(vetMediaZero,
                nPoints, ii)

    return cum
```

APÊNDICE C – Classe para Seleção de Características

```
import numpy as np
from . import HelpersAIA


class FeatureSelectionAIA:
    def __init__(self):
        self.helper = HelpersAIA()

    def FDR(self, C, targets, nClasses):
        _, integer_encoded, _ = self.helper.getEncodeds(
            targets)
        classes = []
        for i in range(nClasses):
            classes.append(C[integer_encoded == i])

        classes = np.array(classes)

        m = np.zeros((nClasses, C.shape[1]))
        var = np.zeros((nClasses,  C.shape[1]))
        for i in range(nClasses):
            m[i] = np.mean(classes[i], axis=0)
            var[i] = np.var(classes[i], axis=0)

        J = np.zeros((nClasses, C.shape[1]))
        for i in range(nClasses):
            for j in range(nClasses):
                if(i != j):
                    J[i] += np.divide(np.power(m[i] - m[j],
                        2),
                                      var[i] + var[j])
```

```
        J = np.sum(J, axis=0)
        I = np.argsort(J, axis=0)
        I = I[::-1]
        return I, J

    def removeRedundancy(self, C, I, maxCor, nDesired):
        print('Selecionando ' + str(nDesired) +
              ' caracteristicas com no maximo ' + str(
                 maxCor*100) + '% de correlacao')

        # the best characteristic is always added to the
           response
        index1 = I[0]
        lastSelected = 0
        selected = []
        selected.append(index1)
        for j in range(nDesired - 1):
            found = False
            index = []
            for i, caract in enumerate(C[:, I[lastSelected
                :]].T):
                cond = True
                for k in range(len(selected)):
                    cor = np.corrcoef(C[:, selected[k]],
                       caract)[0, 1]
                    if(abs(cor) > maxCor):
                        cond = False
                        break
                if(cond):
                    index = I[lastSelected + i]
                    found = True
                    lastSelected += i
```

```
                break

    if(not(found)):
        print("So foi possivel encontrar " + str(j
            +1) +
               " caracteristicas com maxima
                  correlacao especificada")
        print('Saindo da funcao.')
        return selected
    else:
        selected.append(index)
return selected
```

APÊNDICE D – Classe para Classificação

```
import numpy as np
import multiprocessing
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from . import HelpersAIA


class ClassificationAIA:

    def __init__(self):
        self.helper = HelpersAIA()

    def crossValidation(self, X, targets, estimator,
        nSplits=10):

        _, _, onehot_encoded = self.helper.getEncodeds(
            targets)
        seed = 7
        np.random.seed(seed)
        kfold = KFold(n_splits=nSplits, shuffle=True,
                      random_state=seed)
        cv_results = cross_val_score(
            estimator, X, onehot_encoded, cv=kfold, verbose
               =2, n_jobs=-1)
        print("Baseline: %.2f%% (%.2f%%)" %
              (cv_results.mean()*100, cv_results.std()*100)
                 )
        return cv_results
```

```
    def train(self, X, targets, model, epochs, batchSize,
        testSize=0.3, verbose=0):
        _, _, onehot_encoded = self.helper.getEncodeds(
            targets)

        X_train, X_test, y_train, y_test = train_test_split
            (
            X, onehot_encoded, test_size=testSize)
        # Fit the model
        history = model.fit(X_train, y_train,
            validation_data=(
            X_test, y_test), epochs=epochs, batch_size=
                batchSize, verbose=verbose)

        # Prediction
        y_pred = model.predict(X_test)
        y_pred = (y_pred > 0.5)

        # Confusion Matrix
        y_test_non_category = [np.argmax(t) for t in y_test
            ]
        y_predict_non_category = [np.argmax(t) for t in
            y_pred]

        from sklearn.metrics import confusion_matrix
        conf_mat = confusion_matrix(
            y_test_non_category, y_predict_non_category)

        return history, conf_mat
    # summarize history for accuracy
```

```
def train2(self, X_train, X_test, y_train, y_test,
    model, epochs, batchSize, verbose=0):
    # Fit the model
    history = model.fit(X_train, y_train, epochs=epochs
        ,
                        batch_size=batchSize, verbose=
                            verbose)

    # Prediction
    y_pred = model.predict(X_test)
    y_pred = (y_pred > 0.5)

    # Confusion Matrix
    y_test_non_category = [np.argmax(t) for t in y_test
        ]
    y_predict_non_category = [np.argmax(t) for t in
        y_pred]

    from sklearn.metrics import confusion_matrix
    conf_mat = confusion_matrix(
        y_test_non_category, y_predict_non_category)

    return history, conf_mat
# summarize history for accuracy
```

APÊNDICE E – Classe para Detecção

```
import numpy as np


class DetectorMendes:

    def __init__(self, voltage, nPointsPerCycle):
        eDist = np.zeros((voltage.shape[0]))
        for l in range(voltage.shape[0]):
            cycle = voltage[l: nPointsPerCycle + l]
            eDist[l] = np.sum((cycle)**2)
        self.nPointsPerCycle = nPointsPerCycle
        self.threshold = np.mean(eDist)
        self.standard_dev = 3*np.std(eDist)

    def detect(self, voltage):
        index = np.array([-1, -1, -1])
        for i in range(3):
            for k in range(voltage.shape[0] - self.
                nPointsPerCycle):
                d = np.sum((voltage[k: k + self.
                    nPointsPerCycle, i])**2)
                if (d > (self.threshold + self.standard_dev
                    ) or d < (self.threshold - self.
                    standard_dev)):
                    index[i] = k + self.nPointsPerCycle
                    break

        positiveIndexesMask = index >= 0
        positiveIndexes = index[positiveIndexesMask]
        if(len(positiveIndexes) != 0):
            print("Disturbio detectado!")
```

```
            return min(positiveIndexes)
        else:
            print("Nao ha disturbios.")
            return -1
```

APÊNDICE F – Classe Auxiliar

```
import os
import errno
import itertools
import pathlib
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder


class HelpersAIA:
    def __init__(self):
        pass

    def getEncodeds(self, targets):
        label_encoder = LabelEncoder()
        label_encoder.fit(targets)
        integer_encoded = label_encoder.fit_transform(
            targets)

        onehot_encoder = OneHotEncoder(sparse=False)
        onehot_encoded = onehot_encoder.fit_transform(
            integer_encoded.reshape(len(integer_encoded),
                1))

        return label_encoder, integer_encoded,
            onehot_encoded

    def plotConfusionMatrix(self, cm, classes,
```

```
                      normalize=True,
                      title='Confusion matrix',
                      cmap=plt.cm.Blues):
"""
This function prints and plots the confusion matrix
    .
Normalization can be applied by setting 'normalize=
    True'.
"""
if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.
        newaxis]
    print("Normalized confusion matrix")
else:
    print('Confusion matrix, without normalization'
        )

print(cm)

plt.imshow(cm, interpolation='nearest', cmap=cmap)
plt.title(title)
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]),
    range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
```

```
                color="white" if cm[i, j] > thresh
                    else "black")

    plt.ylabel('Rotulo Verdadeiro')
    plt.xlabel('Rotulo Predito')
    plt.tight_layout()

def scatterPlot(self, X, targets, lw=2, alpha=.8):

    label_encoder, integer_encoded, _ = self.
        getEncodeds(targets)
    target_names = label_encoder.classes_
    colors = ['navy', 'turquoise', 'darkorange', 'brown
        ',
              'red', 'gold', 'purple', 'blue', 'lime',
                 'black']
    markers = ['o', 'p', 'x', '*', 's', 'D', '^', '1',
        'v', 'd']

    fig = plt.figure()
    if(X.shape[1] == 2):
        for color, i, target_name, marker in zip(colors
            , len(target_names), target_names, markers):
            plt.scatter(X[integer_encoded == i, 0], X[
                integer_encoded == i, 1], color=color,
                alpha=alpha, lw=lw,
                         label=target_name, marker=
                            marker)
            plt.legend(loc='best', shadow=False,
                scatterpoints=1)
            return fig
```

```
elif(X.shape[1] == 3):
    ax = fig.add_subplot(111, projection='3d')
    for color, i, target_name, marker in zip(colors
        , len(target_names), target_names, markers):
        ax.scatter(X[integer_encoded == i, 0], X[
            integer_encoded == i, 1], X[
            integer_encoded == i, 2], color=color,
            alpha=alpha, lw=lw,
                    label=target_name, marker=marker
                        )
    plt.legend(loc='best', shadow=True,
        scatterpoints=1)
    return fig
else:
    print("X.shape[1] must be two or three")
```

APÊNDICE G – Rotina para Extração de Características

```
import numpy as np
import matplotlib.pyplot as plt
import time
import scipy.io as sio
import os
from os import listdir
from os.path import dirname, join as pjoin
import pathlib
from fractions import Fraction
from ..library import FeatureExtractionAIA
from .helpers import functions


def selectDetectionFiles(data_dir):
    AllfileNames = np.array((os.listdir(data_dir)))
    return np.array(list(filter(lambda x: x.startswith('
        detected'), AllfileNames)))


def cums_calc(signal, nEvents):
    cum_2 = ft_extractor.cum(signal, nEvents, 2)[:, :
        signal.shape[1]//2]
    cum_3 = ft_extractor.cum(signal, nEvents, 3)
    cum_4 = ft_extractor.cum(signal, nEvents, 4)
    return cum_2, cum_3, cum_4


plot = False
initialNumberOfCycles = 1
lastNumberOfCycles = 1
nPointsPerCycle = 256
```

```
nonFaultyPointsAtBegining = 64
initialPointsToDiscard = 64 - nonFaultyPointsAtBegining
nPhases = 4  # A, B, C and Zero

data_dir = pjoin(pathlib.Path(__file__).parent.parent /
                 'data' / 'detected-signals')

save_dir = pathlib.Path(__file__).parent.parent / \
    'data' / ('extracted-cumulants')

functions.createFolder(save_dir)
print('Configuracao inicial')
print('----- Pontos por ciclo: ' + str((nPointsPerCycle)))
print('----- Repetindo proceso de ' + str(Fraction(
    initialNumberOfCycles)) + ' ciclo ate ' +
      str(Fraction(lastNumberOfCycles)) + ' ciclo')

fileNames = selectDetectionFiles(data_dir)
nEvents = fileNames.size
mat = np.array([sio.loadmat(pjoin(data_dir, name)) for name
    in fileNames])
voltages = [voltage['V'] for voltage in mat]
currents = [currents['I'] for currents in mat]
ft_extractor = FeatureExtractionAIA()
if __name__ == '__main__':
    for nCycles in functions.getNumberOfCyclesGen(
            initialNumberOfCycles, lastNumberOfCycles):
        timeStart = time.time()
        print('Calculando cumulantes com ' +
              str(Fraction(nCycles)) + ' ciclos...')
        nPoints = int(nCycles*nPointsPerCycle) + \
```

```
    nonFaultyPointsAtBegining +
       initialPointsToDiscard
print('nPoints: ' + str(nPoints -
   initialPointsToDiscard))
if plot:
    functions.plot_signal(nonFaultyPointsAtBegining
       ,
                          np.array(voltages[0][
                              initialPointsToDiscard
                              :nPoints, :3]))
cumulants = dict()
voltagesA = np.array(
    [voltage[initialPointsToDiscard:nPoints, 0] for
        voltage in voltages[:]])
voltagesB = np.array(
    [voltage[initialPointsToDiscard:nPoints, 1] for
        voltage in voltages[:]])
voltagesC = np.array(
    [voltage[initialPointsToDiscard:nPoints, 2] for
        voltage in voltages[:]])
voltagesZ = np.array(
    [voltage[initialPointsToDiscard:nPoints, 3] for
        voltage in voltages[:]])
currentsA = np.array(
    [current[initialPointsToDiscard:nPoints, 0] for
        current in currents[:]])
currentsB = np.array(
    [current[initialPointsToDiscard:nPoints, 1] for
        current in currents[:]])
currentsC = np.array(
    [current[initialPointsToDiscard:nPoints, 2] for
        current in currents[:]])
```

```
currentsZ = np.array(
    [current[initialPointsToDiscard:nPoints, 3] for
        current in currents[:]])

signals = [voltagesA, voltagesB, voltagesC,
   voltagesZ, currentsA, currentsB,
           currentsC, currentsZ]

print('----- Tipos de sinal:', len(signals))
_max = np.zeros((voltagesA.shape[0], len(signals)))
_rms = np.zeros((voltagesA.shape[0], len(signals)))
for i, signal in enumerate(signals):
    print('---------- Iteracao', i+1, 'de', len(
       signals))
    _max[:, i] = np.array(
        [np.amax(np.absolute(s[
           nonFaultyPointsAtBegining:nPoints])) for
            s in signal])
    _rms[:, i] = np.array(
        [np.sqrt(np.mean(s[
           nonFaultyPointsAtBegining:nPoints]**2))
           for s in signal])
    if(i == 0):
        cum2, cum3, cum4 = cums_calc(signal,
           nEvents)
        continue
    aux_cum2, aux_cum3, aux_cum4 = cums_calc(signal
       , nEvents)
    cum2 = np.column_stack((cum2, aux_cum2))
    cum3 = np.column_stack((cum3, aux_cum3))
    cum4 = np.column_stack((cum4, aux_cum4))
C = np.column_stack((cum2, cum3))
```

```
C = np.column_stack((C, cum4))
C = np.column_stack((C, _max))
C = np.column_stack((C, _rms))

cumulants['C'] = C

cumulants['labels'] = np.array([data['faultType'
    ][0] for data in mat])

timeElapsed = time.time() - timeStart
print("Tempo decorrido: " + str(timeElapsed))

nCyclesAsFraction = Fraction(nCycles)
saveName = "cums" + str(nCyclesAsFraction.numerator
    ) + \
     '-' + str(nCyclesAsFraction.denominator)+"
         Ciclos"
sio.savemat(
     pjoin(save_dir, saveName), cumulants)
print('Salvando resultados!')
```

APÊNDICE H – Rotina para Seleção de Características

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import scipy.io as sio
import itertools
from fractions import Fraction
import multiprocessing
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from os.path import dirname, join as pjoin
import pathlib
import os
import errno
from .helpers import functions
from ..library import FeatureSelectionAIA
from sklearn.preprocessing import StandardScaler
from matplotlib.ticker import MaxNLocator


def getPhaseAndOrder(index, nPointsPerCumPerPhase):
    # print(nPointsPerCumPerPhase)
    label = ""
    nPhases = 4
    nCums = 2.5  # not 3 becuase half of cum2 is thrown
       away
    nSignals = 2  # current and voltage
    totalPoints = (nSignals*nCums*nPhases*
       nPointsPerCumPerPhase)
    acharCum = int(index/(totalPoints/5))
    acharFase = []
```

```
if(acharCum in [0]):
    label += "c2"
    acharFase = int(index / (nPointsPerCumPerPhase/2))
elif (acharCum in [1, 2]):
    label += "c3"
    acharFase = (index - nSignals*nPhases*0.5 *
                 nPointsPerCumPerPhase) // (
                    nPointsPerCumPerPhase)
elif (acharCum in [3, 4]):
    label += "c4"
    acharFase = (index - nSignals*nPhases*1.5 *
                 nPointsPerCumPerPhase) // (
                    nPointsPerCumPerPhase)
else:
    indexWithoutOffset = (index - nSignals*nPhases*2.5
       *
                          nPointsPerCumPerPhase)
    acharMaxRms = indexWithoutOffset // (nSignals*
       nPhases)
    acharFase = indexWithoutOffset - acharMaxRms*
       nSignals*nPhases
    if(acharMaxRms == 0):
        label += 'max'
    elif(acharMaxRms == 1):
        label += 'rms'

if(acharFase == 0):
    label += "vA"
elif (acharFase == 1):
    label += "vB"
elif (acharFase == 2):
    label += "vC"
```

```
    elif (acharFase == 3):
        label += "vZ"
    elif(acharFase == 4):
        label += "iA"
    elif (acharFase == 5):
        label += "iB"
    elif (acharFase == 6):
        label += "iC"
    elif (acharFase == 7):
        label += "iZ"

    print('Index:', index, 'Label:', label)
    return label


def featureSelection(loadName):
    dataSet = sio.loadmat(pjoin(data_dir, loadName))
    saveFlag = loadName[4:]  # Ex: 1-16Ciclos
    C = dataSet['C']
    print(C.shape)
    # C_norm = np.zeros((C.shape))
    # for i in range(C.shape[1]):
    #     max_ = np.max(C[:, i])
    #     min_ = np.min(C[:, i])
    #     C_norm[:, i] = (C[:, i] - min_)/(max_ - min_)

    scaler = StandardScaler()
    C_norm = scaler.fit_transform(C)

    targets = dataSet['labels']
    # 2.5 porque uma metade do cum2 foi descartada
    nPointsPerCumPerPhase = (C.shape[1]/2.5)//(2*4)
```

```
label_encoder = LabelEncoder()
label_encoder.fit(targets)
integer_encoded = label_encoder.fit_transform(targets)
#-------------------------------- FISHER MULTI CLASS
   ------------------------------------#
nClasses = 10

I, J = ft_selection.FDR(C_norm, integer_encoded,
   nClasses)
indexes = ft_selection.removeRedundancy(C_norm, I, 0.5,
    25)

if(plot):
    target_names = label_encoder.classes_
    colors = ['navy', 'turquoise', 'darkorange', 'brown
       ',
              'red', 'gold', 'purple', 'blue', 'lime',
                 'black']
    markers = ['o', 'p', 'x', '*', 's', 'D', '^', '1',
       'v', 'd']
    lw = 2

    plt.figure()
    for color, i, target_name, marker in zip(colors,
       range(10), target_names, markers):
        plt.scatter(C_norm[integer_encoded == i,
           indexes[0]], C_norm[integer_encoded == i,
           indexes[1]], color=color, alpha=.8, lw=lw,
                   label=target_name, marker=marker)
    plt.legend(loc='best', shadow=False, scatterpoints
       =1)
    plt.title('Fault Classification')
```

```
plt.xlabel(
    "C1 [" + getPhaseAndOrder(indexes[0],
       nPointsPerCumPerPhase) + "]")
plt.ylabel(
    "C2 [" + getPhaseAndOrder(indexes[1],
       nPointsPerCumPerPhase) + "]")
savePath = pjoin(figuresDir + '/2d-feature-space/')
functions.createFolder(savePath)
plt.savefig(pjoin(savePath, '2d' + saveFlag))

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
i2plot = [indexes[2], indexes[9], indexes[8]]
for color, i, target_name, marker in zip(colors,
   range(10), target_names, markers):
    ax.scatter(C_norm[integer_encoded == i, i2plot
       [0]], C_norm[integer_encoded == i, i2plot
       [1]], C_norm[integer_encoded == i, i2plot
       [2]], color=color, alpha=.8, lw=lw,
               label=target_name, marker=marker)
plt.legend(loc='upper left', shadow=True,
   scatterpoints=1)
plt.title('Fault Classification')
ax.set_xlabel(
    "C1 [" + getPhaseAndOrder(i2plot[0],
       nPointsPerCumPerPhase) + "]")
ax.set_ylabel(
    "C2 [" + getPhaseAndOrder(i2plot[1],
       nPointsPerCumPerPhase) + "]")
ax.set_zlabel(
    "C3 [" + getPhaseAndOrder(i2plot[2],
       nPointsPerCumPerPhase) + "]")
```

```
        savePath = pjoin(figuresDir + '/3d-feature-space/')
        functions.createFolder(savePath)
        plt.savefig(pjoin(savePath, '3d' + saveFlag))

        plt.figure()
        J_cum = J[:-16]
        # J_cum = J
        nPerCum = int(J_cum.shape[0]/5)
        plt.plot(range(nPerCum), J_cum[:nPerCum], label='
            Cumulantes 2a ordem')
        plt.plot(range(nPerCum, 3*nPerCum),
                 J_cum[nPerCum:3*nPerCum], label='
                    Cumulantes 3a ordem')
        plt.plot(range(3*nPerCum, 5*nPerCum),
                 J_cum[3*nPerCum:], label='Cumulantes 4a
                    ordem')
        plt.title('Discriminante Linear de Fisher')
        plt.xlabel("Caracteristica")
        plt.ylabel("Custo de Fisher")
        plt.legend()
        savePath = pjoin(figuresDir + '/cost/')
        functions.createFolder(savePath)
        plt.savefig(pjoin(savePath, 'custo' + saveFlag))

    print('-----')
    cum_selected = {
        "C": C_norm[:, indexes],
        'indexes': indexes,
        "targets": targets,
        'orders': [getPhaseAndOrder(index,
            nPointsPerCumPerPhase) for index in indexes]
    }
```

```
# print(indexes)
functions.createFolder(save_dir)
sio.savemat(pjoin(save_dir, 'selected' + saveFlag),
    cum_selected)

x = cum_selected['orders']
print(x)
possible_origin = ['Corrente\nFase A', 'Corrente\nFase
    B', 'Corrente\nFase C', 'Corrente\nFase Z',
                   'Tensao\nFase A', 'Tensao\nFase B',
                       'Tensao\nFase C', 'Tensao\nFase Z
                       ']
possible_origin_c2 = ['c2iA', 'c2iB', 'c2iC', 'c2iZ',
                      'c2vA', 'c2vB', 'c2vC', 'c2vZ']
possible_origin_c3 = ['c3iA', 'c3iB', 'c3iC', 'c3iZ',
                      'c3vA', 'c3vB', 'c3vC', 'c3vZ']
possible_origin_c4 = ['c4iA', 'c4iB', 'c4iC', 'c4iZ',
                      'c4vA', 'c4vB', 'c4vC', 'c4vZ']
possible_origin_max = ['maxvA', 'maxvB', 'maxvC', '
    maxvZ',
                       'maxiA', 'maxiB', 'maxiC', '
                           maxiZ']
possible_origin_rms = ['rmsvA', 'rmsvB', 'rmsvC', '
    rmsvZ',
                       'rmsiA', 'rmsiB', 'rmsiC', '
                           rmsiZ']

fig, ax = plt.subplots()
bar_width = 0.11
opacity = 0.8
index = np.arange(len(possible_origin))
```

```
    ax.bar(index, [x.count(origin) for origin in
       possible_origin_c2], bar_width,
           alpha=opacity, label='Cumulante de Ordem 2')
    ax.bar(index+bar_width, [x.count(origin) for origin in
       possible_origin_c3],
           bar_width, alpha=opacity, label='Cumulante de
              Ordem 3')
    ax.bar(index+2*bar_width, [x.count(origin) for origin
       in possible_origin_c4],
           bar_width, alpha=opacity, label='Cumulante de
              Ordem 4')
    ax.bar(index+3*bar_width, [x.count(origin) for origin
       in possible_origin_rms],
           bar_width, alpha=opacity, label='Valor Eficaz')
    ax.bar(index+4*bar_width, [x.count(origin) for origin
       in possible_origin_max],
           bar_width, alpha=opacity, label='Valor maximo
              absoluto')
    ax.yaxis.set_major_locator(MaxNLocator(integer=True))
    plt.ylabel('Quantidade selecionada')
    plt.xlabel('Sinal extracao da caracteristica')
    plt.xticks(index + 2*bar_width, possible_origin,
       rotation=30)
    plt.legend()
    plt.tight_layout()
    savePath = pjoin(figuresDir, 'bar-chart-cumulants')
    functions.createFolder(savePath)
    plt.savefig(pjoin(savePath, 'bar-chart'))


ft_selection = FeatureSelectionAIA()
nonFaultyPointsAtBegining = 64
```

```
data_dir = pathlib.Path(__file__).parent.parent / \
    'data' / ('extracted-cumulants')
save_dir = pathlib.Path(__file__).parent.parent / 'data' /
   \
    ('selected-cumulants')
figuresDir = pjoin(pathlib.Path(__file__).parent.parent / '
   figures')
plot = True
maxCor = 0.7
if __name__ == '__main__':
    initialNumberOfCycles = 1/32
    lastNumberOfCycles = 1/32
    loadNames = functions.loadNamesGen(
        'cums', initialNumberOfCycles, lastNumberOfCycles)
    jobs = []
    while(True):
        try:
            p = multiprocessing.Process(
                target=featureSelection, args=(next(
                    loadNames), ))
            p.start()
            jobs.append(p)
        except Exception as e:
            print(e)
            for proc in jobs:
                proc.join()
            break
```

APÊNDICE I – Rotina para Validação Cruzada

```
import pathlib
from os.path import dirname, join as pjoin
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
from keras.wrappers.scikit_learn import KerasClassifier
from ..library import HelpersAIA as hp
from ..library import ClassificationAIA
from .helpers import functions
from package import pickableFunctions

results = []
figuresDir = pjoin(pathlib.Path(__file__).parent.parent / '
    figures')
aia = ClassificationAIA()

for loadName in functions.loadNamesGen('selected', 1, 1/32)
    :
    saveFlag = loadName[8:]
    dataSet = sio.loadmat(
        pjoin(pathlib.Path(dirname(__file__)).parent / '
            data' / ('selected-cumulants'), loadName))
    targets = dataSet['targets']
    X = dataSet['C']

    # Estimator
    estimator = KerasClassifier(
        build_fn=pickableFunctions.baseline_model,
            _input_dim=X.shape[1], epochs=500, batch_size
            =50, verbose=0)
```

```
    results.append(aia.crossValidation(
        X, targets, estimator,  nSplits=10))

# boxplot algorithm comparison
names = ["1", "1/2", "1/4", "1/8", "1/16", "1/32"]
# names = ["1/16", "1/32"]
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
plt.ylabel("Accuracy")
plt.xlabel("Number of cycles used")
ax.set_xticklabels(names)
savePath = pjoin(figuresDir + '/cross-validation-results')
# functions.createFolder(savePath)
plt.savefig(pjoin(savePath, 'cross-val-scores'))
plt.show()

resultsDict = {
    "results": results
}
savePath = pjoin(pathlib.Path(__file__).parent.parent / '
   data' /
                'cross-validation-results')
functions.createFolder(savePath)
sio.savemat(pjoin(savePath, 'cross-val-scores'),
   resultsDict)
```

APÊNDICE J – Rotina para Gerar Matriz de Confusão

```
from sklearn.model_selection import KFold
import pathlib
from os.path import dirname, join as pjoin
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
from keras.wrappers.scikit_learn import KerasClassifier
from ..library import HelpersAIA as hp
from ..library import ClassificationAIA
from .helpers import functions
from package import pickableFunctions

results = []
figuresDir = pjoin(pathlib.Path(__file__).parent.parent / '
    figures')
aia = ClassificationAIA()

for loadName in functions.loadNamesGen('selected', 1/32,
    1/32):
     saveFlag = loadName[8:]
     dataSet = sio.loadmat(
         pjoin(pathlib.Path(dirname(__file__)).parent / '
             data' / ('selected-cumulants'), loadName))
     targets = dataSet['targets']
     X = dataSet['C']
     helper = hp()
     label_encoder, integer_encoded, onehot_encoded = helper
         .getEncodeds(
         targets)
     print(label_encoder.classes_)
     model = pickableFunctions.baseline_model(X.shape[1])
```

```
history, cm = aia.train(X=X, targets=targets, model=
    model,
                        epochs=500, batchSize=50,
                            testSize=0.3, verbose=1)

plt.figure()
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='best')

# summarize history for loss
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='best')

plt.figure()
helper.plotConfusionMatrix(cm,  classes=label_encoder.
    classes_,
                           title='Matriz de Confusao')
savePath = pjoin(figuresDir + '/confusion-matrix')
plt.savefig(pjoin(savePath, 'c_matrix'))
```