Universidade Federal do Pará

Centro de Ciências Exatas e Naturais

Programa de Pós-Graduação em Ciência da Computação

Jefferson Magalhães de Morais

Estudo de Algoritmos para Classificação de Séries Temporais: Uma Aplicação em Qualidade de Energia Elétrica

Belém

2007

Universidade Federal do Pará

Centro de Ciências Exatas e Naturais

Programa de Pós-Graduação em Ciência da Computação

Jefferson Magalhães de Morais

Estudo de Algoritmos para Classificação de Séries Temporais: Uma Aplicação em Qualidade de Energia Elétrica

Dissertação apresentada para obtenção do grau de Mestre em Ciência da Computação, elaborada sob orientação do Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior

Belém
2007

Universidade Federal do Pará

Centro de Ciências Exatas e Naturais

Programa de Pós-Graduação em Ciência da Computação

Jefferson Magalhães de Morais

Estudo de Algoritmos para Classificação de Séries Temporais: Uma Aplicação em Qualidade de Energia Elétrica

Dissertação apresentada para obtenção do grau
de Mestre em Ciência da Computação

Data da defesa: 24 de agosto de 2007

Conceito:

Banca examinadora

Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior (UFPA) - Orientador

---

Prof. Dr. Allan Kardec Barros (UFMA) - Membro Externo

Prof. Dr. Eloi Luiz Favero (UFPA) - Membro

Prof. Dr. Roberto Célio Limão de Oliveira (UFPA) - Membro

Visto:

Prof. Dr. Eloi Luiz Favero
Coordenador do PPGCC-UFPA

Aos meus pais

Carlos e Maria

e minha esposa

Rosa,

com amor e carinho...

Agradecimentos

Primeiramente à Deus por me conceder saúde, a graça de poder conviver com pessoas de excelente caráter, pelas conquistas alcançadas e pelo que sou hoje.

Ao meu orientador, Aldebaro, apresento meus sinceros agradecimentos pela dedicação, paciência e decisiva contribuição profissional nas discussões técnicas que nortearam o desenvolvimento deste trabalho e, sobretudo, por ter acreditado no meu potencial.

Aos meus pais, Carlos e Maria, por cuidarem da minha saúde, oferecer a oportunidade de estudar e incentivar o meu crescimento profissional e pessoal, sempre dando carinho e forças para superar as adversidades.

Agradeço à minha esposa e eterna namorada, Rosa, por estar presente na minha vida nos momentos mais difíceis e mais alegres, me confortando com uma palavra de carinho, um sorriso ou até mesmo um simples afago.

Aos meus familiares que me apoiram de forma direta ou indireta no desenvolvimento deste trabalho.

Aos amigos e companheiros do Laboratório de Processamento de Sinais (LaPS) Yomara, José, Claudomir, Gustavo, Igor, Márcio, Patrick, Adalbery, Tales, Rosal, Fabiola e todos os outros sem os quais não seria possível a realização deste.

Ao Grupo de Energia e Sistemas de Instrumentações (GSEI) da Universidade Federal do Pará pelo intercâmbio multidisciplinar.

Ao Programa de Pós-Graduação em Ciênica da Computação (PPGCC) da Universidade Federal do Pará por possibilitar o desenvolvimento deste trabalho. Agradeço ainda aos professores e funcionários desse programa e do PPGEE que tomaram parte deste trabalho.

À Coordenadoria de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo suporte financeiro durante os anos de desenvolvimento deste trabalho.

A todos aqueles que não foram citados, mas que tiveram participação na conclusão desta etapa da minha vida.

Resumo

Este trabalho se concentra na classificação automática de faltas do tipo curto-circuito em linhas de transmissão. A maioria dos sistemas de transmissão possuem três fases (A, B e C). Por exemplo, um curto-circuito entre as fases A e B pode ser identificado como uma falta "AB". Considerando a possibilidade de um curto-circuito com a fase terra (T), a tarefa ao longo desse trabalho é classificar uma série temporal em uma das 11 faltas possíveis: AT, BT, CT, AB, AC, BC, ABC, ABT, ACT, BCT, ABCT. Estas faltas são responsáveis pela maioria dos distúrbios no sistema elétrico. Cada curto-circuito é representado por uma seqüência (série temporal) e ambos os tipos de classificação, *on-line* (para cada curto segmento extraído do sinal) e *off-line* (leva em consideração toda a seqüência), são investigados. Para evitar a atual falta de dados rotulados, o simulador *Alternative Transient Program* (ATP) é usado para criar uma base de dados rotulada e disponibilizada em domínio público. Alguns trabalhos na literatura não fazem distinção entre as faltas ABC e ABCT. Assim, resultados distinguindo esse dois tipos de faltas adotando técnicas de pré-processamento, diferentes *front ends* (por exemplo wavelets) e algoritmos de aprendizado (árvores de decisão e redes neurais) são apresentados. O custo computacional estimado durante o estágio de teste de alguns classificadores é investigado e a escolha dos parâmetros dos classificadores é feita a partir de uma seleção automática de modelo. Os resultados obtidos indicam que as árvores de decisão e as redes neurais apresentam melhores resultados quando comparados aos outros classificadores.

Abstract

This work concerns automatic classification of short circuits in transmission lines. Most transmission systems use three *phases*: A, B and C. Hence, a short-circuit between phases A and B will be identified as "AB". Considering the possibility of a short-circuit to "ground" (T), the task is to classify a time series into one among eleven possibilities: AT, BT, CT, AB, AC, BC, ABC, ABT, ACT, BCT, ABCT. These faults are responsible for the majority of the disturbances in electric power systems. Each short circuit is represented by a sequence (time-series) and both online (for each short segment) and offline (taking in account the whole sequence) classification are investigated. To circumvent the current lack of labeled data, the Alternative Transient Program (ATP) simulator is used to create a public comprehensive labeled dataset. Some works in the literature fail to distinguish between ABC and ABCT faults. Then, results differentiated these two faults types adopting preprocessing techniques, different front ends (e.g., wavelets) and learning algorithms (e.g., decision trees and neural networks) are presented. The computational cost of the some classifiers during the test stage is investigated and the choosing parameters of classifiers is done by automatic model selection. The results indicate that decision trees and neural networks outperform the other methods.

Sumário

Lista de Figuras **iv**

Lista de Tabelas **vii**

Lista de Símbolos **viii**

Trabalhos publicados pelo autor **xi**

1 Introdução **1**

1.1 Motivação e descrição geral do problema 1

1.2 Metodologia e objetivos 4

1.3 Revisão Bibliográfica 5

1.4 Estrutura do Trabalho 8

2 Classificação de séries temporais **9**

2.1 Introdução 9

2.2 Representação de séries temporais para fins de classificação 10

2.3 Classificação de faltas on-line versus off-line 11

2.3.1 Classificação on-line 11

2.3.2 Classificação pós-falta 12

2.4 Software WEKA para classificação convencional 13

2.5 Classificadores convencionais 14

2.5.1 Redes neurais artificiais 14

2.5.2 Árvores de decisão 17
2.5.3 K-Nearest Neighbors (KNN) 20
2.5.4 Naïve Bayes 21
2.5.5 Misturas de Gaussianas 23
2.6 Classificadores de seqüências 24
2.6.1 Classificação de seqüências baseada em quadros 24
2.6.2 *Hidden Markov Models* 25
2.6.3 *Dynamic Time Warping* (DTW) 29
2.7 Conclusões do Capítulo 31

3 UFPAFaults: Uma base de dados de faltas em linhas de transmissão 32

3.1 Introdução 32
3.1.1 *Alternative Transient Program* 32
3.1.2 AmazonTP 35
3.2 Composição da base de dados UFPAFaults 37
3.3 Parametrização da base de dados 38
3.3.1 Pré-processamento 39
3.3.1.1 Decimação e acréscimo de ruído 39
3.3.1.2 Normalização 39
3.3.1.3 *Trigger* 40
3.3.2 *Front end* 41
3.3.2.1 Direto da forma de onda - *Raw* 42
3.3.2.2 *Wavelet* 42
3.3.3 Seleção dos Parâmetros do Modelo 45
3.4 Conclusões do capítulo 47

4 Experimentos 48

4.1 Introdução 48

4.2 Resultados para classificação on-line 48
4.3 Discussão acerca de custo computacional 53
4.4 Resultados para classificação pós-falta 56
4.5 Conclusões do capítulo 59

5 Conclusões **61**

5.1 Conclusões 61
5.2 Sugestões de trabalhos futuros 62

Referências Bibliográficas **63**

A Parâmetros dos classificadores **67**

B Scripts utilizados **70**

Lista de Figuras

1.1 Esquema de transmissão de energia elétrica [1] 3

2.1 Classificação convencional considerando $L = 1$ e $K = 6$. 12
2.2 Estrutura de um neurônio artificial. 14
2.3 Exemplo de uma RNA MLP. 15
2.4 Exemplo de árvore de decisão. 18
2.5 Esquema da classificação de seqüências utilizando arquitetura FBSC adotando-se $L = 1$ ($K = 6$). 25
2.6 Estrutura de um HMM com três estados para arquitetura esquerda-direita. . 26
2.7 Modelo de HMM contínuo adotado no software HTK com três estados "emissores" e dois estados extras. A distribuição de saída de cada estado é uma mistura de três Gaussianas. 26
2.8 Fundamentos do HTK. 28
2.9 Estágios de processamento do HTK (vide [2] para maiores detalhes). 28

3.1 Esquema da metodologia adotada para composição da base UFPAFaults e seu posterior uso. 33
3.2 Principais programas relacionados ao ATP. 34
3.3 Bloco para simulação de uma falta (representado com o software ATDraw). Os elementos SW são as chaves e R são as resistências. 35
3.4 Zoom das formas de onda no momento de uma falta AB gerada pelo AmazonTP. 36
3.5 Representação gráfica do sistema Eletronorte no ATPDraw. 37

3.6 Diagrama do modelo ATP para o sub-circuito Tramoeste da Eletronorte com três lihas Z-T (rotuladas "C1"). 38
3.7 Exemplo de formas de onda de tensão (f_s = 2kHz) após a aplicação das normalizações *prefault* (a) e *allfault* (b). Em (a) o pico maior do que 1 pode ser visto enquanto que em (b) o maior pico é exatamente 1. 41
3.8 Exemplo do uso do *trigger* com $f_s = 2$ kHz. 42
3.9 Cabeçalho do arquivo gerado em formato arff do WEKA com $K = 30$ para $L = 5$. 43
3.10 Esquema da decomposição *wavelet* com um nível $D = 1$. 43
3.11 Exemplo de uma decomposição *wavelet* aplicada aos sinais de tensão das fases A e B de uma falta AB simulada no intervalo de 1s. A *wavelet* é uma Daubechies 4 com três níveis de decomposição ($D = 3$). 44
3.12 *Grid* da faixas de valores dos parâmetros -C e -M de uma árvore de decisão. . 47

4.1 Taxa de erro E_f usando o *front end raw* e as estratégias de normalização *prefault* (a) e *allfault* (b). 49
4.2 Resultados (E_f) para os dois *front ends wavelet*. Os melhores resultados utilizando o *front end raw* com normalização *prefault* também é apresentado para efeito de comparação. 50
4.3 Gráfico das matrizes de confusão para rede neural (lado esquerdo) e árvore de decisão J4.8 (lado direito). As colunas indicam as faltas reconhecidas. Os raios dos círculos são proporcionais ao tamanho das entradas. 52
4.4 Exemplo das formas de onda de tensão e corrente das faltas ABC (a) e ABCT(b). Tais gráficos mostram a grande similaridade entre as duas faltas. . 52
4.5 Diferença entre o erro E_f para classificação quadro-a-quadro sem ruído e com ruído usando o *front end raw* e a normalização *prefault*. 54

4.6 Comparação do tempo para o estágio de teste e do custo computacional estimado usando a Tabela 4.6 para os melhores classificadores encontrados pela seleção de modelos, para a normalizações *prefault* com $L = 9$. Os valores estimados do custo dos classificadores foram divididos por 400 para efeito de comparação. O custo computacional estimado do classificador KNN foi em torno de 150 (já dividido por 400) e seu tempo de processamento igual a 148 s, o que o deixa fora da escala dos demais. 57

4.7 Erro E_s para classificação de seqüência baseada em quadros usando os *front ends raw, waveletconcat ($L_{min} = 9$ e $S_{min} = 4$) e waveletenergy* e as estratégias de normalização *prefault*(a) e *allfault*(b). 57

4.8 Diferença $E_f - E_s$ entre as taxas de erro para classificação quadro-a-quadro e seqüência usando a normalização *prefault* e *front-end raw*. 58

4.9 Erro E_s para classificação de seqüências utilizando HMM, segmental KNN e FBSC. 58

Lista de Tabelas

1.1 Distúrbios relacionados à qualidade de energia elétrica e suas respectivas causas [3]. 2

4.1 Parâmetros dos *front ends* usados nas simulações. Além desses, $Q = 6$ e a *wavelet* mãe foi db4 com $D = 3$ níveis de decomposição. 49

4.2 Matriz de confusão do classificador RNA utilizando *front end raw* e normalização *prefault*. As linhas indicam as faltas corretas e as colunas a classe predita pelo classificador. 51

4.3 Matriz de confusão do classificador J4.8 utilizando *front end raw* e normalização *prefault*. 51

4.4 Matriz de confusão do classificador RNA não diferenciando as faltas ABC e ABCT. 53

4.5 Matriz de confusão do classificador J.48 não diferenciando as faltas ABC e ABCT. 53

4.6 Custo computacional estimado para alguns classificadores, onde *sigm* é uma função sigmóide e mac é uma operação de multiplicação e adição, executada em um ciclo em chips DSP modernos. 55

4.7 Pesos para diferentes operações [4]. Por exemplo, uma sigmóide pode ser calculada com uma função exponencial, uma adição e uma divisão, tendo peso total igual a 44. 56

A.1 Resumo do conjunto dos parâmetros para os classificadores. 68

Lista de Símbolos

$\mathbf{X}_n$	-	Matriz de dimensão $Q \times T_n$ representando a n-ésima falta em uma base de dados
Q	-	Número de sinais de tensão e corrente nas fases A, B e C
T_n	-	Número de amostras multidimensionais da n-ésima falta
$\mathbf{F}$	-	Matriz $Q \times L$ chamada quadro (*"frame"*), a qual aglutina L amostras de $\mathbf{X}$
L	-	Número de amostras (tamanho) do quadro (*"frame"*)
$\hat{\mathbf{Z}}$	-	Matriz $Q \times (LN)$ correspondente à concatenação de todos os quadros de $\mathbf{X}$
N	-	Número total de quadros em $\mathbf{X}$
S	-	Deslocamento do quadro: número de amostras entre as amostras de início de quadros consecutivos
$\mathbf{Z}$	-	Matriz chamada "instance", a qual corresponde à matriz $\hat{\mathbf{Z}}$ redimensionada, de maneira a ter dimensão $K \times N$, por conveniência
K	-	Número de linhas de $\mathbf{Z}$, onde $K = QL$
$(\mathbf{Z}, y)$	-	Exemplo, formado por uma "instance" $\mathbf{Z}$ e sua classe y
M	-	Número de exemplos $(\mathbf{Z}_n, y_n), n = 1, \ldots, M$, em um conjunto de treino
y	-	Rótulo ou classe, a qual corresponde à saída de um classificador
$\mathcal{F}$	-	Classificador convencional
$\mathcal{G}$	-	Classificador de seqüências
d_{max}	-	Nível mais profundo dentre todas as folhas de uma árvore de decisão
K	-	Número de vizinhos mais próximos no classificador KNN
P	-	Usado para ambas funções massa e densidade de probabilidade
f_s	-	Freqüência de amostragem

$P(y\|z)$	-	Probabilidade a *posteriori*
$P(z\|y)$	-	Probabilidade (ou densidade) *likelihood*
$P(y)$	-	Probabilidade a *priori*
$P(z)$	-	Probabilidade (ou densidade) chamada *evidência*
$\mathbf{\Sigma}$	-	Matriz de covariância
G_y	-	Número de Gaussianas em uma mistura que representa a classe y
$g_i(\mathbf{Z})$	-	Score acumulado para cada classe i
E_f	-	Taxa de erro de classificação para o módulo quadro-a-quadro (on-line)
E_s	-	Taxa de erro de classificação para o módulo de seqüência (pós-falta)
S_i	-	Número de estados no i-ésimo modelo HMM
W	-	Percurso de ajuste em um DTW
$DTW(\mathbf{Z}, \hat{\mathbf{Z}})$	-	Custo da deformação entre duas seqüências
$\gamma(i, j)$	-	Distância acumulativa usada na programação dinâmica do DTW
$X_{\max}$	-	Valor máximo absoluto de um segmento de forma de onda
$s(t)$	-	Sinal analisado na transformada wavelet
$g(k)$	-	Filtro passa baixa da transformada wavelet
$h(k)$	-	Filtro passa alta da transformada wavelet
$\mathbf{a}$	-	Coeficiente de aproximação da transformada wavelet
$\mathbf{d}$	-	Coeficiente de detalhe da transformada wavelet
L_{min}	-	Tamanho do quadro para os coeficientes wavelet que têm menor f_s
S_{min}	-	Deslocamento para os coeficientes wavelet que têm menor f_s
D	-	Número de níveis da decomposição wavelet
T_a	-	Número de amostras no coeficiente de aproximação $\mathbf{a}$
E	-	Energia em cada banda de freqüência obtida pela decomposição wavelet

p_n - Número de nós internos em uma árvore de decisão

p_h - Número de neurônios na camada escondida de uma rede neural artificial

$sigm$ - Função sigmóide

mac - Número de operações de adição e multiplicação, executadas em um ciclo em modernos chips DSP

p_m - Número de centróides no classificador KNN

p_g - Número de Gaussianas por mistura no classificador GMM

Trabalhos Publicados Pelo Autor

1. J. M. Morais, Y. P. Pires, C. Cardoso, A. B. R Klautau. "An Experimental Evaluation of Automatic Classification of Sequences Representing Short Circuits in Transmission Lines". *In: VIII CBRN - Congresso Brasileiro de Redes Neurais*, Florianópolis-SC, Brasil, outubro 2007.

2. J. M. Morais, Y. P. Pires, C. Cardoso, A. B. R Klautau. "Data Mining Applied to the Electric Power Industry: Classification of Short-Circuit Faults in Transmission Lines". *In: 7th International Conference on Intelligent Systems Design and Applications (ISDA)*, Rio de Janeiro-RJ, Brasil, outubro 2007.

3. Y. P. Pires, J. M. Morais, G. Guedes, J. Borges, M. Nunes, A. B. R. Klautau. "AMAZONTP: Uma Ferramenta para Simulação de Eventos de Qualidade de Energia Elétrica". X*XVII Congresso Ibero-Americano de Métodos Computacionais em Engenharia (CILANCE)*, Belém-PA, Brasil, setembro 2006.

Capítulo 1

Introdução

1.1 Motivação e descrição geral do problema

Devido às novas normas e à descentralização no setor elétrico, em muitos países, incluindo o Brasil, é grande a necessidade das concessionárias em garantir a qualidade da energia elétrica (QEE) entregue a seus consumidores [5,6], sob o risco de serem penalizadas com pagamento de multas bastante elevadas.

QEE é geralmente definida como qualquer distúrbio ou alteração nos valores de tensão, corrente ou desvio de freqüência que resulte em uma falta ou má operação dos equipamentos dos consumidores [7].

Os distúrbios de qualidade de energia elétrica são provocados por defeitos no sistema elétrico (Tabela 1.1) [3] como, por exemplo, faltas do tipo curto-circuito. Tais situações faltosas podem ocorrer em diversos componentes de um sistema de potência, dentre os quais podemos destacar as "linhas de transmissão" (LT), representadas na Figura 1.1, especialmente se considerarmos suas dimensões físicas, complexidade funcional e o ambiente em que se encontram, o que apresentaria uma maior dificuldade para manutenção e monitoramento [8].

Após a energia elétrica ter sido gerada na casa de força, a estação distribuidora transporta a eletricidade através de cabos aéreos. Fora dos centros urbanos, esse cabos são pendurados em grande torres de metal. O conjunto desses cabos e torres forma uma linha de transmissão. A eletricidade passa por diversas subestações, durante o percurso entre as usinas e as cidades, onde equipamentos denominados de transformadores aumentam ou diminuem a tensão elétrica que será repassada ao sistema de distribuição.

Atualmente, as indústrias de energia elétrica possuem uma logística sofisticada para aquisição e armazenamento de séries temporais (formas de onda) correspondentes a eventos

Distúrbio	**Descrição**	**Causas**
Interrupções	Interrupção total da alimentação elétrica	Curto-circuitos, descargas atmosféricas, e outros acidentes que exijam manobras precisas de fusíveis, disjuntores,etc.
Transientes	Distúrbio na curva senoidal, resultando em rápido e agudo aumento de tensão	Descargas atmosféricas Manobras da concessionária ou Manobras de grandes cargas e bancos de capacitores
Sag / Swell	Subtensões (sags) ou sobretensões (swells) curtas (meio ciclo até 3 segundos) Sags respondem por cerca de 87% de todos os distúrbios elétricos	Queda/Partida de grandes equipamentos Curto-circuitos, falha em equipamentos ou manobras da concessionária
Ruídos	Sinal indesejado de alta freqüência que altera o padrão normal de tensão (onda senoidal)	Interferência de estações de rádio e TV Operação de equipamentos eletrônicos
Harmônicos	Alteração do padrão normal de tensão causada por freqüências múltiplas da fundamental (50-60Hz)	UPS, Reatores eletrônicos, inversores de freqüência, retificadores e outras cargas não-lineares
Variações de Tensão de Longa Duração	Variações de tensão com duração acima de 1 minuto	Equipamentos e fiação sobrecarregados, Utilização imprópria de transformadores Fiação subdimensionada ou conexões mal feitas

Tabela 1.1: Distúrbios relacionados à qualidade de energia elétrica e suas respectivas causas [3].

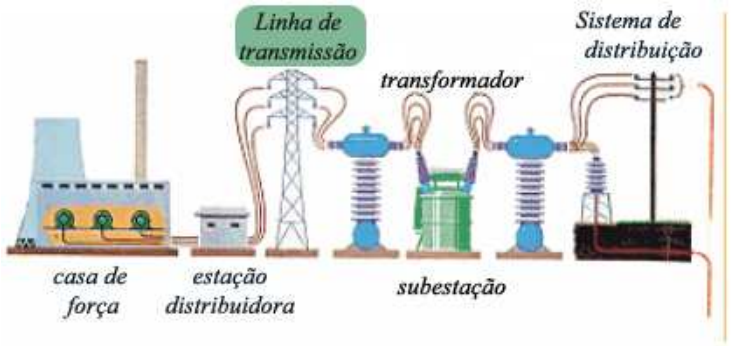

Figura 1.1: Esquema de transmissão de energia elétrica [1] .

de qualidade de energia elétrica. Entretanto, minerar tais dados para inferir, por exemplo, relações de causa-efeito é ainda uma atividade incipiente. Um dos principais problemas é que as séries temporais digitalizadas são freqüentemente armazenadas sem serem devidamente rotuladas, o que complica a aplicação de um algoritmo de *aprendizado supervisionado* [9]. Por exemplo, equipamentos de oscilografia [10] implementam algoritmos relativamente simples que detectam se as formas de onda de tensão desviam dos seus valores de amplitude nominal. Se a variação é maior que um limiar, o circuito programável denominado de *trigger* inicia o armazenamento dos dados, junto com informações adicionais tais como a data e a hora.

O problema encontra-se no fato de que, tipicamente, não existem ações para relacionar os eventos com dados adicionais que poderiam ajudar a inferir a causa. Estabelecer esta relação é um objetivo a longo prazo nas indústrias de energia elétrica. Isto requer um *data warehouse* para integrar seus sistemas legados SCADA (Controle e Aquisição de Dados Supervisórios), os quais informam tempo em segundos e não fornecem formas de onda, com os denominados dispositivos eletrônicos inteligentes (IEDs) como os registradores de falta digitais (DFRs) ou relés digitais, que podem suportar freqüências de amostragem de até 10 kHz e implementam sofisticados algoritmos [10].

Antes deste cenário chegar a ser uma realidade, há uma necessidade de se estabelecer estratégias eficientes, por meio da inteligência computacional, que ajudem os especialistas a rotularem os dados de séries temporais. A motivação inicial deste trabalho foi o desenvolvimento de um projeto de Pesquisa & Desenvolvimento em parceria com a Eletronorte, uma companhia de energia elétrica que tem uma grande quantidade de dados de oscilografia não rotulados e interesse em minerar tais dados para estabelecer relações de causa-efeito. A próxima seção descreve a metodologia e estabelece os objetivos adotados para evitar o círculo vicioso: não há algoritmo para rotular corretamente os dados pois não existem dados rotulados para aplicação de algoritmos de aprendizado supervisionado.

1.2 Metodologia e objetivos

A metodologia adotada neste trabalho é baseada estritamente no escopo da investigação de uma solução para análise automática de uma particular e importante classe de causas de eventos de qualidade de energia elétrica: faltas do tipo curto-circuito em linhas de transmissão. Estudos mostraram que essas faltas foram responsáveis por 70% dos distúrbios e blackouts [7, 11]. Daí a importância de identificar essas faltas em sistema de potência.

Na primeira etapa, as faltas são simuladas com o simulador *Alternative Transient Program* (ATP) [12] e técnicas de mineração de dados (pré-processamento e algoritmos de aprendizagem de máquina) são usadas para treinar e testar os classificadores. Modelos ATP têm uma longa história de boa reputação e são muito utilizados para descrever de forma concisa o comportamento atual dos sistemas. Assim, os dados gerados artificialmente podem ser (certamente) usados para treinamento dos classificadores. Na segunda etapa, os classificadores ajudarão a pré-rotular os dados de oscilografia, a partir do qual um subconjunto é pós-processado pelos especialistas para corrigir ou confirmar os rótulos, avaliar o procedimento e permitir o refinamento através de novas iterações.

O principal objetivo deste trabalho é apresentar os resultados obtidos na primeira etapa. Este não inclui a avaliação dos procedimentos com dados atuais de oscilografia. Todavia, já é possível prover duas importantes contribuições.

A primeira é o desenvolvimento e a livre distribuição de uma base de dados de séries temporais rotuladas baseado no framework proposto em [13] denominada de UFPAFaults. Similarmente a outras áreas onde se aplica mineração de séries temporais [14], investigações em QEE sofrem da falta de "benchmarks" padronizados e livremente disponíveis. As bases de dados de eventos de QEE são freqüentemente proprietárias e, conseqüentemente, a reprodução prévia de resultados é impossível.

A segunda contribuição é uma revisão dos diversos assuntos e avaliação dos algoritmos de pré-processamento e aprendizado de máquina aplicados à classificação de faltas. A maior parte da literatura em classificação de faltas adota como *front end* a forma de onda em sua forma original (aqui chamado *raw*) ou *wavelet* e redes neurais artificiais como algoritmo de aprendizado.

Mais especificamente, este trabalho apresenta uma comparação preliminar entre redes neurais, árvores de decisão e outros classificadores, assumindo os dois tipos de *front end*: *raw* e *wavelet*.

1.3 Revisão Bibliográfica

Propostas recentes de classificação de faltas do tipo curto-circuito em linhas de transmissão têm usado técnicas de processamento digital de sinais (janelamento, representação rms e transformada *wavelet*) como ferramenta para extração das características do sinal, essas características são usadas como entrada para os sistemas de reconhecimento de padrões e de classificação baseados em inteligência computacional. Alguns trabalhos baseado nesses método são apresentados.

As publicações de Mladen Kezunovic (veja, por exemplo, [11,15–18]) representam bem a evolução do assunto. O artigo [11] é um dos primeiros trabalhos que utiliza as redes neurais artificiais no processo de detecção e classificação de faltas em linhas de transmissão. Foram geradas 1000 situações faltosas utilizando o simulador EMTP divididas em um conjunto de treino e teste. A freqüência de amostragem foi de 2 kHz e a extração dos parâmetros foi feita a partir do janelamento das formas de onda de tensão e corrente com tamanho da janela fixado em 1 ciclo. No processo de classificação foram consideradas 12 classes (11 tipos de faltas e a situação de não-falta). A taxa de acerto da rede neural foi de 95.93%.

Em [15–17] ao invés de utilizar uma rede neural multi-layer Perceptron utiliza-se uma rede baseada no algoritmo de agrupamento ISODATA que pertence a um grupo de redes neurais denominada de mapas auto-organizáveis. Esta rede neural não possui camada escondida e sua estrutura auto-organizada depende somente do conjunto de dados de entrada apresentado. O treinamento da rede neural é dividido em dois estágios: não-supervisionado e supervisionado. No aprendizado não-supervisionado, padrões são apresentados sem seus rótulos categóricos, e este procedimento tenta identificar protótipos que podem servir como centros dos grupos formados. O aprendizado não-supervisionado forma um conjunto de grupos que podem ser homogêneos se possuir somente padrões com a mesma classe ou heterogêneos se possuírem padrões com duas ou mais classes. No aprendizado supervisionado, a classe é associada com cada padrão de entrada permitindo a identificação e a separação dos clusters homogêneos e não-homogêneos. O parâmetro denominado de vigilante é utilizado como uma medida de confiança que decresce durante as iterações. Este parâmetro controla o tamanho e o número de grupos gerados. Foi utilizado o simulador *Alternative Transient Program* ATP para simular diferentes situações faltosas variando alguns parâmetros tais como: ângulo da falta, resistência da falta com relação a fase terra e distância. Assim como em [11], foram considerados 11 tipos de faltas mais o estado de não-falta. Em [15] foi aplicada a técnica de janelamento nas formas de onda de tensão e corrente sendo o tamanho da janela fixado em 1 ciclo em uma freqüência de amostragem de 2 kHz. Os dados de treino e teste foram normalizados com média zero e variância um. Na fase de teste foi utilizado o algoritmo k-vizinho mais próximo(KNN). A

menor taxa de erro de classificação para o melhor k (k= 1) foi de 0.48%. A contribuição de [16] foi acrescentar uma técnica de classificação fuzzy que generaliza o algoritmo KNN. Em [17] é proposta uma melhoria no algoritmo de aprendizagem supervisionada e foram avaliados vários tamanhos de janelas e diferentes freqüência de amostragem. A taxa de erro média nesse trabalho foi de 0.82%.

Em [18] é proposta uma ferramenta integrada em tempo real para análise de falta em linha de transmissão. As duas principais técnicas usadas na ferramenta de análise de faltas são: rede neural baseada na teoria da ressonância adaptativa fuzzy e a sincronização de amostras. O artigo faz várias extensões das duas técnicas de forma que elas podem ajustar-se bem nas situações reais. Um estudo de avaliação é implementado para comparar a ferramenta de análise proposta com a tradicional (distância entre relés).

Em [19] é apresentado um classificador de faltas que utiliza uma abordagem híbrida baseada em redes neurais artificiais e *wavelet*. O esquema de classificação no estudo realizado foi definido como um problema multi-classe com 10 faltas do tipo curto-circuito. Várias condições de faltas foram simuladas com o programa de simulação PSCAD/EMTDC a partir de um sistema de distribuição. A técnica da análise multi-resolução *wavelet* (MRA) foi utilizada para extração dos parâmetros das faltas a partir das formas de onda de tensão e corrente. Em seguida a rede neural foi empregada para tarefa de classificação. A rede neural utilizada foi uma rede auto-organizável, com o algoritmo de aprendizado de Kohonen e a técnica de LVQ (Learning Vector Quantization) tipo-um. A taxa de acerto no estágio de treino foi de 99% e para o estágio de teste foi 92%.

Em [20] é proposto um método de detecção e classificação de faltas em linhas de transmissão através da análise dos registros de oscilografia. No módulo de detecção o primeiro passo é coletar as amostras de tensão e corrente dos registros de oscilografia. As amostras de corrente são normalizadas e em seguida a transformada *wavelet* (Daubechies-4) discreta é calculada. Posteriormente, calcula-se a energia dos coeficientes *wavelet* gerados. No caso de não falta, nenhum dado é transferido. Caso contrário, as amostras relacionadas a falta são identificadas por meio do coeficientes *wavelet* da corrente. No módulo de classificação somente as amostras relacionadas com a falta são analisadas. Primeiro, essas amostras são normalizadas. Em seguida, as formas de onda são reamostradas para uma freqüência de amostragem de $f_s = 1200$ Hz. Um janelamento das amostras forma um conjunto de padrões de entrada para a rede neural. O treinamento da rede neural é realizado com dados obtidos por simulação e na fase de teste utiliza-se os dados da oscilografia. Uma codificação binária é usada para a saída da rede neural de maneira que uma falta é caracterizada pela presença (1) ou ausência (0) de uma ou mais fases e da fase terra. Então, por exemplo, uma falta AB é expressa pela saída binária 0110. A taxa de erro de classificação na fase de teste foi de 0.27%.

Em [21] baseado na análise wavelet e na idéia de entropia, o conceito e a definição de entropia da energia *wavelet* (WEE) para extração dos parâmetros das faltas é apresentado. Primeiramente aplica-se uma decomposição *wavelet* no sinal adquirido e em seguida foi calculado a WEE para realizar a extração dos parâmetros.Os vetores característicos contendo informações sobre a falta são usados como a entrada para o sistema de inferência neuro-fuzzy adaptativo. A rede neural, assim como em [20], é composta por 4 neurônios na camada de saída de maneira que uma falta do tipo AG é representada pela saída 0001. Neste trabalho foram considerados apenas 9 tipos de falta.

Em [22] um estudo comparativo do desempenho da transformada de Fourier e da transformada wavelet baseado em métodos para detecção, classificação e localização de faltas em linhas de transmissão é apresentado. Os algoritmos criados são baseados na análise da transformada de Fourier de sinais transitórios de corrente registro a partir de um curto-circuito na linha de transmissão.Análise similar é realizada usando a análise multi-resolução *wavelet* (Daubechies-8) e um comparativo entre os dois métodos é discutido.

Em [23] é proposto um método baseado no algoritmo SVM para classificação de faltas. O desempenho do método foi avaliado treinando e testando o classificador SVM a partir de dados obtidos de diferentes fontes. Os dados usados nos experimentos foram obtidos a partir de duas fontes: registros de oscilografia de dois sistemas de potência distintos e dados simulados. As faltas são caracterizadas pelos seus valores rms e são divididas em quatro tipos: fase-terra, fase-fase-terra, fase-fase, fase-fase-fase. Outro tipo de distúrbio de tensão considerado no trabalho é quanto saturação de transformador. Assim no artigo foram consideradas 5 classes e diferentes cenários experimentais. Para classificar os 5 tipos de distúrbios foram usadas 5 SVMs binárias. Foram apresentados resultados para os diferentes cenários experimentais adotados, sendo que a taxa de acerto média dos experimentos foi em torno de 92%.

Em [24] é proposto um método baseado em redes neurais para classificação de distúrbios elétricos em tempo real. Foi desenvolvido um protótipo de um sistema para tal tarefa. O sistema consiste de uma aplicação desktop que inclui um cartão de aquisição, um ambiente de monitoramento do sinal adquirido, e um núcleo de classificação baseado em inteligência computacional (particularmente redes neurais) para classificar possíveis distúrbios. Além disso, foi desenvolvido um gerador de padrões elétricos como uma forma de prover as redes neurais com um conjunto de dados de treino. Para extração dos parâmetros foi utilizado a decomposição *wavelet*. A partir dos coeficientes de detalhe da decomposição *wavelet* foram extraídos os seguintes valores: a energia, o valor máximo absoluto e o valor rms das formas de onda. Foram projetadas 4 rede neurais: uma para detectar se houve ou não distúrbio, e três redes neurais paralelas para classificar diferentes distúrbios existente no sinal.

1.4 Estrutura do Trabalho

Este trabalho está divido em cinco capítulos, organizados da seguinte maneira: o primeiro capítulo discute de forma introdutória a motivação e a descrição do problema, a metodologia, os objetivos e a estrutura da pesquisa. O Capítulo 2 apresenta o conceito de séries temporais, a representação de faltas em linhas de transmissão e as principais arquiteturas para classificação de faltas, assim como os algoritmos de aprendizado utilizados. No Capítulo 3 é exposta a metodologia empregada para o processo de composição e parametrização da base de dados UFPAFaults com os eventos de QEE. No Capítulo 4 são analisados os cenários usados para os experimentos e seus respectivos resultados. Finalmente, no Capítulo 5 as conclusões e as propostas para prosseguimento do trabalho são apresentadas.

Capítulo 2

Classificação de séries temporais

2.1 Introdução

É crescente a quantidade de dados armazenados em um sistema computacional. Uma boa parte desses dados, em particular dados coletados automaticamente por aplicações de monitoração, são dados de séries temporais. Uma série temporal [25] é uma seqüência de observações realizadas ao longo do tempo. Esta dependência em relação ao tempo as distingue de outros tipos de dados estatísticos. Neste trabalho, as séries temporais representam faltas do tipo curto-circuito em linhas de transmissão de sistema de potência.

Uma das principais atividades na análise de séries temporais de faltas é a **classificação**. Esta consiste na busca de uma função (mapeamento) que permita associar cada série temporal a um rótulo categórico denominado de classe [9, 26]. Uma vez determinada em uma fase de treino, essa função pode ser aplicada em uma fase de teste a novos registros, de forma a prever as classes nas quais tais séries melhor se enquadram.

A maioria dos sistemas de transmissão possuem três fases (A, B e C). Por exemplo, um curto-circuito entre as fases A e B pode ser identificado como uma falta "AB". Considerando a possibilidade de um curto-circuito com a fase terra (T), a tarefa ao longo desse trabalho é classificar uma série temporal em uma das 11 faltas possíveis: AT, BT, CT, AB, AC, BC, ABC, ABT, ACT, BCT, ABCT. Algoritmos para resolver este problema de classificação são usados por registradores digitais de faltas (DFRs), relés digitais e outros equipamentos [10].

Os equipamentos de captura de sinais podem ser alocados nos dois pontos terminais da linha de transmissão. A maioria deles são capazes de digitalizar formas de onda de tensão e corrente. Assume-se, também, que cada equipamento possui um circuito gatilho (*trigger*) que descarta amostras da forma de onda sem qualquer tipo de anomalia e armazena somente

o intervalo de interesse (a falta) e um número predefinido de amostras antes e depois da falta. O *trigger* será brevemente discutido na Seção 3.3.1.3.

A classificação de séries temporais de faltas em linhas de transmissão corresponde a um problema de classificação de seqüências, uma tarefa de classificação especial onde os dados de entrada são representados por uma matriz que possui tamanho variável. Esta matriz de entrada pode ser representada e classificada de diversas maneiras. Por isso, o objetivo deste capítulo é fornecer uma notação precisa e elucidativa sobre possíveis alternativas.

2.2 Representação de séries temporais para fins de classificação

Cada falta é uma série temporal multivariada de duração variável. A n-ésima falta $\mathbf{X}_n$ em uma base de dados (registros de oscilógrafos, por exemplo) é representada por uma matriz $Q \times T_n$. A coluna $\mathbf{x}_t$ de $\mathbf{X}_n$, $t = 1, \ldots, T_n$, é uma *amostra* multidimensional representada por um vetor de Q elementos. Por exemplo, neste trabalho adota-se $Q = 6$ (tensão e corrente das fases A, B e C) nos experimentos. Em algumas situações [18], é possível obter *amostras sincronizadas* dos dois pontos terminais de uma linha de transmissão. Neste caso, a amostra é um vetor duas vezes maior do que quando se captura a forma de onda de apenas um ponto terminal. Para o exemplo anterior, a dimensão da amostra para medidas nos dois pontos terminais seria $Q = 12$.

Uma amostra composta por valores de tensão e corrente é denominada de *raw* (ou seja, direta ou simples). Alternativamente, outras representações, paramétricas tais como wavelets [27] e transformada de Fourier [28] podem ser usadas.

Independente da representação paramétrica adotada, uma simples amostra geralmente não carrega informação suficiente que permita a realização de decisões razoáveis. Por este motivo, as amostras são freqüentemente concatenadas para criar um quadro (*frame*) $\mathbf{F}$. Os quadros tem dimensão $Q \times L$, onde L é o tamanho do quadro (*frame length*) e sua concatenação $\hat{\mathbf{Z}} = [\mathbf{F}_1 \ldots \mathbf{F}_N]$ é uma matriz de dimensão $Q \times LN$, onde N é o número de quadros.

Os quadros podem sobrepor-se de tal forma que o *deslocamento do quadro* S, isto é, o número de amostras entre o início de dois quadros consecutivos pode ser menor do que o tamanho da janela. Conseqüentemente, o número de quadros para uma falta $\mathbf{X}_n$ é igual a

$$N_n = 1 + \lfloor (T_n - L)/S \rfloor \tag{2.1}$$

onde $\lfloor \cdot \rfloor$ é uma função *floor* que retorna o maior inteiro não maior do que o valor passado

como parâmetro. Deve-se ressaltar que se $S = L$ (não há sobreposição) e um quadro é uma *concatenação de amostras* $\mathbf{F} = [\mathbf{x}_{t-0.5(L-1)}, \ldots, \mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1}, \ldots, \mathbf{x}_{t+0.5(L-1)}]$, as matrizes $\mathbf{X} = \hat{\mathbf{Z}}$ coincidem.

Os quadros (matrizes) podem ser convenientemente organizados como vetores de dimensão $K = QL$, e $\hat{\mathbf{Z}}$ redimensionada para criar $\mathbf{Z} = [\mathbf{z}_1 \ldots \mathbf{z}_N]$ de dimensão $K \times N$. É assumido, daqui por diante, que o processamento é realizado em $\mathbf{Z}$ (não na série original $\mathbf{X}$), visto que $\mathbf{Z}$ permite aos algoritmos de classificação não se importarem com o processo de janelamento.

2.3 Classificação de faltas on-line versus off-line

Sistemas de classificação de faltas podem ser divididos em dois tipos: sistemas de classificação de faltas *on-line* e pós-falta (ou *off-line*). Ambos serão discutidos nas próximas subseções, ressaltando-se suas relações com a classificação convencional e classificação de seqüências.

2.3.1 Classificação on-line

A classificação de faltas on-line [18, 24] tem como foco a classificação quadro-a-quadro $\mathbf{F}$ e geralmente é realizada em nível, por exemplo, de proteções de relés. Os sistemas de classificação de faltas *on-line* lidam com vetores $\mathbf{z}$ de dimensão fixa K e devem fornecer uma saída em espaço de tempo relativamente curto. Este tempo é freqüentemente baseado em um quadro correspondente à metade de um ciclo ou um ciclo completo de um sinal senoidal de 60 ou 50 Hz. Assumindo 60 Hz e uma freqüência de amostragem $f_s = 2$ kHz, um ciclo corresponde a $L = 2000/60 \approx 33$ amostras. Assim, os sistemas *on-line* tentam resolver problemas que podem ser tratados como problemas de *classificação convencional*, onde uma decisão deve ser tomada para cada 33 amostras do sinal, por exemplo.

Na classificação convencional tem-se um *conjunto de treino* $\{(\mathbf{z}_1, y_1), ..., (\mathbf{z}_M, y_M)\}$ contendo M *exemplos*. Cada exemplo $(\mathbf{z}, y)$ consiste de um vetor $\mathbf{z} \in \mathbb{R}^K$ chamado *"instance"* e um *rótulo* $y \in \{1, \ldots, Y\}$. Um classificador convencional é um mapeamento $\mathcal{F} : \mathbb{R}^K \rightarrow \{1 \ldots, Y\}$. Alguns classificadores são capazes de gerar valores de confiança $f_i(\mathbf{z})$ para cada classe $i = 1, \ldots, Y$, tais como a probabilidade de distribuição sobre y. Por conveniência, pode-se assumir que todo classificador retorna um vetor $\mathbf{y}$ com Y elementos. Se o classificador naturalmente não retornar valores de confiança, o vetor $\mathbf{y}$ apresenta um elemento igual a um para a classe correta $f_y(\mathbf{z}) = 1$, enquanto que outras são zero $f_i(\mathbf{z}) = 0, i \neq y$.

Assim, a decisão final pode sempre ser baseada no valor máximo dentre os elementos de $\mathbf{y}$ (chamada regra *max-wins*)

$$\mathcal{F}(\mathbf{z}) = \arg\max_i f_i(\mathbf{z}).$$

Para ilustrar de forma contextualizada a classificação convencional, é utilizado um pequeno trecho da base de dados gerada pelas simulações descritas no Capítulo 3. Nesse exemplo, a classificação depende dos quadros $\mathbf{F}$ gerados pelas concatenações das amostras de tensão e corrente analisadas. Esses quadros passam a ser os atributos de entrada do classificador, por exemplo árvore de decisão. A primeira etapa é treinar o classificador com um conjunto de dados rotulados com os 11 tipos de falta e, após treinamento, testar o classificador com um outro conjunto de dados não rotulados para medir o grau de confiabilidade deste, como mostra o esquema da Figura 2.1.

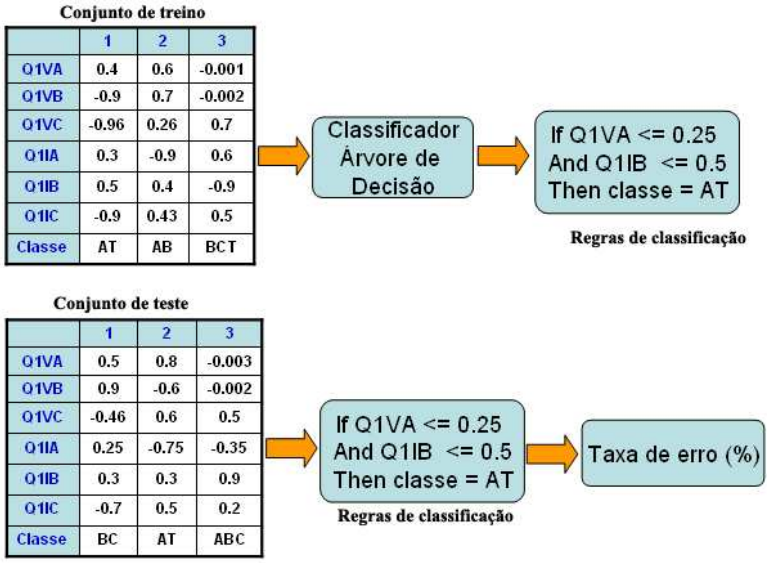

Figura 2.1: Classificação convencional considerando $L = 1$ e $K = 6$.

2.3.2 Classificação pós-falta

Alternativamente à classificação on-line, a classificação pós-falta [20, 29] lida diretamente com matrizes $\mathbf{Z}_n$ de dimensão variável $K \times N_n$ e pode ser feita em um centro supervisório em um estágio pós-falta. Os sistemas de classificação pós-falta buscam resolver problemas que podem ser tratados como problemas de *classificação de seqüências*.

Ao contrário do módulo on-line, no módulo pós-falta a classificação é feita sobre a seqüência de quadros. O classificador é então um mapeamento $\mathcal{G} : \mathbb{R}^{K \times N} \rightarrow \{1, \ldots, Y\}$ e o conjunto de treino $\{(\mathbf{Z}_1, y_1), \ldots, (\mathbf{Z}_M, y_M)\}$ contém M seqüências e seus respectivos rótulos.

Como será discutido nesse capítulo, ambas classificações on-line e pós-falta podem ser executadas a partir de classificadores convencionais. Assim, a próxima seção discutirá brevemente o software WEKA, utilizado nesse trabalho. As seções seguintes listarão os principais classificadores utilizados. Como cada um desses classificadores poderia render material para uma dissertação inteira, ao invés de discuti-los em profundidade, busca-se prioritariamente ilustrar como os mesmos são usados no WEKA.

2.4 Software WEKA para classificação convencional

O pacote WEKA (*Waikato Environment for Knowledge Analysis*) é uma ferramenta de aprendizado de máquina de domínio público (disponível em http://www.cs.waikato.ac.nz/ml/weka/) que foi desenvolvido na Universidade de Waikato na Nova Zelândia [9]. O sistema é formado por um conjunto de implementações de algoritmos de diversas técnicas de mineração de dados, e foi implementado na linguagem de programação Java, tornando-o acessível nas principais plataformas computacionais.

O Weka possui ferramentas para pré-processamento de dados, classificação, regressão, agrupamento, regras de associação e visualização. Atualmente está na versão 3.5.6, sendo organizado em três módulos de operação. O primeiro, "simple Command Line Interface" (CLI), a interação do usuário com Weka ocorre através de linhas de comando. O módulo "Explorer" executa a interface gráfica para execução dos algoritmos de aprendizagem de máquinas suportados pelo Weka. E o terceiro, é o módulo "Experimenter" no qual o usuário, também por meio de interface gráfica, executa testes estatísticos em diferentes algoritmos simultaneamente a fim de avaliar os resultados obtidos.

Antes de utilizar o pacote Weka, os dados devem ser convertidos para o formato arff (*Attribute-Relation File Format*), o qual é um arquivo ASCII composto de três partes. A primeira parte chamada de relação indicada pelo marcador @relation, que fica na primeira linha do arquivo, identifica o nome da relação. A segunda parte, iniciada sempre com o marcador @attribute, contém a lista de todos os atributos, onde se deve definir o tipo de atributo ou os valores que eles podem assumir, ao utilizar os valores estes devem estar entre "{ }" separados por vírgula. A terceira, encontra-se logo após a linha com o marcador @data e consiste das *instances*, isto é, os dados a serem minerados com o valor dos atributos para cada instância (linha). O formato arff não especifica qual atributo é a classe, pois isso permite que o mesmo seja mais flexível. Assim, o que o WEKA entende por *instance*, reúne o que a notação aqui adotada chama de *instance* $\mathbf{z}$ mais o rótulo y.

Vale ressaltar que o arff não dá suporte para séries temporais de duração variável, já

que exige que cada instância tenha um número pré-fixado de atributos.

2.5 Classificadores convencionais

O WEKA possui diversos algoritmos de aprendizagem e este trabalho usa apenas alguns dos principais, tais como árvores de decisão, redes neurais artificiais multicamadas treinadas com algoritmo backpropagation, naïve Bayes e K-nearest neighbor [9]. Um classificador de misturas de Gaussianas (GMM), o qual não faz parte do Weka, também foi usado. As próximas subseções discutem tais classificadores.

2.5.1 Redes neurais artificiais

As redes neurais artificiais (RNAs) são sistemas paralelamente distribuídos compostos por unidades simples de processamento denominados de neurônios artificiais que calculam uma certa função matemática (usualmente não-linear). Tais unidades são dispostas em uma ou mais camadas e interligadas pelos chamados pesos sinápticos. O comportamento inteligente de uma rede neural artificial vem das interações entre as unidades de processamento da rede. A Figura 2.2 exibe a estrutura de um neurônio artificial.

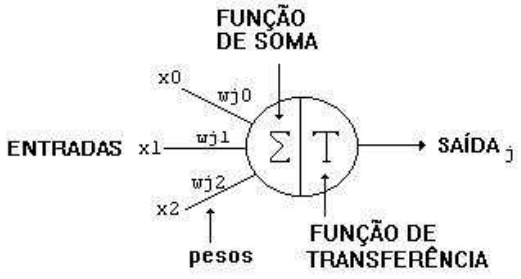

Figura 2.2: Estrutura de um neurônio artificial.

O ajuste de pesos é realizado em função de um cálculo que aponta a quantidade de erro do resultado (saída). Este ajuste procura corrigir os pesos de modo que se produza a saída desejada diante da respectiva entrada. Dentre os diversos tipos de cálculos para este fim, a Regra Delta é a mais utilizada.

Existem muitos tipos de algoritmos de aprendizado para redes neurais artificiais. Estes diferem entre si principalmente pelo modo como os pesos são modificados. Os mais conhecidos são o aprendizado supervisionado e o não supervisionado. No aprendizado supervisionado a rede neural recebe um conjunto de entradas e saídas de dados. Neste tipo de algoritmo, o

aprendizado ocorre por meio dos ajustes nos pesos, os quais são modificados até que os erros entre os padrões de saída gerados pela rede tenham um valor desejado ou próximo do desejado. Já no aprendizado não supervisionado, a rede neural trabalha os dados de forma a determinar algumas propriedades do conjunto de dados. A partir destas propriedades é que o aprendizado é constituído.

Denomina-se iteração ou época uma apresentação de todos os pares (entrada e saída) do conjunto de treinamento no processo de aprendizado. A correção dos pesos numa iteração pode ser executada de modo *standard* ou em *batch*. No modo *standard* o erro é estimado a cada apresentação de um conjunto de treino à rede. Enquanto que no modo *batch* estima-se o erro médio após todos os exemplos do conjunto de treinamento serem apresentados à rede.

Dentre os modelos de aprendizagem de uma rede neural temos o perceptron de múltiplas camadas (MLP), que consiste de um conjunto de nós fonte os quais formam a camada de entrada, uma ou mais camadas escondidas e uma camada de saída. Uma MLP é uma generalização do modelo perceptron, forma mais simples de uma rede neural usada para classificação de padrões. A Figura 2.3 exibe um rede neural do tipo perceptron de múltiplas camadas.

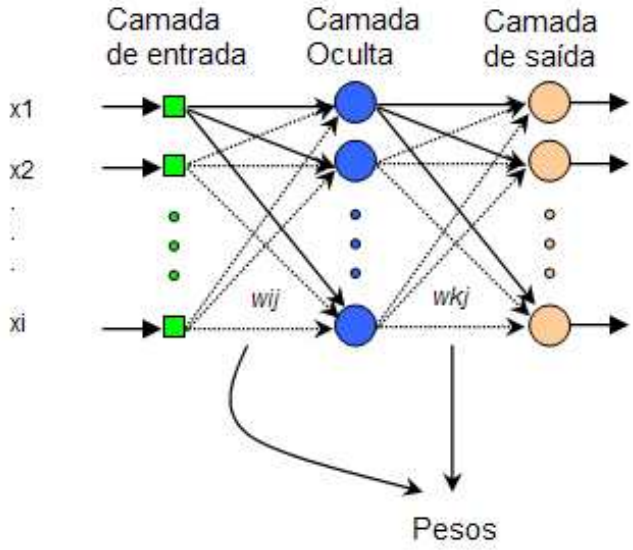

Figura 2.3: Exemplo de uma RNA MLP.

O número de neurônios na camada de entrada é determinado pela dimensão do espaço observado. Em contrapartida, a quantidade de neurônios na camada de saída é determinado pela dimensionalidade requerida pela resposta. Por exemplo, na classificação de faltas em linhas de transmissão o número de neurônios na camada de entrada depende do número de sinais Q e do tamanho do quadro L. Assim, se $Q = 6$ (3 sinais de tensão e 3 de corrente) e $L = 5$, o número de neurônios na camada de entrada é igual a 30 [30]. Quanto ao número

de neurônios na camada de saída pode-se ter duas representações: uma com 4 neurônios representando as fases A, B, C e T, onde, por exemplo, uma resposta da rede 1 1 0 0 equivale a uma falta do tipo AB [20, 21, 31] e outra com 11 neurônios (adotada neste trabalho) [6, 11]. Além disso, uma MLP leva em consideração os seguintes aspectos: determinação do número de camadas escondidas, de neurônios em cada uma destas camadas, assim como a especificação do tipo de algoritmo de aprendizado supervisionado utilizado.

O algoritmo utilizado neste trabalho para treinamento das redes neurais MLP foi o backpropagation também chamado de regra delta generalizada [32–34]. O algoritmo consiste em um processo de aprendizagem supervisionada que utiliza um conjunto pré-determinado de pares de exemplo de entrada e saída para ajustar os pesos da rede através de um esquema de correção de erros realizado em ciclos de propagação. O backpropagation é dividido em duas fases: a primeira fase consiste em propagar (*forward*) o vetor de entrada a partir da primeira até a última camada e comparar o valor da saída com o valor desejado. A segunda fase consiste em retropropagar (*backward*) o erro partindo da última camada até chegar a camada de entrada ajustando os pesos dos neurônios das camadas intermediárias. Após ajustar todos os pesos da rede, é apresentado mais um conjunto de exemplos encerrando uma época. Este processo é repetido até que o erro torna-se aceitável para o conjunto de treinamento, momento denominado de convergência da rede.

O comportamento de uma rede neural MLP durante seu treinamento varia mediante a alteração de algumas de suas características [33]:

- Inicialização dos pesos: Os pesos das conexões entre os neurônios podem ser inicializados uniformemente ou de forma aleatória.
- Taxa de aprendizado: A taxa de aprendizado controla a velocidade do aprendizado, aumentando ou diminuindo o ajuste de pesos que é efetuado a cada iteração durante o treinamento. Intuitivamente, seu valor deve ser maior que 0 e menor que 1. Se a taxa de aprendizado for muito pequena, o aprendizado ocorrerá muito lentamente. Caso a taxa seja muito grande (maior que 1), a correção seria maior do que o erro observado, fazendo com que a rede neural ultrapassasse o ponto de aprendizado ótimo, tornando o processo de treinamento instável.
- Parametrização da função de transferência: Também conhecida como limiar lógico, essa função é quem define e envia para fora do neurônio o valor passado pela função de ativação. A função de ativação pode ter muitas formas e métodos. As mais conhecidas são: função linear, função sigmóide e função exponencial.

Neste trabalho foi utilizada a classe FastNeuralNetwork, que é uma implementação da

rede neural MLP baseada na implementação do WEKA. O código fonte e uma descrição geral desta classe são disponibilizados em: http://www.laps.ufpa.br/aldebaro/weka/.

A classe FastNeuralNetwork é mais rápida que a rede neural MLP implementada no WEKA. Porém, ambas as implementações possuem os mesmos parâmetros e fornecem os mesmos resultados caso os dados atendam às restrições extras impostas pela FastNeuralNetwork. Por esse motivo, qualquer uma das duas implementações pode ser usada nas simulações descritas no Capítulo 4. Os principais parâmetros das duas implementações são:

- -L: Corresponde à taxa de aprendizado utilizada pelo algoritmo *backpropagation*. Este valor deve ser entre 0 e 1 (Padrão é 0.3).
- -M: Taxa de momento para o algoritmo *backpropagation*. Este valor deve ser entre 0 e 1 (Padrão é 0.2).
- -N: Este parâmetro corresponde ao número de épocas para treinamento da rede. O Padrão é 500.
- -H: Corresponde à quantidade de camadas ocultas que podem ser criadas na rede. Por exemplo -H 3, 2 cria duas camadas intermediárias com 3 e 2 neurônios respectivamente. Outra forma de representar pode ser através do uso de letras: a opção a corresponde a (números de atributos + número de classes)/2; as outras opções são: i (número de atributos), o (número de classes) e, t (números de atributos + número de classes).

2.5.2 Árvores de decisão

Árvores de decisão (Figura 2.4) constituem-se de nós internos e externos, hierárquicos conectados por ligações ou ramos. O nó interno, também chamado de nó decisão, é a unidade tomadora de decisão que avalia através de teste lógico qual será o próximo nó descendente ou filho. Em contrapartida, um nó externo (sem nó descendente), também conhecido como folha ou nó terminal, é associado a uma classe ou um rótulo. Em geral, um conjunto de dados é apresentado ao nó inicial ou raiz (também um nó interno) da árvore; baseado na resposta do teste lógico feito pelo nó, a árvore ramifica-se para um dos nós filhos e este procedimento é repetido até que um nó terminal é alcançado, o qual não realiza mais nenhum teste lógico. Cada nó folha carrega um rótulo, e o padrão de teste é associado à classe do nó folha alcançado.

Uma árvore de decisão é formada por um conjunto de regras de classificação. Cada caminho da raiz até uma folha representa uma destas regras. A árvore de decisão deve ser definida de forma que, para cada observação da base de dados, haja apenas um caminho da

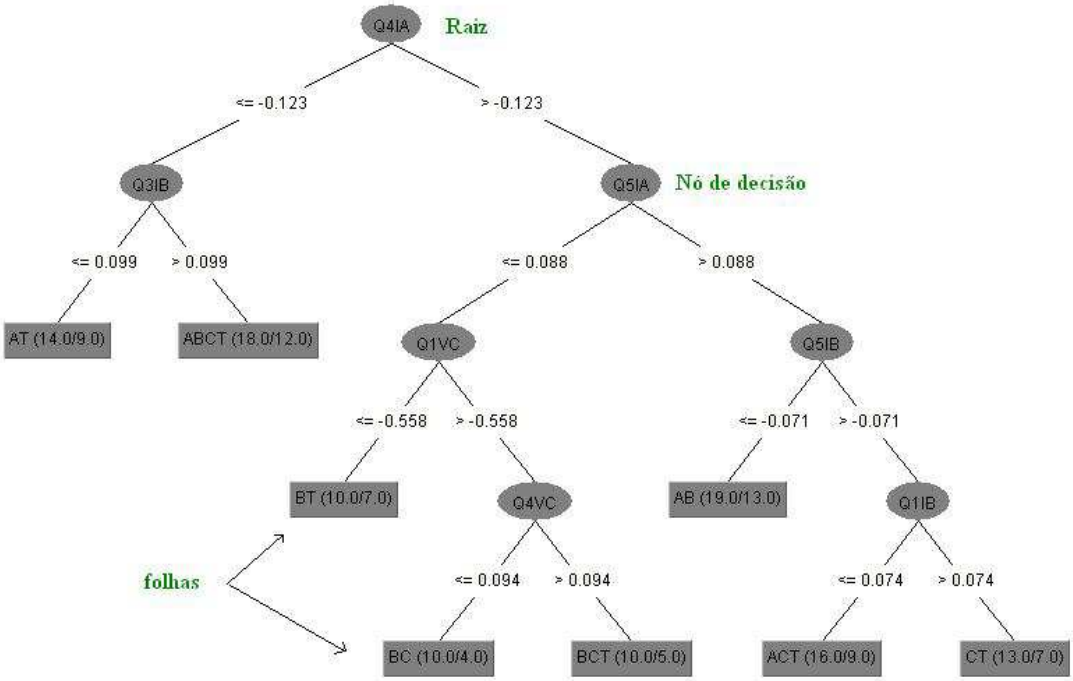

Figura 2.4: Exemplo de árvore de decisão.

raiz até a folha. Regras de classificação são compostas por um antecedente (pré-condição) e um conseqüente (conclusão). Um antecedente deve ser formado por um ou mais atributos preditivos enquanto que o conseqüente define a classe ou classes. Por exemplo, na Figura 2.4 temos o seguinte exemplo de regra: if Q4IA <= -0.123 and Q3IB <= 0.099 then classe=AT.

Uma questão chave para a construção de uma árvore de decisão consiste na estratégia para a escolha dos atributos que possam determinar a classe a qual uma *instance* pertence. Observe que na Figura 2.4, o atributo Q4IA encontra-se na raiz da árvore, pois foi considerado pelo algoritmo classificador como o atributo mais importante para determinar o tipo de falta (curto-circuito) em linha de transmissão. Geralmente são utilizadas medidas baseadas na entropia para tratar este problema. Este tipo de medida mede a aleatoriedade do valor de um atributo antes de decidir qual atributo usar para prever a classe. Quanto maior a entropia, maior a aleatoriedade dos valores que os atributos podem assumir. Esse critério de seleção de atributo elege o atributo que gerará partições com menor entropia, ou seja, a construção de uma árvore de decisão é guiada no sentido de diminuir a entropia, ou seja, a dificuldade de previsão da variável objetivo.

Como árvores de decisão são métodos que utilizam um algoritmo recursivo para sucessivas divisões em um conjunto de treino. O principal problema consiste então na confiabilidade das estimativas do erro usado para selecionar as divisões. Apesar da estimativa obtida com

os dados de treinamento usado durante o crescimento da árvore conhecida como "erro de re-substituição" continuar decrescendo, geralmente, as escolhas da divisão em níveis maiores da árvore produzem estatísticas não muito confiáveis. Portanto, a qualidade da amostra influencia diretamente na precisão das estimativas do erro. Como a cada iteração o algoritmo divide o conjunto de dados de treinamento, os nós internos tomam decisões a partir de amostras cada vez menores. Isto significa que as estimativas de erro têm menos confiabilidade à medida que crescemos a árvore. Para minimizar este problema e evitar o *overfittting* dos dados de treinamento com árvores muito complexas, têm-se estratégias conhecidas como métodos de podagem [9, 32].

Basicamente, há duas classes de métodos para podagem da árvore de decisão: a pós-podagem e a pré-podagem. A pós-podagem consiste em permitir que a árvore cresça até um tamanho máximo, isto é, até que os nós folhas tenham mínimo de impureza, para posterior aplicação da podagem. Esse é o método usado pelo algoritmo C4.5, o qual é provavelmente o algoritmo mais famoso para projeto de árvores e foi desenvolvido por J. Quinlan [35]. WEKA tem uma implementação do C4.5 a qual é chamada J4.8.

A árvore J4.8 constrói um modelo de árvore de decisão baseado num conjunto de dados de treinamento e usa esse modelo para classificar as instâncias do conjunto de teste. A seguir são apresentados os principais parâmetros desse classificador implementado no WEKA.

- -U: não utiliza poda da árvore e a mesma cresce até seu tamanho máximo
- -C: Corresponde ao fator limiar de confiança de poda. O padrão é 0.25.
- -M: indica o número mínimo de exemplos por folha (padrão 2).
- -R: usa o método de erro de redução da poda [35].
- -N: indica o número de campos para o erro de redução de poda. Um campo é usado como conjunto de poda (padrão 3).

No que diz respeito ao custo computacional, nota-se inicialmente que uma árvore de decisão binária *completa* (ou *perfeita*) apresenta as seguintes propriedades: os nós têm dois descendentes (os chamados nós internos) ou nenhum (chamados folhas) e todas as folhas estão no mesmo nível $d_{\max}$ (ou *depth*). Uma árvore binária completa tem $2^{d_{\max}} - 1$ nós e pode-se encontrar uma folha após $d_{\max} - 1$ comparações. Uma árvore de decisão J4.8 (ou C4.5) geralmente não é completa, porém seu custo no pior caso é dado por $d_{\max} - 1$ comparações, onde $d_{\max}$ é o nível mais profundo dentre todas folhas.

2.5.3 K-Nearest Neighbors (KNN)

Os classificadores vistos anteriormente são caracterizados pelo fato de utilizarem os dados de treinamento para construírem um modelo de classificação, o qual, uma vez encontrado e testado, estará pronto para testar qualquer padrão novo. Diferentemente desses classificadores, o classificador K-Nearest Neighbors (K-vizinhos mais próximos) utiliza os próprios dados de treinamento como modelo de classificação, isto é, para cada novo padrão que se quer classificar, utiliza-se os dados do treinamento para verificar quais são os exemplos nessa base de dados que são "mais próximos" do padrão em análise. A cada novo padrão a ser classificado faz-se uma varredura nos dados de treinamento, o que provoca um grande esforço computacional.

Suponhamos um conjunto de treinamento com M exemplos. Seja $\mathbf{z} = (z_1, \ldots, z_K)$ uma nova *instance*, ainda não classificada. A fim de classificá-la, calcula-se as distâncias, através de uma medida de similaridade, entre $\mathbf{z}$ e todos os exemplos do conjunto de treinamento e considera-se os K exemplos mais próximos (com menores distâncias) em relação $\mathbf{z}$. Verifica-se então, qual a classe que aparece com mais freqüência, entre os K vizinhos encontrados. O padrão $\mathbf{z}$ será classificado de acordo com a classe y mais freqüente dentre os K exemplos encontrados.

A distância entre duas *instances* é calculada utilizando-se uma medida de similaridade. Uma medida de similaridade bastante popular é a *distância euclidiana* [9, 32]. Tal medida calcula a raiz quadrada da norma do vetor diferença entre os vetores $\mathbf{z}$ e $\hat{\mathbf{z}}$:

$$d(\mathbf{z}, \hat{\mathbf{z}}) = \sqrt{\sum_{i=1}^{K} (z_i - \hat{z}_i)^2} \tag{2.2}$$

Nesse trabalho, o algoritmo de agrupamento K-means [32] foi utilizado com objetivo de reduzir o tamanho da base de dados de treino, e com isso diminuir o custo computacional do KNN. Este algoritmo consiste em encontrar K centros que melhor representam os dados do conjunto de treino. Não se deve confundir o K do K-means com o K do KNN. Suponhamos que o tamanho do conjunto de treino seja $M = 1000$, aplicando-se o algoritmo K-means com K=250 centros, o KNN usará uma nova base de treino com $M' = 250$. Ainda nesse exemplo, seria possível usar K=1 ou K=5 para o KNN, por exemplo.

Assim como no KNN, o cálculo dos centros do K-means depende da medida de similaridade. A distância euclidiana foi utilizada para ambos KNN e K-means.

O KNN no WEKA é implementado na classe IBK e seus principais parâmetros são:

- -N : número de centros (ou K).

- -S: esta opção gera aleatoriamente os centros.

Nota-se que o princípio do KNN pode ser usada para classificação de seqüências, mas nesse caso a medida de similaridade não pode ser a distância euclidiana, visto que as seqüências podem não ter a mesma dimensão. Nesses casos, a medida de similaridade pode ser baseada em *dynamic time warping* (DTW) e o algoritmo é chamado na literatura de reconhecimento de voz como *segmental K-means* [36].

2.5.4 Naïve Bayes

A técnica do classificador naïve Bayes é baseada no chamado teorema de Bayes e esse classificador é particularmente utilizado quando a dimensionalidade dos dados de entrada é grande. Assim, para representar esse classificador na classificação de faltas, com base em [32], adotamos a seguinte nomenclatura[1]: $P(y|\mathbf{z})$, $P(\mathbf{z}|y)$, $P(y)$ e $P(\mathbf{z})$ são chamados, respectivamente, de *a posteriori, likelihood, a priori e evidência*, e são expressas através do teorema de Bayes

$$P(y|\mathbf{z}) = \frac{P(\mathbf{z}|y)P(y)}{P(\mathbf{z})}. \tag{2.3}$$

Este tipo de classificador tenta selecionar o rótulo

$$\mathcal{F}(\mathbf{z}) = \arg \max_{y=1...,Y} P(\mathbf{z}|y)P(y), \tag{2.4}$$

que maximiza a probabilidade a posteriori. Entretanto, $P(y)$ e $P(\mathbf{z}|y)$ não são conhecidas. Conseqüentemente, os classificadores usam estimativas $\hat{P}(y) e \hat{P}(\mathbf{z}|y)$ e maximizam

$$\mathcal{F}(\mathbf{z}) = \arg \max_{y=1,...,Y} \hat{P}(\mathbf{z}|y)\hat{P}(y). \tag{2.5}$$

Na maioria dos casos, a priori $P(y)$ pode ser estimada com precisão contando-se as classes no conjunto de treino, isto é, pode-se assumir que $\hat{P}(y) = P(y)$. Estimar $\hat{P}(\mathbf{z}|y)$ é freqüentemente a tarefa mais difícil. Assim, classificadores Bayesianos assumem tipicamente uma distribuição paramétrica $\hat{P}(\mathbf{z}|y) = \hat{P}_{\theta_y}(\mathbf{z}|y)$ onde θ_y descreve os parâmetros da distribuição que precisam ser determinados (por exemplo, a média e a matriz de covariância se o modelo de likelihood é uma distribuição Gaussiana).

O algoritmo naïve Bayes assume que os atributos $(z_1, \ldots, z_K)$ de $\mathbf{z}$ são todos condicionalmente independentes um do outro, dado y. Isto significa que o algoritmo simplifica

[1] P denota as funções massa de probabilidade e densidade de probabilidade

drasticamente a representação de $P(\mathbf{z}|y)$, e o problema da estimação deste a partir do conjunto de treino. Considerando, por exemplo, o caso onde $\mathbf{z} = (z_1, z_2)$, tem-se:

$$P(\mathbf{z}|y) = P(z_1, z_2|y) = P(z_1|z_2, y)P(z_2|y) = P(z_1|y)P(z_2|y) \tag{2.6}$$

onde $P(z_1, z_2|y) = P(z_1|z_2, y)P(z_2|y)$ é uma propriedade geral proveniente da definição de probabilidade condicional, enquanto $P(z_1, z_2|y) = P(z_1|y)P(z_2|y)$ só é válida por se assumir independência condicional. Generalizando a Equação (2.6) para K atributos condicionalmente independentes, tem-se

$$P(\mathbf{z}|y) = P(z_1, ..., z_K|y) = \prod_{i=1}^{K} P(z_i|y). \tag{2.7}$$

Ao treinar um classificador naïve Bayes, este produzirá uma distribuição de probabilidade $P(z_i|y)$ e $P(y)$ para todos os possíveis valores de y, ou seja, $y_k, k = 1, \ldots, Y$. Para calcular a probabilidade posterior de cada classe y, usa-se o teorema de Bayes:

$$p(y_k|\mathbf{z}) = \frac{P(y_k)P(z_1, \ldots, z_K|y_k)}{\sum_j P(y_j)P(z_1, \ldots, z_K|y_j)} \tag{2.8}$$

onde o somatório no denominador é realizado sobre todos os possíveis valores y_k de y, de maneira a normalizar e se ter uma probabilidade. Assumindo-se que z_i é condicionalmente independente dado y, pode-se reescrever a Equação (2.8) a partir da Equação (2.7) como:

$$p(y_k|\mathbf{z}) = \frac{P(y_k)\prod_i P(z_i|y_k)}{\sum_j \left(P(y_j)\prod_i P(z_i|y_j)\right)}. \tag{2.9}$$

A Equação (2.9) é a equação fundamental do classificador naïve Bayes. Dado um novo exemplo $\mathbf{z}$, esta equação mostra como calcular a probabilidade para cada y. Tal cálculo depende apenas dos valores dos atributos observados do novo exemplo e das distribuições $P(y)$ e $P(z_i|y)$ estimadas a partir dos dados de treino. Se deseja-se apenas o valor mais provável de y então se pode simplificar para:

$$\mathcal{F}(\mathbf{z}) = \arg\max_{y_k} P(y_k) \prod_i P(z_i|y_k) \tag{2.10}$$

já que o denominador não depende de y_k. Ou ainda, usando-se o fato do logaritmo ser uma função monotônica:

$$\mathcal{F}(\mathbf{z}) = \arg\max_{y_k} \left[\log P(y_k) + \sum_i \log P(z_i|y_k)\right]. \tag{2.11}$$

No WEKA, o naïve Bayes usa por padrão a distribuição normal quando os atributos são numéricos. Existe a opção -K para se usar estimadores de densidade. Em contraste, a opção -D usa discretização supervisionada para primeiro transformar os atributos numéricos em discretos e daí usar funções massa de probabilidade.

2.5.5 Misturas de Gaussianas

Um caso particular de classificador de Bayes é obtido quando a likelihood é modelada utilizando uma Gaussiana,

$$\hat{P}(\mathbf{z}|y) = \mathcal{N}(\mathbf{z}|y, \mu, \mathbf{\Sigma}) = \frac{1}{\sqrt{(2\pi)^K |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{z}-\mu_y)'\mathbf{\Sigma^{-1}}(\mathbf{z}-\mu_y)\right), \tag{2.12}$$

ou uma mistura de G_y Gaussianas,

$$\hat{P}(\mathbf{z}|y) = \sum_{g=1}^{G_y} \omega_{yg}\mathcal{N}(\mathbf{z}|y, \mu_{yg}, \mathbf{\Sigma}_{yg}), \tag{2.13}$$

onde K é a dimensionalidade do vetor $\mathbf{z}$ e $\mathbf{\Sigma}$ representa uma matriz de covariância. Nota-se que o número G_y de Gaussianas por classe pode variar para cada classe y.

A partir de tais equações diversos algoritmos de reconhecimento de padrões bem conhecidos na literatura podem ser obtidos. Um caso de interesse para este trabalho é obtido quando a matriz de covariância $\mathbf{\Sigma}$ é diagonal e individual para cada Gaussiana pertencente à mistura. Neste caso o classificador é chamado de *Gaussians Mixture Model* (GMM) [37]. A GMM, assim como muitos classificadores de Bayes, utiliza o MLE (*Maximum Likelihood Estimation*) para encontrar os parâmetros Θ (médias e matrizes de covariância) das misturas.

Durante o treinamento, assume-se que os vetores $\mathbf{z}$ de entrada são independentes e identicamente distribuídos, e busca-se via MLE:

$$\Theta^g = \arg\max_{\Theta} R^g(\Theta),$$

onde

$$R^g(\Theta) = \prod_{n=1}^{M} \hat{P}(\mathbf{z}_n/y_n).$$

Normalmente estes parâmetros são estimados através de algoritmos como o EM (*Expectation Maximization*) [38].

O algoritmo GMM (veja [37] para mais detalhes) não faz parte do pacote WEKA e foi desenvolvido em Java, com a classe sendo denominada de GaussiansMixture. O código fonte desta classe e uma descrição de como adicioná-la ao WEKA podem ser obtidos em *http://www.laps.ufpa.br/freedatasets/ufpafaults/scripts*. Os principais parâmetros da classe GaussiansMixture são:

- -I : Número de Gaussianas (padrão igual 1).
- -N: número máximo de iterações (padrão é 20).
- -C: valor mínimo para os elementos da matriz de covariância (padrão é 1e-4).

2.6 Classificadores de seqüências

Para classificação de seqüências pode-se adotar uma arquitetura baseada em quadros (FBSC) ou aplicar técnicas para implementar $\mathcal{G}$, que lidam diretamente com as seqüências, tais como *Hidden Markov Models* (HMM) [38] e *Dynamic Time-Warping* (DTW) [39]. As próximas subseções discutem essas alternativas.

2.6.1 Classificação de seqüências baseada em quadros

Quando se adota a classificação de seqüências baseada em quadros, como mostra o esquema da Figura 2.5, o módulo de falta invoca repetidamente um classificador convencional $\mathcal{F}$ (rede neural, árvore de decisão, etc.) para obter scores $\mathbf{y} = (f_1(\mathbf{z}), \ldots, f_Y(\mathbf{z}))$ para cada classe. Para chegar a uma decisão final, o módulo de falta pode então considerar os scores de todos os N quadros. Por exemplo, o módulo pode calcular um score acumulado $g_i(\mathbf{Z})$ para cada classe e então usar a regra

$$G(\mathbf{Z}) = \arg\max_i g_i(\mathbf{Z}) \tag{2.14}$$

onde possíveis alternativas são:

$$g_i(\mathbf{Z}) = \sum_{n=1}^{N} f_i(\mathbf{z}_n) \tag{2.15}$$

ou

$$g_i(\mathbf{Z}) = \sum_{n=1}^{N} \log(f_i(\mathbf{z}_n)). \tag{2.16}$$

Outra alternativa seria apenas contar qual a classe mais frequênte dentre as N decisões.

Em FBSC, o desempenho de $\mathcal{G}(\mathbf{Z})$ depende primordialmente do classificador $\mathcal{F}(\mathbf{z})$. Tais desempenhos podem ser avaliados de acordo com as **taxas de erro de classificação** E_s e E_f, para seqüências (usada em pós-falta) e quadro-a-quadro (*on-line*), respectivamente.

Faltas do tipo curto-circuito em linhas de transmissão são processos dependentes do tempo. Tais faltas apresentam diferentes durações, freqüências e magnitudes. Além disso, o mesmo tipo de falta com mesma duração tende a variar suas características de freqüência e magnitude, devido à natureza randômica associada com a falta. Para obter uma medida de distância global entre dois padrões de falta, um alinhamento temporal dinâmico ou *Dynamic Time Warping* (DTW) [39] pode ser realizado ou mesmo adotar-se HMMs. Esse é o assunto das próximas subseções, as quais representam alternativas à arquitetura FBSC.

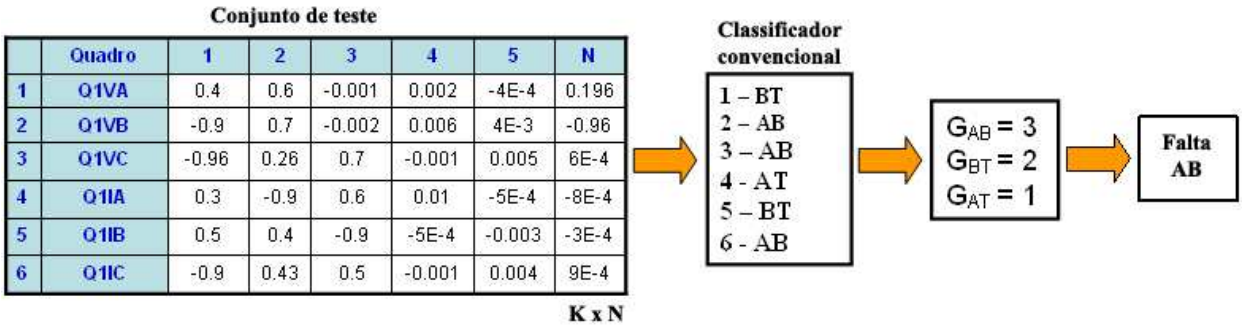

Figura 2.5: Esquema da classificação de seqüências utilizando arquitetura FBSC adotando-se $L = 1$ ($K = 6$).

2.6.2 *Hidden Markov Models*

Hidden Markov Models (HMM) ou cadeias escondidas de Markov é a técnica predominante em reconhecimento de voz [38] e foi aplicada ao problema de classificação de faltas em [40], onde um modelo HMM discreto foi utilizado.

No modelo de HMM discreto é definido um conjunto de estados e alfabetos de símbolos de saída. Cada estado é caracterizado por duas distribuições de probabilidade: a distribuição de transição entre os estados e distribuição de observações de saída.

Uma fonte aleatória descrita por um modelo de HMM discreto gera uma sucessão de símbolos discretos de saída. A cada intervalo de tempo a fonte está em um estado, e depois de emitir um símbolo de saída de acordo com a distribuição de observação, a fonte vai para um próximo estado de acordo com a distribuição de transição do seu estado atual. Como a atividade da fonte é observada indiretamente, através da seqüência dos símbolos de saída, e a seqüência de estados não é diretamente observável, a cadeia de estados fica oculta, originando o termo *hidden* (escondida).

Ao contrário do modelo discreto, os símbolos de saída do HMM contínuo são emitidos a partir de uma função densidade de probabilidade (geralmente Gaussiana) ao invés de uma função massa de probabilidade. Este trabalho adota como modelo o HMM contínuo.

A topologia de HMM mais utilizada para modelar séries temporais, é a "esquerda-direita". Nesta topologia cada estado, exceto o último, possui apenas duas transições possíveis: permanecer no estado corrente ou mover-se para o próximo estado. Tal arquitetura é apresentada na Figura 2.6, onde a indica a distribuição de probabilidade de transição de estados, b_i indica densidade de probabilidade dos símbolos observados no estado corrente i.

Os parâmetros de um HMM contínuo com S estados podem ser representado por $\lambda =$

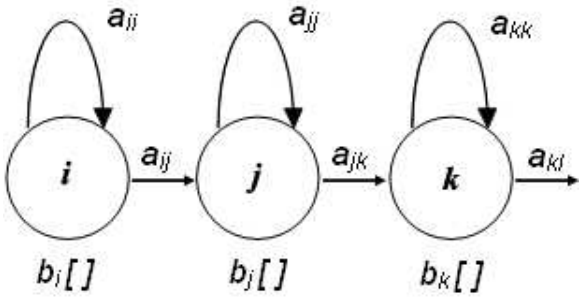

Figura 2.6: Estrutura de um HMM com três estados para arquitetura esquerda-direita.

(A, B, π), onde A é a matriz de transição de estados, B é o conjunto de funções de densidade de probabilidade de observação de **S** (tipicamente Gaussianas multivariadas ou misturas delas) e π é um vetor de dimensão S, composto por uma função massa de probabilidade dos estados iniciais. O elemento a_{ij} de A é a probabilidade de realizar uma transição do estado i para j, com $\sum_{i=1,\ldots,S} a_{ij} = 1, \forall_i$. Em algumas modelagens (no software HTK [2], a ser discutido, por exemplo), o vetor π é incorporado em uma matriz de transição estendida $\hat{A}$, a partir da definição de dois estados que não emitem um símbolo de saída (entrada e saída) por HMM como mostra a Figura 2.7. Assim, os parâmetros do HMM passam a ser $\lambda = (\hat{A}, B)$, com $\hat{A}$ sendo uma matriz $(S+2) \times (S+2)$ na qual a primeira coluna e a última linha não são utilizadas.

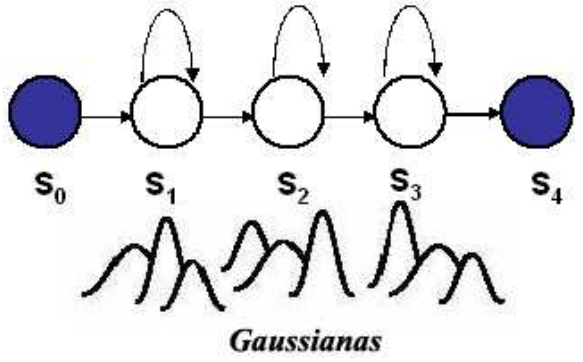

Figura 2.7: Modelo de HMM contínuo adotado no software HTK com três estados "emissores" e dois estados extras. A distribuição de saída de cada estado é uma mistura de três Gaussianas.

Para o problema de classificação de faltas em linhas de transmissão, os modelos de HMM contínuos são aplicados da seguinte maneira: um modelo HMM é treinado para cada tipo de falta através do algoritmo Baum-Welch, o qual é uma eficiente implementação da estimação de máxima verossimilhança (MLE) e um caso especial do algoritmo *expectation-maximization* (EM) [38]. Adota-se nesse caso o paradigma de aprendizagem *geracional* ao invés do *discriminativo* e, consequentemente, o conjunto de treino de um dado HMM é composto

apenas pelos exemplos da correspondente falta. Tal paradigma permite o tratamento mais rápido e eficaz da grande quantidade de dados.

Após o treinamento, durante o estágio de teste, dado um evento $\mathbf{Z}$ a ser classificado, o algoritmo de Viterbi [38] é executado para cada modelo HMM i para estimar as likelihoods $P(\mathbf{Z}|y_i), i = 1, \ldots, Y$, onde Y é o número de modelos de HMM (ou faltas, assim $Y = 11$ neste trabalho, como citado). A probabilidade a priori $P(y_i)$ é também usada e o sistema calcula a decisão da máxima posteriori (MAP) através da regra de Bayes:

$$\mathcal{G}(\mathbf{Z}) = \arg \max_{i=1,\ldots,Y} P(y_i|\mathbf{Z}) = \arg \max_{i=1,\ldots,Y} \frac{P(\mathbf{Z}|y_i)P(y_i)}{P(\mathbf{Z})} = \arg \max_{i=1,\ldots,Y} P(\mathbf{Z}|y_i)P(y_i) \qquad (2.17)$$

Nesse trabalho, como citado, para o desenvolvimento e manipulação de HMMs foi utilizado um conjunto de aplicativos (*toolkit*) denominado de ***Hidden Markov Model Toolkit (HTK)***[2] [2]. O HTK pode ser usado para modelar qualquer tipo de série temporal. Entretanto, o HTK foi inicialmente projetado para construção de HMMs a serem usados em reconhecimento de voz. Por isso, grande parte da infra-estrutura de suporte no HTK é dedicada a esta tarefa.

Como mostra a Figura 2.8, existem dois principais estágios de processamento envolvidos no HTK. Primeiramente, as ferramentas de treinamento do HTK são usadas para estimar os parâmetros de um conjunto de HMMs usando suas "transcrições" (rótulos) associadas. No segundo estágio, os eventos desconhecidos são transcritos usando as ferramentas de reconhecimento do HTK e comparados com a transcrição correta.

Além do projeto de HMMs, o HTK possui rotinas para a análise do sinal e também para o cálculo de desempenho. A Figura 2.9 mostra os principais estágios de processamento deste software, juntamente com suas funções.

O primeiro estágio, o de preparação de dados, consiste na conversão e concatenação dos dados de entrada para um formato aceito pelo HTK. São exemplos de algumas das principais funções usadas nessa fase:

- HCopy: é usado para copiar um ou mais arquivos de origem para um arquivo de saída. Normalmente, HCopy copia todo o arquivo, porém uma variedade de mecanismos são fornecidos para extração de segmentos e concatenação dos arquivos;
- HList: pode ser usado para checar o resultado de todas conversões do HCopy, o que é útil antes de se processar uma grande quantidade de dados;

[2]Mais informações sobre o HTK podem ser encontradas em http://htk.eng.cam.ac.uk/.

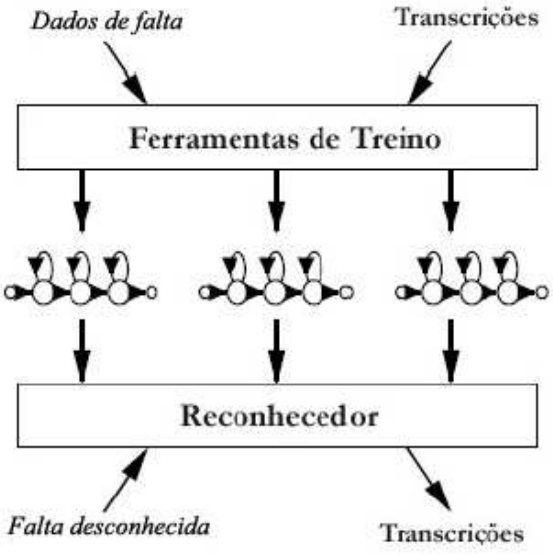

Figura 2.8: Fundamentos do HTK.

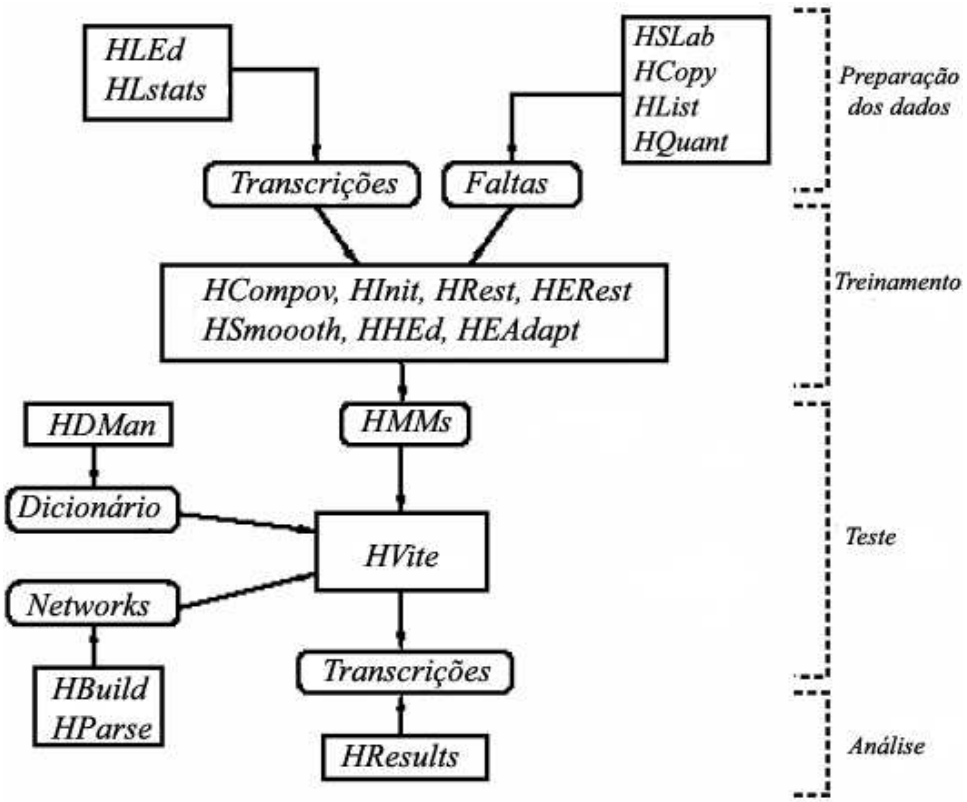

Figura 2.9: Estágios de processamento do HTK (vide [2] para maiores detalhes).

Neste trabalho, o HCopy não foi utilizado na conversão dos arquivos de entrada para o formato apropriado. Esta tarefa foi realizada sem a ajuda do HTK pois a maioria dos tipos de parâmetros disponíveis no HCopy é específica para o sinal de voz.

O estágio de treinamento inicia-se por definir a topologia necessária para cada HMM (número de estados e suas conexões / transições). Esta topologia é descrita por arquivos textos denominados de protótipos permitindo a edição destes em qualquer editor de texto. Alternativamente, a distribuição HTK padrão inclui exemplos de protótipos de HMM e um script para gerar topologias automaticamente.

Após a definição dos protótipos, realiza-se a inicialização dos modelos HMM. A ferramenta HCompV pode ser usada para calcular a média (eventualmente um vetor) e variância (eventualmente uma matriz de covariância) global do conjunto de treino, e usar tais valores como estimativa inicial para cada estado da HMM. A seguir, faz-se a reestimação dos modelos.

Considera-se aqui que se possui o conhecimento dos *end points* (início e fim) dos eventos nas formas de onda, ou seja, há transcrições que indicam o início e fim de cada falta. Nesse caso (há outro programa de reestimação chamado HERest caso os tempos de ocorrência dos eventos não tenham sido anotados), após a inicialização dos modelos HMM, há duas ferramentas do HTK aptas a reestimá-los: o HInit e o HRest. O utilitário HInit utiliza o algoritmo segmental k-means [36] e o HRest utiliza o algoritmo Baum-Welch [41].

Para a execução de testes dos modelos, após o treinamento, usa-se a ferramenta HVite. O HVite implementa o algoritmo Viterbi [38] para encontrar o modelo HMM de maior probabilidade dada uma seqüência de teste. O HVite gera um arquivo com as transcrições como resultado, as quais indicam os eventos encontrados e seus instantes de início e fim. É o utilitário HResults que busca então alinhar as transcrições corretas e as transições encontradas pelo HVite, calculando então o número de erros de substituição, exclusão e inserção. O HTK assume que mais de um evento pode ser encontrado em uma dada seqüência. Todavia, nesse trabalho, informou-se ao HTK (através de uma gramática) que havia um único evento por seqüência. Daí, não há erros de inserção e exclusão, apenas de substituição.

A criação de sistemas usando o HTK permite que as HMMs sejam refinadas gradualmente, ou seja, inicia-se com modelos simples contendo uma única Gaussiana por estado, para posteriormente realizar uma sofisticação gradual de tais modelos, aumentando-se o número de Gaussianas.

2.6.3 *Dynamic Time Warping* (DTW)

De forma similar a HMMs, a técnica DTW permite comparar seqüências de durações distintas. DTW é uma medida de similaridade fundamentada na programação dinâmica e em modelos de similaridade. A programação dinâmica é um método de decomposição que se baseia no princípio da otimalidade de Bellman [42], aplicável em situações nas quais não é

fácil chegar a uma seqüência ótima de decisões sem testar todas as seqüências possíveis para então escolher a melhor. A idéia básica da programação dinâmica é construir por etapas uma resposta ótima combinando respostas já obtidas em partes menores.

Mais especificamente, o alinhamento, usando DTW, entre duas séries temporais de faltas $\mathbf{Z} = (\mathbf{z}_1, ..., \mathbf{z}_p)$ e $\hat{\mathbf{Z}} = (\hat{\mathbf{z}}_1, ..., \hat{\mathbf{z}}_q)$ consiste em gerar uma matriz $\mathbf{D}$ de dimensão $p \times q$ cujo elemento (i, j) contém a distância euclidiana $d(\mathbf{z}_i, \hat{\mathbf{z}}_j)$ (outras medidas de similaridade podem ser adotadas) entre os vetores $\mathbf{z}_i$ e $\hat{\mathbf{z}}_j$. Um percurso de ajuste W, é um conjunto de elementos contínuos da matriz que define um mapeamento entre os vetores de $\mathbf{Z}$ e $\hat{\mathbf{Z}}$. O k-ésimo elemento de W é definido como $w_k = (i, j)_k$. Portanto

$$W = w_1, w_2, ..., w_{|W|}, \ \max(q, p) \leq |W| < p + q - 1, \tag{2.18}$$

onde $|W|$ é a cardinalidade (número de elementos) de W. O elemento $\mathbf{D}(w_k)$ corresponde à distância euclidiana respectiva ao elemento w_k da matriz.

O percurso de ajuste deve satisfazer às seguintes condições [43]:

- Iniciar e terminar em células diagonalmente opostas da matriz, ou seja, $w_1 = 1, 1$ e $w_{|W|} = (p, q)$
- Dado $w_k = (a, b)$ então $w_{k-1} = (a', b')$ onde $a - a' \leq 1$ e $b - b' \leq 1$. Isto quer dizer que a seqüência entre os elementos do percurso deverá conter elementos adjacentes da matriz (incluindo elementos diagonalmente posicionados).
- Dado $w_k = (a, b)$ então $w_{k-1} = (a', b')$ onde $a - a' > 0$ e $b - b' > 0$. Isto indica que a seqüência não poderá "voltar" no caminho dos elementos da matriz.

Dos muitos percursos de ajuste que satisfazem às condições acima, o percurso escolhido é aquele que minimiza o custo da deformação:

$$DTW(\mathbf{Z}, \hat{\mathbf{Z}}) = \min_{W} \sum_{k=1}^{|W|} \frac{\mathbf{D}(w_k)}{|W|} \tag{2.19}$$

onde $|W|$ é usado para compensar o fato de que o percurso de ajuste tenha diferentes tamanhos.

Este percurso pode ser encontrado de modo eficiente usando a programação dinâmica para avaliar a recursão sucessiva que define a distância acumulativa $\gamma(i, j)$. Um possível exemplo de cálculo de $\gamma(i, j)$, o qual foi usado nesse trabalho, é (vide [38] para outras opções):

$$\gamma(i, j) = \mathbf{D}(i, j) + \min\{\gamma(i - 1, j - 1), \gamma(i - 1, j), \gamma(i, j - 1)\}. \tag{2.20}$$

Nota-se que tal cálculo é baseado na distância $\mathbf{D}(i, j)$ encontrada na célula corrente e o mínimo das distâncias acumuladas dos elementos adjacentes.

O DTW foi usado como medida de similaridade do algoritmo KNN aplicado à classificação de seqüências, visto que utilizar a distância euclidiana como medida de similaridade, na classificação de seqüências, não é possível em muitos casos ou mesmo o mais recomendável [14, 43]. Neste caso, o classificador KNN para o módulo de classificação de seqüências é chamado de segmental KNN. A literatura também costuma apresentá-lo como 1NN-DTW [43].

2.7 Conclusões do Capítulo

Este capítulo apresentou algumas das principais técnicas e algoritmos de classificação para séries temporais. As séries temporais multivariadas caracterizam bem a natureza e complexidade de um conjunto de dados relacionados a faltas em linhas de transmissão. Assim, o capítulo se preocupou em apresentar uma notação para representar e classificar séries temporais multivariadas, levando em conta o processo de janelamento. Foi visto que a classificação de faltas pode ser realizada a nível de quadro em um cenário de análise *on-line* ou a nível de seqüência em um cenário de análise pós-falta.

Na análise *on-line*, a classificação de faltas é realizada quadro-a-quadro, permitindo o uso de classificadores convencionais. A escolha dos classificadores convencionais descritos neste trabalho foi baseada no fato de que estes são representações populares de diferentes paradigmas de aprendizagem (probabilístico, *lazy*, etc) [9].

Na análise pós-falta, foi visto que se pode utilizar técnicas como HMM e DTW, as quais lidam diretamente com as seqüências. Alternativamente, para utilizarmos classificadores convencionais tais como árvores de decisão e rede neural na solução de problemas de classificação de seqüências, pode-se empregar a arquitetura de classificação de seqüências baseada em quadros (FBSC).

O próximo capítulo apresenta uma descrição completa da base de dados utilizada para aplicação dos algoritmos de classificação.

Capítulo 3

UFPAFaults: Uma base de dados de faltas em linhas de transmissão

3.1 Introdução

Este capítulo descreve a base de dados UFPAFaults[1] bem como os principais programas e rotinas utilizadas para o desenvolvimento da mesma. A primeira versão da UFPAFaults tinha apenas 300 faltas, o que não era suficiente para treinamento robusto dos classificadores. A segunda versão (UFPAFaults2) foi utilizada para realização de experimentos publicados em [44]. A necessidade de alterações e adaptações nesta segunda versão foi observada, sendo então necessária a criação da terceira base, denominada de UFPAFaults3. A base UFPAFaults3 é composta por 1500 faltas, com formas de onda de tensão e corrente no tempo[2], organizadas em três conjuntos de arquivos disjuntos com 500 exemplos cada, para serem usados no processo de treinamento, validação e teste. A Figura 3.1 mostra esquematicamente o procedimento adotado e os softwares utilizados para composição da base de dados UFPAFaults e seu posterior uso.

3.1.1 *Alternative Transient Program*

Alternative Transient Program (ATP) é um software bastante conhecido e utilizado na área de sistemas de potência, reconhecido pelo "Operador Nacional do Sistema" (ONS)

[1] A base UFPAFaults está disponível em (*www.laps.ufpa.br/freedatasets/UfpaFaults/*) .

[2] As simulações são arquivos no formato ASCII organizados da seguinte maneira: a primeira coluna indica o intervalo de tempo, as colunas 2, 3 e 4 são os valores de tensão e as demais colunas são os valores de corrente.

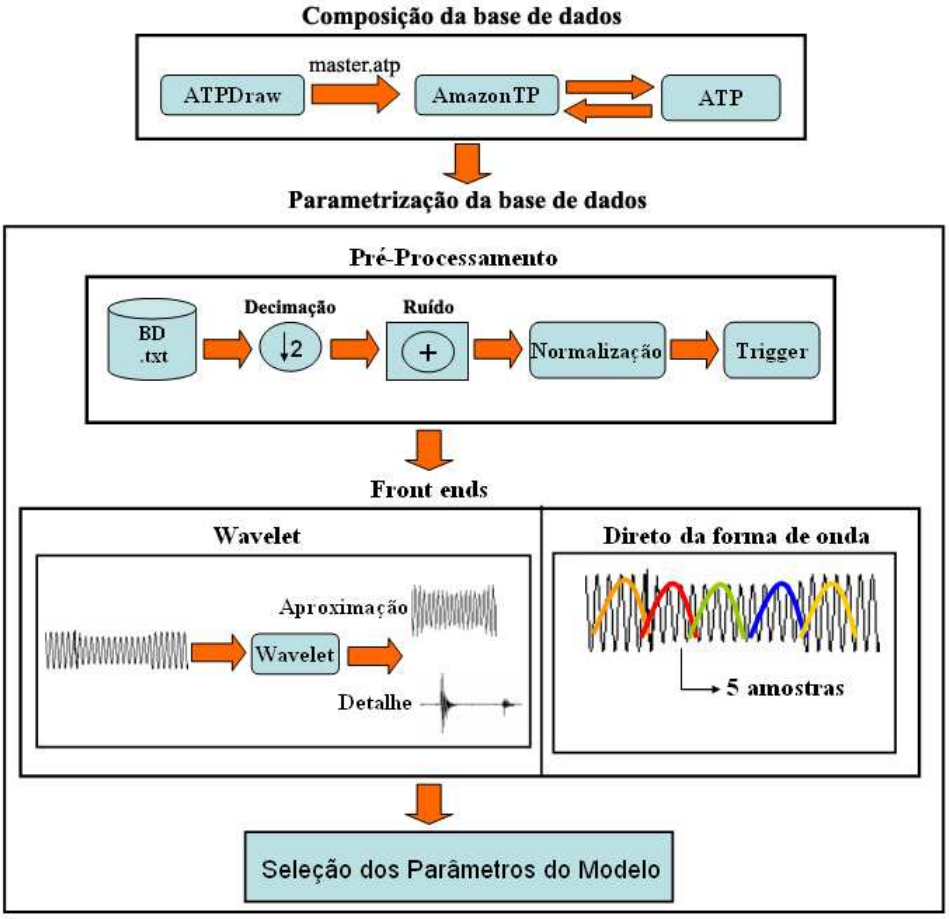

Figura 3.1: Esquema da metodologia adotada para composição da base UFPAFaults e seu posterior uso.

elétrico como uma das melhores ferramentas para execução de análise e simulações digitais de fenômenos transientes em sistemas de potência, tanto de natureza eletromagnética quanto eletromecânica.

A estrutura do ATP compreende um pacote de programas e rotinas de suporte. A Figura 3.2 mostra de forma esquemática o relacionamento entre o programa principal do ATP (TPBIG.EXE) e os programas responsáveis pelo pré e pós-processamento, tais como:

- ATPDRAW[3]: programa para criação do arquivo de dados a partir da elaboração de um desenho do circuito a ser simulado;
- LCC: programa para o cálculo de parâmetros de linhas e cabos;

[3]Tanto o ATPDraw (versão 5.1) quanto ATP (versão 4.12) são disponibilizados, para usuários cadastrados (isto é, deve-se fornecer uma senha), no www.eeug.org.

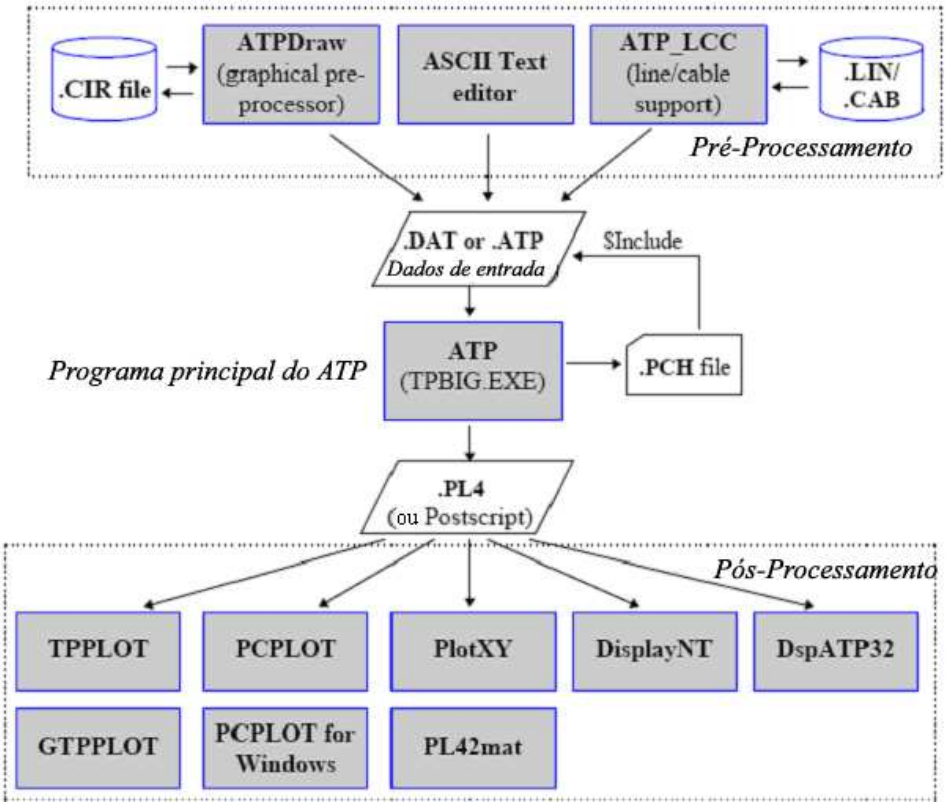

Figura 3.2: Principais programas relacionados ao ATP.

- PlotXY, GTPPLOT, TPPLOT e PCPLOT e outros: programas para apresentação dos resultados sob a forma de gráficos;

A simulação de um processo transitório com o programa principal, o TPBIG.exe, baseia-se no método de DOMMEL, que combina a regra de integração numérica trapeizodal com o método de Bergeron, e se realiza com um passo de integração constante que deve ser escolhido pelo usuário [45]. De um modo geral, os parâmetros necessários para o processamento de casos no ATP envolvem o fornecimento de instruções de informações tais como: o passo de integração, tempo máximo de simulação, freqüência de saída de resultados, etc.

O processo de simulação utilizando o ATP depende de um arquivo de dados em formato texto (ASCII) que neste trabalho chamamos de master.atp. Como o ATP é um programa antigo, desenvolvido em Fortran, a maioria de seus usuários utiliza a interface gráfica ATPDraw para gerar o arquivo com a descrição completa do circuito elétrico (geralmente um modelo de uma rede real) a ser simulado. De um modo geral, o programa ATP lê este arquivo de dados em estudo e, após efetuar o processamento desse arquivo, gera outro arquivo geral com todo o estudo efetuado, cujo nome possui extensão .LIS. Também há a possibilidade da geração, pelo ATP, de um outro arquivo com a extensão .PL4, que apresenta os resultados obtidos na

simulação de tensão, corrente, potência e energia, na forma de vetores coluna.

3.1.2 AmazonTP

O AmazonTP é um software desenvolvido no LaPS da UFPA objetivando automatizar as simulações envolvendo o simulador ATP. O AmazonTP foi escrito em Java, em conformidade com o paradigma da orientação a objetos. O mesmo encontra-se aberto à comunidade acadêmica, como incentivo à reprodução de resultados, facilitando a colaboração científica. O software é distribuído sob a licença "General Public License" (GPL) e se encontra disponível em http://sourceforge.net.

Como foi visto anteriormente, a maneira mais comum de interação com o ATP é usando a interface gráfica ATPDraw, a qual pode criar um arquivo de entrada ATP. Dessa forma, dado um arquivo de entrada ATP representando um circuito elétrico com pelo menos uma linha de transmissão Z-T [12] (devido à limitação atual do AmazonTP, de apenas suportar faltas nesse tipo de linha), o AmazonTP identifica automaticamente todas as ocorrências das linhas de transmissão Z-T, substitui cada uma delas por um bloco descrito na Figura 3.3, e cria um novo arquivo ATP. Baseado neste arquivo *master*, o AmazonTP pode, então, simular curtos-circuitos através de fechamento e abertura das chaves. Por exemplo, como ilustrado n Figura 3.4, uma falta do tipo AB é simulada com o fechamento das chaves SW-A e SW-B, mantendo-se as demais abertas.

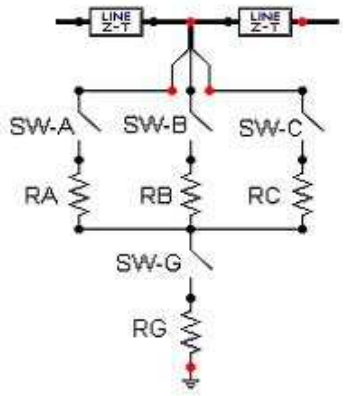

Figura 3.3: Bloco para simulação de uma falta (representado com o software ATDraw). Os elementos SW são as chaves e R são as resistências.

A localização da falta é determinada a partir das dimensões das duas linhas Z-T identificadas na Figura 3.3. Por exemplo, se a linha Z-T original é de comprimento 4 km e deseja-se simular uma falta que ocorre a 1 km do início da linha, então o primeiro bloco de linha Z-T apresenta comprimento de 1 km, enquanto o segundo tem 3 km.

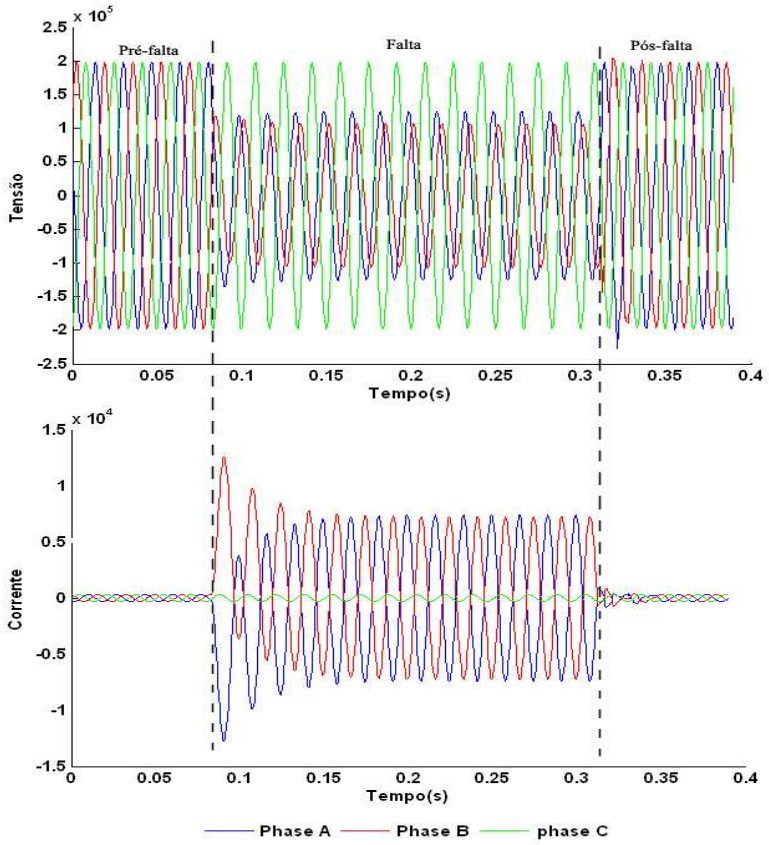

Figura 3.4: Zoom das formas de onda no momento de uma falta AB gerada pelo AmazonTP.

O AmazonTP fornece duas maneiras de variação dos parâmetros relacionados a faltas do tipo curto-circuito em linhas de transmissão: determinística ou probabilística. No modo determinístico o usuário define a posição da falta, o tipo da falta (AB,AT,...,etc), os parâmetros dos componentes de resistência da falta e os instantes de operação das chaves. Já no modo probabilístico, o usuário define uma função densidade de probabilidade uniforme ou Gaussiana para os parâmetros. Isto ajuda a incluir conhecimento na operação da rede através de estatísticas previamente coletadas. Um possível conjunto de parâmetros selecionados aleatoriamente seria: tipo de falta = ABC; índice da linha = 3; início da falta = 0.4 s; término da falta = 0.8s; localização da falta = 7.2% (do tamanho da linha); resistência entre as linhas = 72 Ohms; resistência da linha para a fase terra = 39 Ohms.

A partir das informações de configuração fornecidas pelo usuário, o AmazonTP invoca repetidamente o ATP, modificando o arquivo master ATP para criar novos arquivos ATP. No fim de cada interação com o ATP, o AmazonTP gera um arquivo no formato ASCII com as formas de onda de tensão e corrente medidas.

3.2 Composição da base de dados UFPAFaults

Existem áreas, como como processamento de voz, onde existem bases de dados e tarefas bem definidas, proporcionando, desta forma a possibilidade de facilmente comparar diferentes algoritmos. Em contraste, a área de mineração de dados (na qual a classificação é uma das possíveis atividades) de séries temporais possui limitações neste aspecto [14]. Assim, comparações entre diferentes algoritmos nesta área são problemáticos. Esta situação é compartilhada no caso de séries temporais consistindo de eventos de qualidade de energia elétrica. Além disso, em eventos de QEE, a composição de uma base de dados estatisticamente significante e rotulada é uma tarefa que consome tempo e custos.

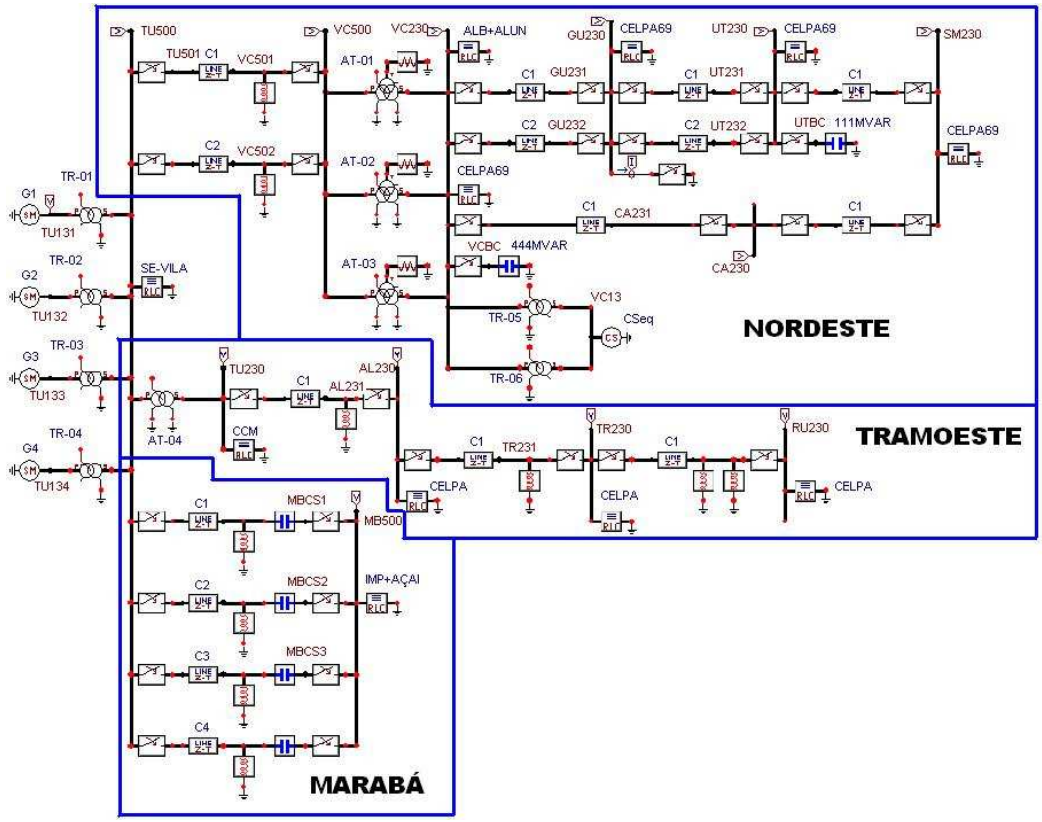

Figura 3.5: Representação gráfica do sistema Eletronorte no ATPDraw.

Nesse contexto, verificou-se a necessidade de compor uma base de dados de séries temporais representando eventos de qualidade de energia elétrica utilizando o software AmazonTP [13]. Como mencionado, o processo de composição da base de dados UFPAFaults com eventos de faltas do tipo curto-circuito (fases A, B, C e terra) dependeu de um arquivo denominado de master ATP, o qual foi gerado com o ATPDraw (Seção 3.1.2). Este arquivo foi desenvolvido com base em um modelo ATP de um sistema real da Eletronorte. A Figura 3.5 exibe o circuito no formato do ATPDraw desenvolvido pelo Grupo de Energia e Sistemas de

Instrumentação (GSEI) da UFPA. O circuito desse sistema Eletronorte foi subdividido para efeito de estudos em três sub-circuitos: Marabá, Nordeste e Tramoeste. Entretanto, a base UFPaFaults3 usou somente o sub-circuito elétrico Tramoeste, descrito com mais detalhes na Figura 3.6. Observa-se nesta figura que o modelo do sistema Tramoeste da Eletronorte tem três linhas Z-T e foi assumido que a falta pode ocorrer com a mesma probabilidade para qualquer uma das três linhas. A partir do sub-circuito Tramoeste obtém-se um arquivo master.atp e este é apresentado ao AmazonTP para geração de um novo arquivo master modificado contendo eventos de faltas.

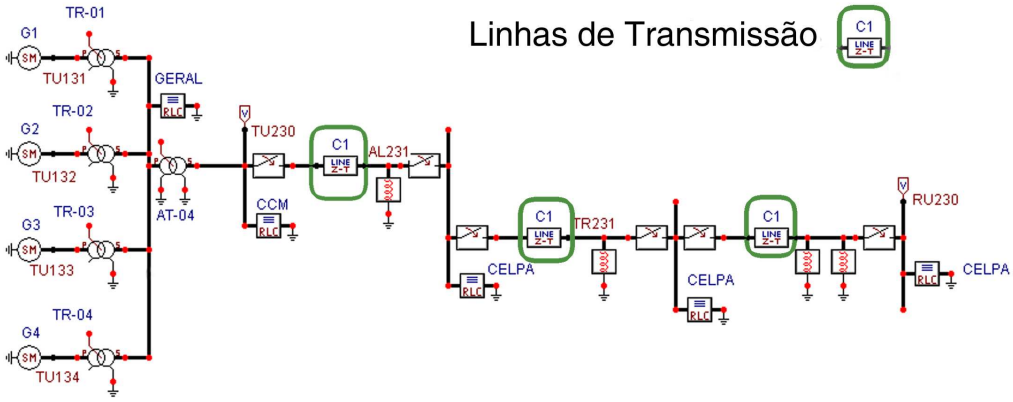

Figura 3.6: Diagrama do modelo ATP para o sub-circuito Tramoeste da Eletronorte com três lihas Z-T (rotuladas "C1").

O AmazonTP foi usado para gerar 11 tipos de faltas uniformemente distribuídas. Os parâmetros necessários à geração de uma falta foram randomicamente gerados. Os valores das quatro resistências foram obtidos como amostras independentes e identicamente distribuídas (i.i.d) de uma função densidade de probabilidade (pdf) uniforme $\mathcal{U}(0.1, 10)$, com suporte de 0.1 até 10 Ohms. O instante do início da falta (do qual o *ângulo de incidência* depende [7]) e a duração da falta, ambos em segundos, foram derivados a partir de $\mathcal{U}(0.1, 0.9)$ e $\mathcal{U}(0.07, 0.5)$, respectivamente. A localização foi obtida a partir de $\mathcal{U}(2, 98)$, onde o número corresponde à porcentagem do tamanho total da linha.

3.3 Parametrização da base de dados

Esta etapa pode ser subdividida em três tarefas: primeiramente realiza-se o pré-processamento; em seguida define-se quais os parâmetros que serão utilizados pelos algoritmos de aprendizagem, para isso utilizam-se os chamados *front ends*; e por último realiza-se o

pós-processamento com a seleção de parâmetros do *front end*.

3.3.1 Pré-processamento

O principal objetivo do pré-processamento é permitir que os dados sejam representados de uma forma mais adequada, sem alterar sua organização ao longo do tempo. Para isso, o pré-processamento foi subdividido em três atividades: decimação e acréscimo de ruído, normalização e *trigger*, as quais são descritas a seguir.

3.3.1.1 Decimação e acréscimo de ruído

As formas de onda geradas pelas simulações ATP tiveram um período de amostragem igual a 0.25 microssegundos (no ATP, deltaT =2.5E-5), o que corresponde a uma freqüência de amostragem de 40 kHz e à geração de uma grande quantidade de dados. Assim, no pré-processamento pode-se reduzir o número de amostras do sinal por meio do processo de decimação (subamostragem). O objetivo da decimação é simplesmente reduzir a quantidade de amostras para diminuir o custo computacional despendido ao aplicar os algoritmos de mineração de dados na base de dados.

Após a decimação, os dados podem receber uma carga de ruído aditivo Gaussiano (controlado pela RSR (relação Sinal/Ruído) em dB especificada pelo usuário) para geração de sinais mais próximos dos sinais reais.

Para realizar essas tarefas foram desenvolvidas rotinas (Apêndice B) na linguagem de programação Java que recebem como entrada os arquivos com os dados gerados pelo ATP. Por exemplo, na rotina de decimação, o usuário determina o fator de decimação no qual o sinal deve ser subamostrado, isto é, com fator de decimação igual a 20 e freqüência de amostragem igual a 40 kHz , a nova freqüência de amostragem passa a ser 2 kHz. Na adição de ruído, o usuário passa como parâmetro a relação sinal/ruído que deseja e, quanto menor esse valor, maior é o ruído adicionado.

3.3.1.2 Normalização

Os diferentes elementos de um vetor **z** de parâmetros de entrada dos classificadores possuem eventualmente distintas faixas dinâmicas. Os algoritmos de aprendizagem podem eventualmente serem completamente ofuscados por parâmetros que têm grandes faixas dinâmicas quando comparados às faixas de outros (por exemplo, no caso, lida-se com tensões de kiloVolts e correntes em Ampéres). Diante disso, tipicamente aplica-se um processo de normalização nos

dados de entrada para que as formas de onda tenham amplitudes, por exemplo, no intervalo de [1, -1] (por unidade ou *pu*).

Em geral, um registro de distúrbio de QEE, que corresponda a uma falta, apresenta três situações [20, 46], nas quais os sinais de tensão e corrente se comportam de modos distintos: os estágios de pré-falta, falta e pós-falta (Figura 3.4). O estágio pré-falta corresponde ao estado normal de operação do sistema elétrico, o segundo indica o intervalo real da falta e o último estágio de pós-falta corresponde ao momento que o sistema de proteção atua tentando restabelecer as condições normais de funcionamento do sistema.

Neste contexto, este trabalho adotou dois tipos de normalizações que são chamadas aqui de *prefault* e *allfault*. Na normalização prefault (Figura 3.7a), um intervalo de duração fixa do sinal que antecede o distúrbio é usado para encontrar o valor máximo absoluto $X_{\max}$. Então, este valor máximo é utilizado como base para conversão de cada fase em pu. Já a normalização allfault (Figura 3.7b) considera a duração de toda forma de onda para calcular as amplitudes máxima e mínima de cada fase, e a converte em pu. Ambas normalizações adotam um fator de normalização distinto para cada uma das formas de onda Q. Por exemplo, se uma forma de onda tem um valor que excede a amplitude $X_{\max}$, a allfault pode converter $X_{\max}$ para 1 e manter toda a forma de onda no intervalo [1, -1], enquanto que prefault pode converter $X_{\max}$ para um valor maior do que 1, já que seu fator de normalização é baseado no intervalo pré-falta, onde as amplitudes são geralmente próximas aos seus valores nominais.

3.3.1.3 *Trigger*

Na classificação de distúrbios de QEE, a duração das faltas varia porque na maioria dos casos é utilizado um gatilho *trigger* que descarta as amostras quando a forma de onda não apresenta qualquer tipo de anomalia, e passa aos estágios subsequentes a matriz $\mathbf{X}$ de tamanho variável, contendo somente o segmento onde a falta foi detectada [40, 47].

O *trigger* por si só corresponde a um problema de classificação binário: "falta" ou não "falta". Em [11, 20, 24] o *trigger* ou módulo de detecção de faltas é realizado a partir da aplicação de *wavelets* ou de classificadores tais como rede neural.

Estimados o início e fim da falta, é comum se processar também um dado número de ciclos de amostras antes e depois da falta. A Figura 3.8 ilustra um caso de aplicação de *trigger* com 3 ciclos antes e depois da falta.

Neste trabalho o *trigger* usa diretamente a informação proveniente do ATP, para assim segmentar a falta a partir do seu início e fim. O software para isso foi implementado como um método de uma classe em Java (Apêndice B). Este método recebe como parâmetro o início e

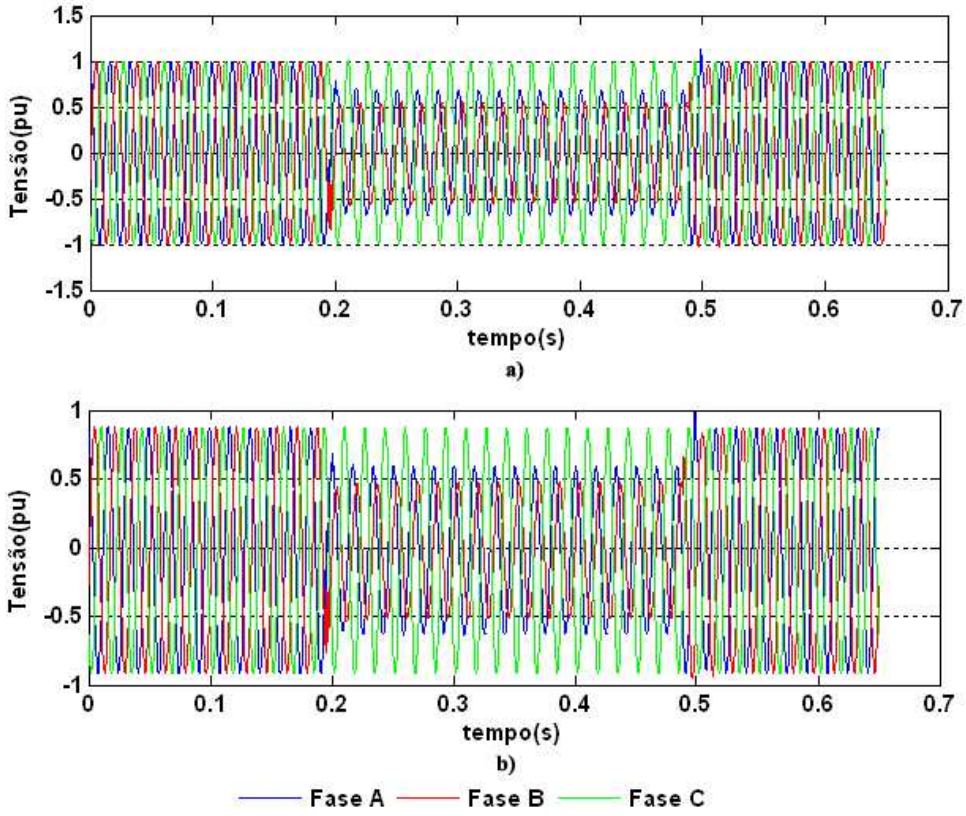

Figura 3.7: Exemplo de formas de onda de tensão ($f_s = 2$kHz) após a aplicação das normalizações *prefault* (a) e *allfault* (b). Em (a) o pico maior do que 1 pode ser visto enquanto que em (b) o maior pico é exatamente 1.

o fim da falta fornecidos pela simulação utilizando AmazonTP, e converte os dados de entrada em outros arquivos de dados somente com as amostras correspondente à falta.

3.3.2 *Front end*

O *front end* é responsável por todas as operações de extração de parâmetros que geram as seqüências que serão passadas aos algoritmos de classificação. A maior parte da literatura em classificação de faltas adota a rede neural como algoritmo de aprendizagem e dois tipos de *front end*: *wavelets* e um aqui chamado *raw* [13, 18, 20]. Todos os *front ends* nesse trabalho assumem seqüências $\mathbf{Z}$ com $Q = 6$ formas de onda de tensão e corrente.

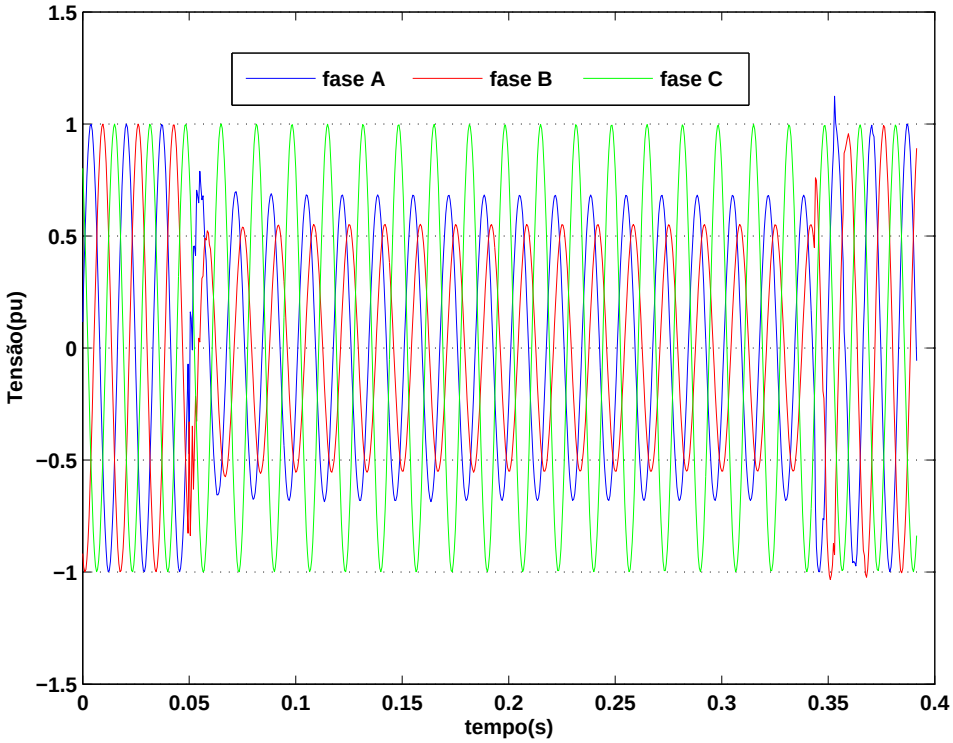

Figura 3.8: Exemplo do uso do *trigger* com $f_s = 2$ kHz.

3.3.2.1 Direto da forma de onda - *Raw*

Este é o primeiro exemplo de *front end*, o qual foi obtido pela simples concatenação das amostras do sinal original a partir do tamanho do quadro L fornecido pelo usuário. Foi desenvolvido um pequeno script em Java (Apêndice B) que implementa tal *front end*. Essa classe gera um arquivo no formato arff do WEKA [9] com K indicando o número de atributos no arquivo arff. Cada linha neste arquivo corresponde ao número de quadros extraídos do sinal. A Figura 3.9 mostra o cabeçalho do arquivo arff gerado pela classe. Observa-se, nesta figura, que os atributos foram nomeados de maneira que pudessem identificar os quadros, implicando, posteriormente, em uma maior interpretabilidade dos resultados. Por exemplo, os atributos Q5VA e Q5IB indicam, respectivamente, o valor do sinal de tensão da fase A no quinto quadro e o do sinal de corrente da fase B também no quinto quadro.

3.3.2.2 *Wavelet*

Como alternativa à representação dos dados em sua forma bruta, a transformada *wavelet* fornece informações do sinal no domínio do tempo e da freqüência, através da "análise de multiresolução" (AMR) [27]. Estas informações podem ser de fundamental importância ao processo de classificação de faltas em linhas de transmissão, pois identificam características

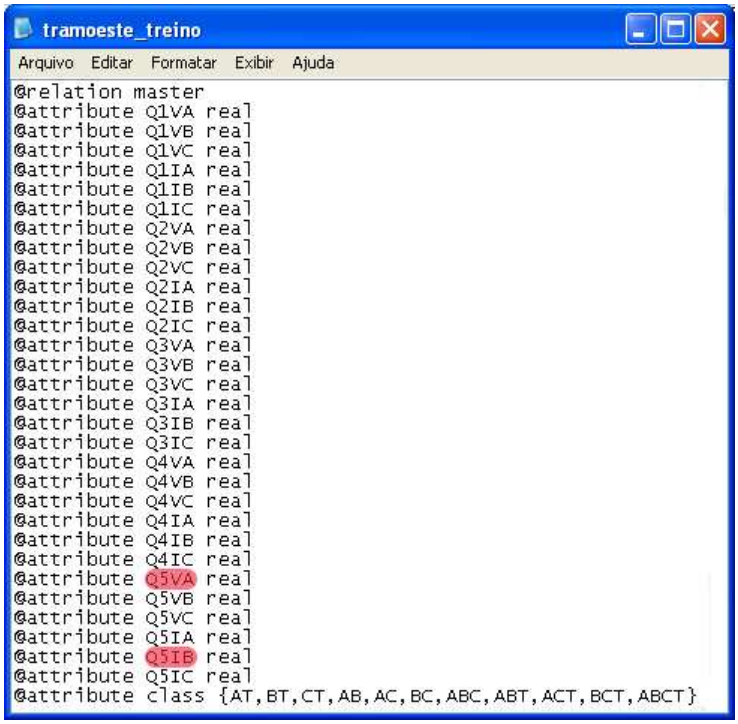

Figura 3.9: Cabeçalho do arquivo gerado em formato arff do WEKA com $K = 30$ para $L = 5$.

únicas das diferentes faltas localizadas nos vários níveis de decomposição do sinal [20, 48, 49].

Na *wavelet* [27], o sinal $s(k)$ analisado é primeiramente decomposto através de dois filtros, um passa-baixa $g(k)$ e um passa-alta $h(k)$ que são representados por funções escalas e *wavelets*, respectivamente, como mostra o esquema da Figura 3.10.

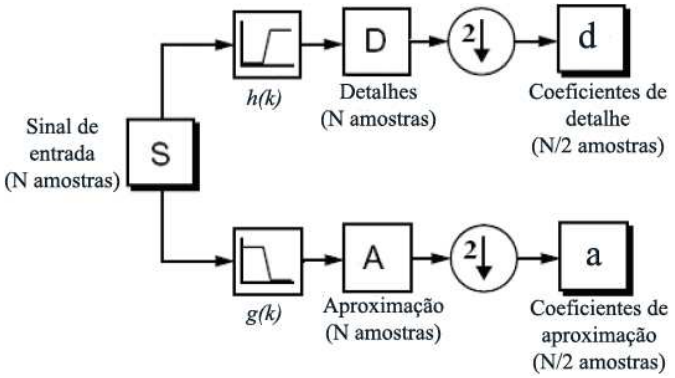

Figura 3.10: Esquema da decomposição *wavelet* com um nível $D = 1$.

A saída do filtro passa-baixa representa o conteúdo de baixa freqüência do sinal original ou uma aproximação do mesmo representado pelos coeficientes de aproximação **a**. A saída do filtro passa-alta representa o conteúdo de alta freqüência do sinal original ou os detalhes representados pelos coeficientes de detalhe **d**. Esses conjuntos são obtidos através da convolução

de $s(k)$ com os filtros, seguido por decimação por dois, isto é, a cada duas amostras de saída do filtro, descarta-se uma delas. A aproximação do sinal original **a**, é então, novamente decomposta, resultando em dois novos sinais de detalhe e aproximação, em diferentes níveis de freqüência. Este processo é repetido de maneira iterativa obtendo-se uma decomposição em múltiplos níveis. A Figura 3.11 apresenta um exemplo da aplicação aos sinais de tensão das fases A e B de uma falta AB.

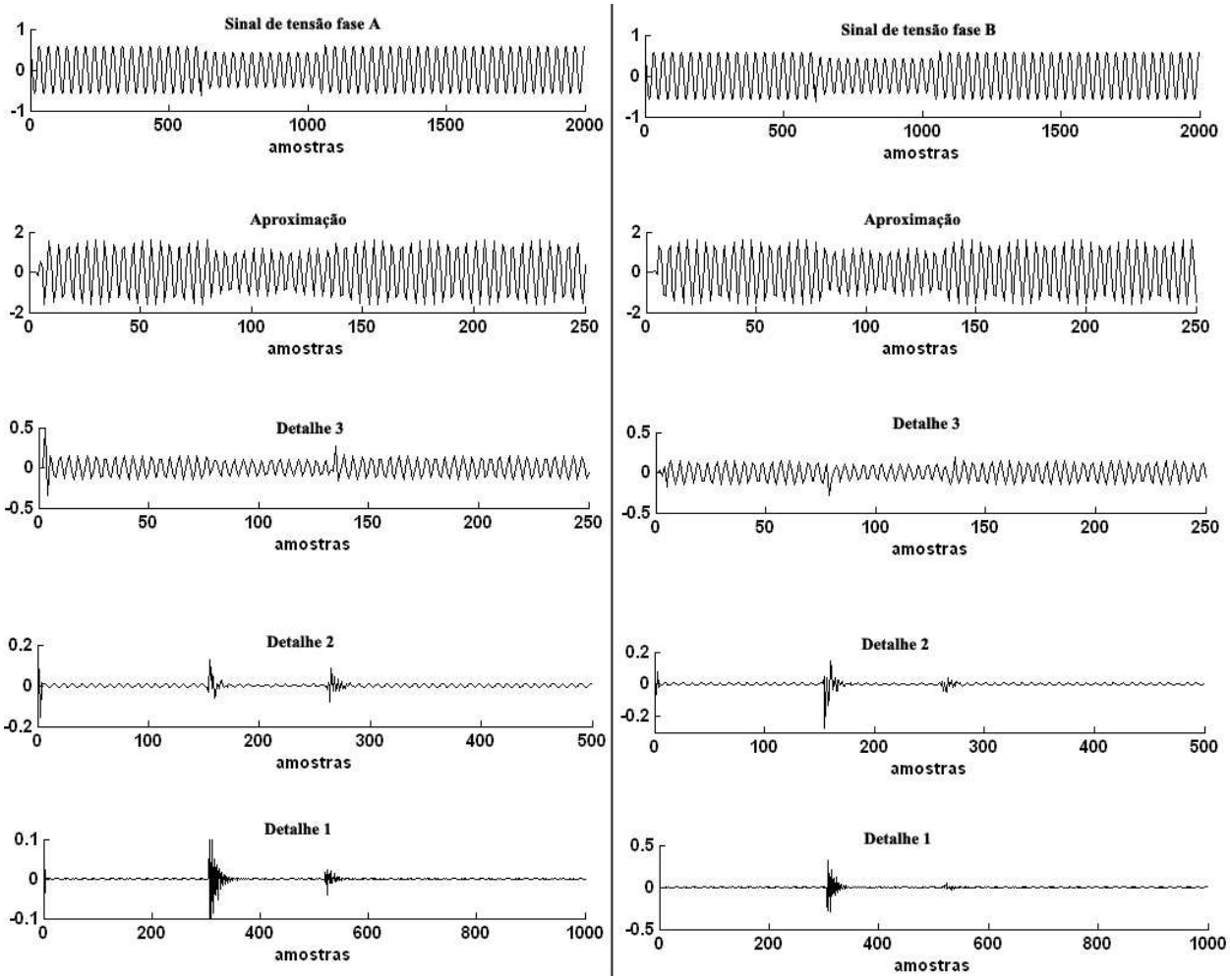

Figura 3.11: Exemplo de uma decomposição *wavelet* aplicada aos sinais de tensão das fases A e B de uma falta AB simulada no intervalo de 1s. A *wavelet* é uma Daubechies 4 com três níveis de decomposição ($D = 3$).

Existem diferentes maneiras de representar uma seqüência através de coeficientes *wavelets*. Neste trabalho dois *front ends* baseados em *wavelets* são investigados.

Alguns trabalhos na literatura usam somente um dos coeficientes de detalhe ou calculam a energia média dos coefiecientes [29]. Em contrapartida, o *front end* denominado aqui de *waveletconcat*, concatena todos os coeficientes *wavelet* e organiza-os em uma matriz **Z**. Este

leva em consideração que os coeficientes possuem diferentes freqüências de amostragem. Por exemplo, assumindo uma aplicação *wavelet* com 3 níveis de decomposição de um sinal com $f_s = 2$ kHz, existem 4 sinais (seqüências de coeficientes) para cada uma das formas de onda Q: a aproximação $\mathbf{a}$, e três detalhes $\mathbf{d}_1$, $\mathbf{d}_2$ e $\mathbf{d}_3$, que possuem freqüências de amostragem iguais a, respectivamente, 250, 1000, 500 e 250. Assim, ao invés de utilizarmos um simples L, o *front end waveletconcat* adota um valor L_{min} para os sinais com menor f_s ($\mathbf{a}$ e $\mathbf{d}_3$ no exemplo anterior). Os outros usam um múltiplo de L_{min}. Considerando ainda o exemplo anterior, o tamanho dos quadros para $\mathbf{d}_1$ e $\mathbf{d}_2$ são, respectivamente, $4L_{min}$ e $2L_{min}$, e o número total de amostras é $L = 8L_{min}$. Uma representação similar pode ser aplicada ao deslocamento S, a qual requer a definição de S_{min}.

Em suma, após aplicação da decomposição *wavelet* os coeficientes são então organizados em um quadro $\mathbf{F}$ de dimensão $Q \times L$, onde $L_k = 2^k L_{min}$ para um nível de decomposição k. Em geral, assumindo que D é o número de níveis da decomposição *wavelet*, o número total de amostras de um quadro é $L = 2^D L_{min}$. O número de quadros é dado por

$$N = 1 + \lfloor (T_a - L_{min})/S_{min} \rfloor \tag{3.1}$$

onde T_a é o número de amostras no coeficiente de aproximação $\mathbf{a}$. Por exemplo, assumindo $L_{min} = 5$, $S_{min} = 2$ e uma matriz $\mathbf{X}$ de dimensão 6×2000 ($f_s = 2$kHz), um quadro $\mathbf{F}$ seria uma matriz 6×40, onde a primeira linha contém todos os coeficientes *wavelets* correspondente à forma de onda de tensão da fase A. Para este exemplo, $K = 240$ e existem $N = 123$ quadros.

Outra alternativa para organizar os coeficientes *wavelet* é calculando a energia média de cada coeficiente. Este *front end* é chamado aqui neste trabalho de *waveletenergy*, e semelhante ao *waveletconcat*, este tem que tratar com sinais de diferentes freqüências de amostragem. A principal diferença entre esses *front ends* é que, ao invés de concatenar todos os coeficientes, *waveletenergy* representa $\mathbf{X}$ por meio da energia E em cada banda de freqüência obtida pela decomposição *wavelet*. A energia é estimada no estilo "média móvel" (*moving average*), ou seja, calcula-se a energia para curtos intervalos ou quadros (mas deve-se evitar confusão com o já utilizado termo "quadro"). Por exemplo, assumindo $L_{min} = S_{min} = 1$ e novamente uma matriz $\mathbf{X}$ de dimensão 6×2000, temos uma matriz 6×4, onde a primeira linha contém a energia média do coeficiente *wavelet* $\mathbf{a}$ calculado a partir da tensão da fase A, logo para o caso da *waveletenergy* $K = 24$ e $N = 125$.

3.3.3 Seleção dos Parâmetros do Modelo

Essa subseção aborda o importante problema de seleção do modelo (*model selection*), ou seja, a escolha dos parâmetros a serem usados nos classificadores, tais como o número de

neurônios na camada escondida de uma rede neural. Essa tarefa não deve ser confundida com a seleção de parâmetros no sentido de *feature selection*, onde um determinado subconjunto dos parâmetros **Z** é escolhido para representar a forma de onda [50].

Freqüentemente, a seleção de parâmetros do modelo é realizada por meio de validação-cruzada (*cross-validation*) [9]. Esta é uma abordagem computacionalmente custosa, mas é importante para evitar ajustar o modelo usando-se o mesmo conjunto a ser usado no teste, pois isso faria com que o modelo eventualmente se "viciasse" no conjunto de teste. Se esse fosse o caso, a taxa de erro no conjunto de teste não exprimiria adequadamente a capacidade de generalização do modelo. Entretanto, a validação-cruzada só poderia ser usada no presente trabalho com a seguinte ressalva.

Convencionalmente assume-se que os exemplos de treino são independente e identicamente distribuídos (i.i.d) de uma desconhecida mas fixa distribuição $P(\mathbf{z}, y)$. Entretanto, esta hipótese não é válida quando a base de dados possui vetores $\mathbf{z}$ extraídos de uma composição de quadros contínuos. Este fato é importante na prática já que quando se realiza seleção de modelo baseado em validação-cruzada, o procedimento pode levar ao que a literatura chama de sobre-especialização (*"overfitting"*), pois os vetores que foram extraídos da mesma forma de onda possuem um grau elevado de similaridade entre eles. Por causa deste fato, a seleção automática de modelos adotada neste trabalho utilizou um arquivo de validação correspondente a um conjunto de exemplos disjunto em relação aos dos arquivos de treino e teste.

Tendo-se o erro de classificação no conjunto de validação como figura de mérito, a melhor combinação dos possíveis parâmetros dos classificadores foram procuradas com o auxílio de um *grid* (produto cartesiano dentre as opções sugeridas ao programa de busca). Cada parâmetro compondo o grid (cada eixo cartesiano) foi especificado a partir de incremento linear ou geométrico (usando-se adição ou multiplicação, respectivamente). Em ambos os incrementos as opções dos parâmetros dos classificadores são especificadas de acordo com um valor mínimo V_{min}, um valor máximo V_{max} e o número de passos V_p. O fator de variação das opções dos parâmetros de um determinado classificador utilizando o incremento linear V_l é dado por:

$$V_l = \frac{(V_{max} - V_{min})}{(V_p - 1)} \tag{3.2}$$

No caso do incremento geométrico o fator de variação V_g é igual a V_p. Considerando uma árvore de decisão e seus parâmetros -C (fator de confiança) e -M (número mínimo de instâncias por folha) com as opções especificadas, respectivamente, a partir de um incremento linear e geométrico, e iguais a $V_{min} = 5$, $V_{max} = 25$ e $V_p = 10$, temos a formação do *grid* representado na Figura 3.12.

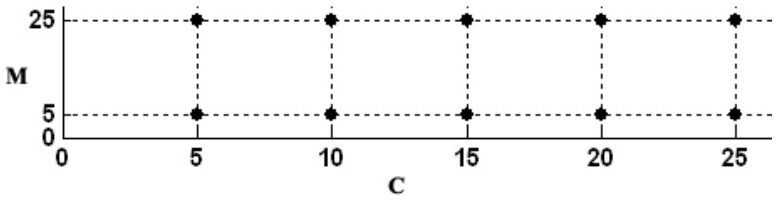

Figura 3.12: *Grid* da faixas de valores dos parâmetros -C e -M de uma árvore de decisão.

3.4 Conclusões do capítulo

Este capítulo apresentou as etapas realizadas para compor e parametrizar a base de dados UFPAFaults. No processo de composição foram utilizados softwares como o popular ATP e o de livre distribuição AmazonTP. Foi visto que o AmazonTP é usado para simular diversas situações de faltas do tipo curto-circuito no sistema de energia elétrica estudado invocando repetidamente o ATP.

A parametrização da base UFPAFaults foi dividida em 3 tarefas: pré-processamento, extração de parâmetros via aplicação de *front end* e seleção de parâmetros do modelo.

No pré-processamento foram apresentadas técnicas tais como decimação e acréscimo de ruído, normalização (*prefault* e *allfault*) e *trigger* para tratamento das formas de onda obtidas pelas simulações sem alterar sua organização no eixo do tempo.

Foram discutidos dois tipos de *front end*: direto da forma de onda (*raw*) e *wavelet*. No *front end raw* simplesmente concatenamos as amostras do sinal original a partir da definição do tamanho do quadro L. No *front end wavelet* foram propostas duas técnicas: concatenando todos os coeficientes *wavelet* denominada de *waveletconcat* e calculando a energia do coeficientes denominada de *waveletenergy*.

Na fase de seleção de parâmetros do modelo foi discutido que utilizar técnicas como *cross-validation* só é recomendado para dados que sejam independentes e identicamente distribuídos, o que não é o caso de amostras retiradas de uma mesma série temporal (falta). Desta forma, nesta fase deve ser utilizado um arquivo de validação disjunto em relação aos arquivos de treino e teste. Usou-se nesse trabalho a variação das opções dos parâmetros a partir de um incremento linear (adição) ou geométrico (multiplicação).

O próximo capítulo apresenta os resultados obtidos após a aplicação dos algoritmos na base de dados UFPAFaults.

Capítulo 4

Experimentos

4.1 Introdução

Este capítulo tem como objetivo avaliar os resultados obtidos utilizando as técnicas de classificação de séries temporais apresentadas na Seção 2.3.

As formas de onda de 40 kHz do UFPaFaults foram decimadas por 20 para criar seqüências com $f_s = 2$ kHz. Resultados para os três diferentes *front ends* na Tabela 4.1 são discutidos. Por exemplo, no primeiro *front end*, direto da forma de onda ou simplesmente *raw*, quando o tamanho do quadro $L = 9$ (uma amostra central, quatro à esquerda e quatro à direita), a dimensão do vetor de entrada K é igual a 54. Neste *front end* não foi usada sobreposição, isto é, $S = L$.

Os dois *front ends wavelet* usaram como *wavelet* mãe a *Daubechies* 4 (db4) [27] com $D = 3$ níveis de decomposição. Conseqüentemente, para cada uma das $Q = 6$ formas de onda, a decomposição *wavelet* gerou quatro sinais. Na *waveletconcat*, por exemplo, quando $L_{min} = 4$ e $S_{min} = 2$, $K = 192$. Já na *waveletenergy*, $K = 24$ para $L_{min} = S_{min} = 1$.

4.2 Resultados para classificação on-line

Os resultados para classificação *on-line* usando o *front end raw* são mostrados na Figura 4.1. Tal figura apresenta uma comparação entre os dois métodos de normalização apresentados neste trabalho, *prefault* e *allfault*. Para ambos, os melhores resultados foram obtidos pelos classificadores RNA e J4.8. Na normalização *prefault* o melhor resultado para o classificador RNA foi com o tamanho do quadro $L = 9$ e na *allfault* o melhor foi com $L = 33$. Já para o clas-

Front end	L ou L_{min}	S ou S_{min}	**Dimensão** (K)
Raw	1	1	6
	5	5	30
	7	7	42
	9	9	54
	11	11	66
	33	33	198
Waveletconcat	4	2	192
	5	2	240
	7	3	336
	9	4	486
Waveletenergy	1	1	24

Tabela 4.1: Parâmetros dos *front ends* usados nas simulações. Além desses, $Q = 6$ e a *wavelet* mãe foi db4 com $D = 3$ níveis de decomposição.

sificador J4.8 o melhor resultado tanto para a normalização *prefault* quanto para a *allfault* foi com $L = 9$. O resultado do naïve Bayes não foi bom, o que podia ser explicado pelo fato da independência condicional não se aplicar no caso. O desempenho dos outros dois classificadores, GMM e KNN, requer uma análise mais pormenorizada, pois seus parâmetros precisariam ser modificados sistematicamente de forma mais ampla e auferidos os desempenhos.

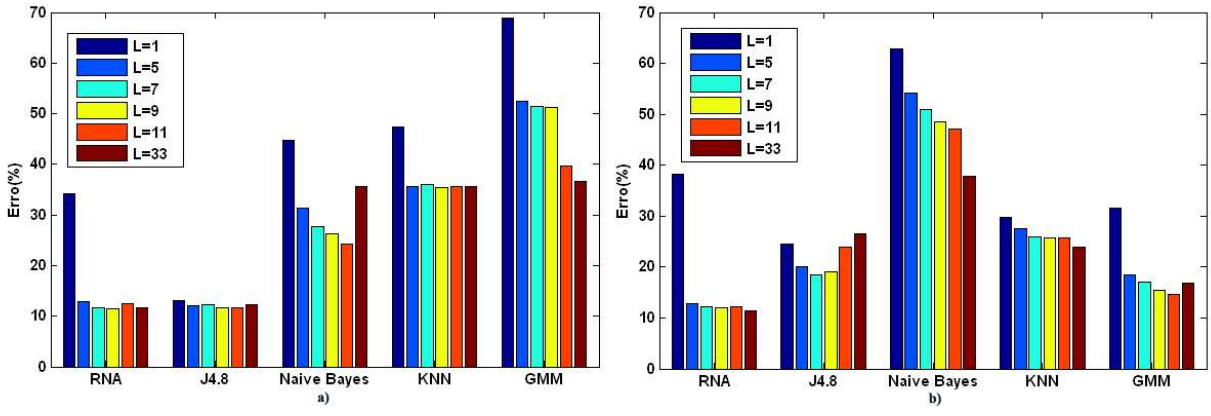

Figura 4.1: Taxa de erro E_f usando o *front end raw* e as estratégias de normalização *prefault* (a) e *allfault* (b).

A Figura 4.2 apresenta os resultados obtidos utilizando os dois *front ends wavelet* com

normalização *prefault* (os resultados para o melhor *raw* foram repetidos por conveniência). Os resultados para *waveletconcat* não são melhores que os obtidos com o *front end raw*, apesar de que, com o classificador RNA a performance não foi tão diferente. O *front end waveletenergy* ao ser comparado com *waveletconcat* apresenta uma melhor performance, exceto em relação aos classificadores RNA e J4.8. Comparado ao *front end raw*, *waveletenergy* apresenta melhores resultados somente com os classificadores KNN e GMM. Deve-se observar, entretanto, que existem diversas maneiras de aplicar um *front end* baseado na transformada *wavelet*, por isso estes resultados devem ser vistos como resultados preliminares.

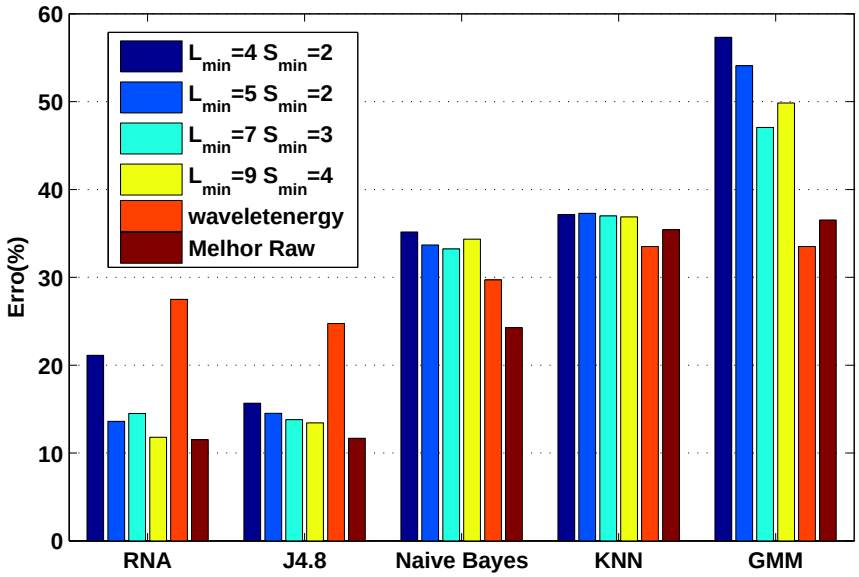

Figura 4.2: Resultados (E_f) para os dois *front ends wavelet*. Os melhores resultados utilizando o *front end raw* com normalização *prefault* também é apresentado para efeito de comparação.

Para um maior detalhamento dos resultados obtidos utilizando o *front end raw* e a normalização *prefault*, são apresentadas nas tabelas 4.2 e 4.3 as matrizes de confusão dos classificadores RNA ($E_f = 11.5\%$) e J4.8 ($E_f = 11.7\%$), respectivamente. O número de acertos, para cada classe, é apresentado na diagonal principal e as outras posições representam erros na classificação. Uma representação gráfica das matrizes de confusão é mostrada na Figura 4.3.

Observando-se as Tabelas 4.2 e 4.3 e a Figura 4.3 pode-se constatar que, tanto o classificador RNA quanto o J4.8, não conseguem distinguir as faltas ABC das ABCT. Este mesmo problema ocorreu com os outros classificadores, incluindo HMM e segmental KNN. Analisando somente as formas de onda de tensão e corrente das fases A, B e C fica difícil para os algoritmos de aprendizagem distinguirem as faltas trifásicas ABC e ABCT devido ao alto grau de similaridade entre suas formas de onda (Figura 4.4). Nota-se que alguns trabalhos [20,31] não

	AT	BT	CT	AB	AC	BC	ABC	ABT	ACT	BCT	ABCT	Total	% Erro
AT	2358	-	1	-	-	-	-	-	-	-	-	2359	0.04
BT	2	2687	1	-	-	-	1	1	-	-	-	2692	0.18
CT	-	1	3015	-	-	-	-	-	-	-	-	3016	0.03
AB	2	3	-	2066	-	-	2	1	1	-	1	2076	0.5
AC	5	-	3	-	2201	1	2	-	1	-	-	2213	0.5
BC	-	2	3	-	-	2572	3	-	-	8	-	2588	0.6
ABC	-	-	-	6	4	-	1116	5	2	9	1204	2346	52.43
ABT	3	32	-	28	1	30	19	2297	2	18	-	2430	5.47
ACT	2	-	3	2	17	1	-	-	2018	-	-	2043	1.22
BCT	-	6	8	2	-	14	1	-	-	2390	-	2421	1.28
ABCT	1	-	-	4	5	2	1659	-	11	7	1514	3203	52.73

Tabela 4.2: Matriz de confusão do classificador RNA utilizando *front end raw* e normalização *prefault*. As linhas indicam as faltas corretas e as colunas a classe predita pelo classificador.

	AT	BT	CT	AB	AC	BC	ABC	ABT	ACT	BCT	ABCT	Total	% Erro
AT	2354	-	3	-	2	-	-	-	-	-	-	2359	0.2
BT	-	2673	8	-	1	5	-	-	1	4	-	2692	0.7
CT	-	1	3014	-	-	-	-	-	1	-	-	3016	0.1
AB	2	8	-	2042	1	1	1	17	2	-	2	2076	1.6
AC	1	-	3	-	2164	-	-	-	45	-	-	2213	2.2
BC	-	3	4	-	-	2537	1	-	-	43	-	2588	2
ABC	-	-	-	4	1	10	1239	-	1	4	1087	2346	47.2
ABT	4	1	1	63	-	-	2	2355	-	-	4	2430	3.1
ACT	1	-	-	-	51	-	-	-	1991	-	-	2043	2.5
BCT	-	2	5	-	-	93	2	-	-	2319	-	2421	4.2
ABCT	-	-	-	1	1	7	1681	2	4	3	1504	3203	53

Tabela 4.3: Matriz de confusão do classificador J4.8 utilizando *front end raw* e normalização *prefault*.

fazem distinção entre esses dois tipos de faltas. Assim, as Tabelas 4.4 e 4.5 mostram, respectivamente, as matrizes de confusão dos classificadores RNA e J4.8 considerando que as faltas ABC e ABCT pertencem a uma mesma classe. Neste caso o erro E_f para a rede neural foi de 1% e para a árvore J4.8 foi de 1.6%. Este novo resultado obtido para RNA, desconsiderando que as faltas ABC e ABCT são diferentes, é equivalente aos resultados obtidos em [20, 31].

A Figura 4.5 mostra o impacto na robustez dos classificadores J4.8 e RNA após a adição de ruído branco Gaussiano nas formas de onda de tensão e corrente. Ambos os classificadores foram treinados com formas de ondas sem qualquer tipo de ruído e testados sob uma condição de razão sinal ruído de 30 dB. Observa-se, neste caso, que a J4.8 é menos robusta à adição de

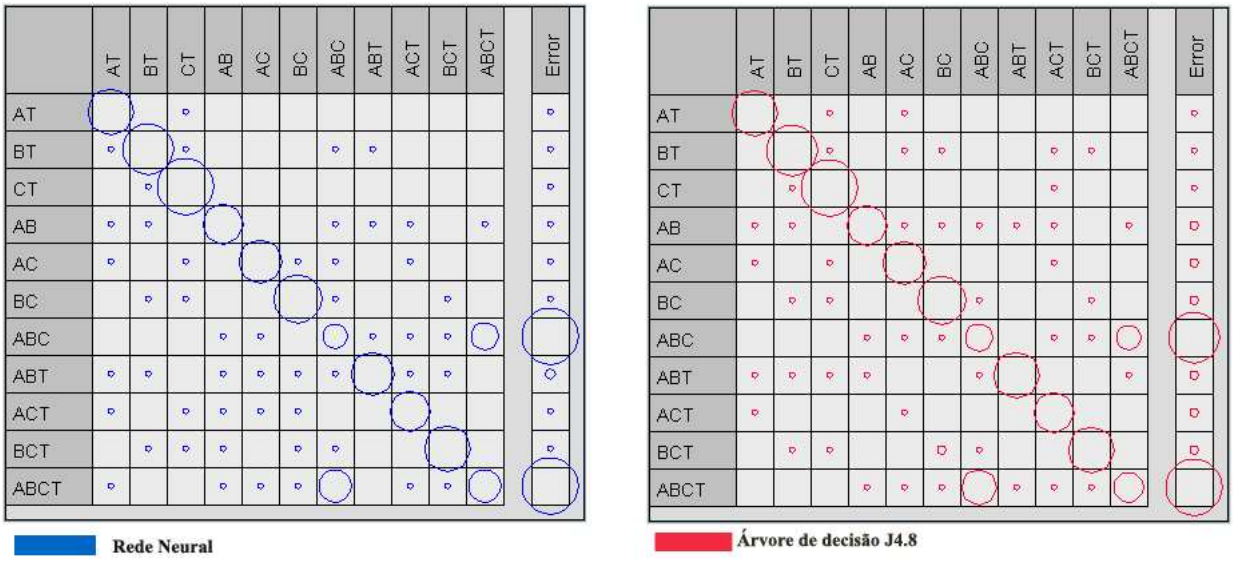

Figura 4.3: Gráfico das matrizes de confusão para rede neural (lado esquerdo) e árvore de decisão J4.8 (lado direito). As colunas indicam as faltas reconhecidas. Os raios dos círculos são proporcionais ao tamanho das entradas.

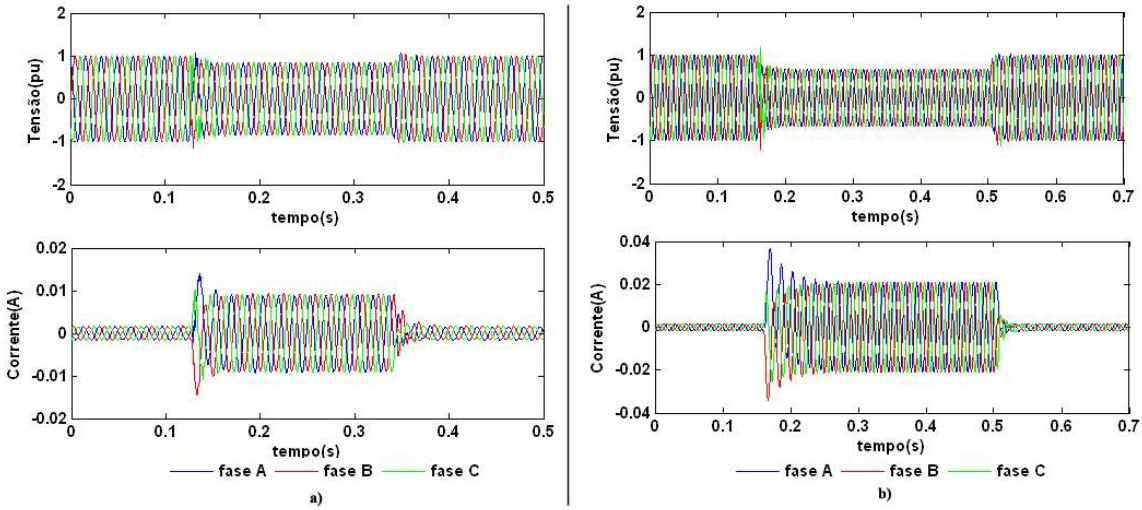

Figura 4.4: Exemplo das formas de onda de tensão e corrente das faltas ABC (a) e ABCT(b). Tais gráficos mostram a grande similaridade entre as duas faltas.

ruído do que a RNA, independente do tamanho de quadro L adotado.

	AT	BT	CT	AB	AC	BC	ABT	ACT	BCT	ABC/ABCT	Total	% Erro
AT	2358	-	1	-	-	-	-	-	-	-	2359	0.04
BT	2	2687	1	-	-	-	1	-	-	1	2692	0.2
CT	-	1	3015	-	-	-	-	-	-	-	3016	0.03
AB	2	3	-	2066	-	-	1	1	-	3	2076	0.5
AC	5	-	3	-	2201	1	-	1	-	2	2213	0.5
BC	-	2	3	-	-	2572	-	-	8	3	2588	0.6
ABT	3	32	-	28	1	30	2297	2	18	19	2430	5.5
ACT	2	-	3	2	17	1	-	2018	-	-	2043	1.2
BCT	-	6	8	2	0	14	-	-	2390	1	2421	1.2
ABC/ABCT	1	-	-	10	9	2	5	13	16	5493	5549	1

Tabela 4.4: Matriz de confusão do classificador RNA não diferenciando as faltas ABC e ABCT.

	AT	BT	CT	AB	AC	BC	ABT	ACT	BCT	ABC/ABCT	Total	% Erro
AT	2354	-	3	-	2	-	-	-	-	-	2359	0.2
BT	-	2673	8	-	1	5	-	1	4	-	2692	0.7
CT	-	1	3014	-	-	-	-	1	-	-	3016	0.07
AB	2	8	-	2042	1	1	17	2	-	3	2076	1.64
AC	1	-	3	-	2164	-	-	45	-	-	2213	2,21
BC	-	3	4	-	-	2537	-	-	43	1	2588	2
ABT	4	1	1	63	-	-	2355	-	-	6	2430	3
ACT	1	-	-	-	51	-	-	1991	-	-	2043	2.5
BCT	-	2	5	-	-	93	-	-	2319	2	2421	4.2
ABCT	-	-	-	5	2	17	2	5	7	5511	5549	0.7

Tabela 4.5: Matriz de confusão do classificador J.48 não diferenciando as faltas ABC e ABCT.

4.3 Discussão acerca de custo computacional

É interessante comparar o custo computacional de alguns classificadores durante o estágio de teste. Esta comparação pode ser feita também durante o treinamento, entretanto é geralmente menos importante. O custo pode ser estimado como descrito a seguir.

Como discutido, uma árvore de decisão binária como a J4.8 com nível de profundidade máximo $d_{\max}$ requer $d_{\max} - 1$ comparações para alcançar uma folha.

Uma RNA composta por uma única camada escondida H com p_h neurônios, calcula, para cada um desses neurônios, um produto interno entre vetores de dimensão $K+1$ (dimensão de entrada mais o bias) e uma função sigmóide. Um procedimento similar é feito para cada um dos Y neurônios da camada de saída, porém com vetores de dimensão p_h. Desta forma, o custo computacional estimado de uma rede neural é dado por: $(Y+p_h)$ *sigm* $+ (K+Y)p_h$ mac, onde *sigm* é uma função sigmóide (função exponencial + 1 soma + 1 divisão) e mac corresponde a

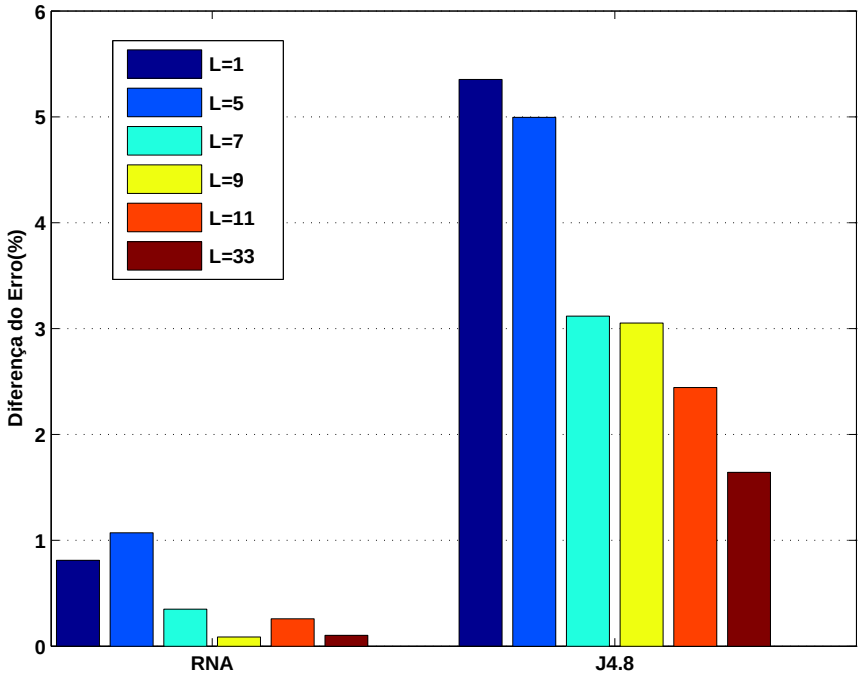

Figura 4.5: Diferença entre o erro E_f para classificação quadro-a-quadro sem ruído e com ruído usando o *front end raw* e a normalização *prefault*.

operações de adição e multiplicação, executadas em um ciclo em modernos chips DSP.

Um classificador KNN usando p_m centróides (note que o K no nome KNN não se refere ao número p_m de centróides) obtidas pelo processo de agrupamento, requer a computação de p_m produtos internos para o cálculo das distâncias Euclidianas ou, alternativamente, o cálculo de p_m normas dos vetores de dimensão K elevadas ao quadrado.

Assumindo que cada uma das Y classes seja modelada por uma mistura com p_g Gaussianas de matrizes de covariância diagonais, o classificador GMM requer Yp_g distâncias Euclidianas para calcular as likelihoods. Por exemplo, considere o cálculo do log da likelihood de uma classe $Y = 1$ considerando apenas uma Gaussiana $p_g = 1$ de dimensão $K = 2$, a qual pode ser decomposta em duas Gaussianas unidimensionais:

$$\begin{aligned} \ln P(X|Y=1) &= \ln\left(G(x_1|\mu_1,\theta_1)G_2(x_2|\mu_1,\theta_1)\right) \\ &= \ln(G_1(x1|\mu_1,\theta_1)) + \ln(G_2(x_2|\mu_2,\theta_2)) \\ &= \ln(c_1 e^{-0.5(x_1-\mu_1)^2\theta'_1}) + \ln(c_2 e^{-0.5(x_2-\mu_2)^2\theta'_2}) \\ &= c - 0.5(x_1-\mu_1)(x_1-\mu_1)\theta'_1 - 0.5(x_2-\mu_2)(x_2-\mu_2)\theta'_2 \end{aligned} \tag{4.1}$$

onde c é uma constante computada uma única vez e $\theta' = 1/\theta$ é o inverso da variância. Logo, o custo computacional estimado da GMM é igual a $p_g YK$ operações de MAC.

Há várias implementações do naïve Bayes, mas quando o mesmo usa uma Gaussiana

para cada elemento da entrada **z** (como neste trabalho), seu custo é equivalente ao de uma GMM com uma Gaussiana por classe.

A estimativa (aproximada) do custo total desses classificadores na fase de teste é resumido na Tabela 4.6.

Classificador	**Custo**
J4.8	$d_{\max} - 1$ instruções if-else
RNA	$(Y + p_h)$ sigm, $(K + Y)p_h$ mac
KNN	$p_m K$ mac
GMM	$Y p_g K$ mac
Naive Bayes	YK mac

Tabela 4.6: Custo computacional estimado para alguns classificadores, onde *sigm* é uma função sigmóide e mac é uma operação de multiplicação e adição, executada em um ciclo em chips DSP modernos.

Para fornecer uma estimativa simples do custo, a Tabela 4.7 sugerida em [4] foi usada para atribuir pesos às diferentes operações tais como aritmética e lógica. Estes pesos, obviamente, dependem muito da plataforma computacional, porém eles são úteis para prover uma aproximação inicial.

A Figura 4.6 mostra o custo computacional estimado de alguns classificadores comparado ao tempo (em segundos) de processamento destes no estágio de teste usando um arquivo de teste relativamente pequeno, dimensão de entrada $K = 54$ (para $L = 9$), $Y = 11$ e parâmetros dos classificadores RNA, J4.8, KNN e GMM dados por ph=32, dmax=258, pm =1111 e pg=8, respectivamente. Analisando a Figura 4.6 observa-se duas situações interessantes.

Na primeira situação percebe-se que o classificador naïve Bayes apresenta um custo computacional estimado inferior em relação aos custos estimados para os classificadores GMM e RNA. Entretanto, no que concerne ao tempo de processamento, o naïve Bayes mostrou-se mais lento.

A outra situação que merece destaque envolve a árvore de decisão J4.8. Embora este classificador tenha apresentado o menor custo computacional estimado, seu tempo de processamento foi, em termos absolutos, equivalente aos tempos dos classificadores RNA e GMM.

Estas duas situações mostram que não é fácil estimar o custo computacional dos classi-

Operação	**Exemplo**	**Peso**
adições	()=()+()	1
multiplicações	()=()*()	1
multiplicação-adição (mac)	()=()*()+()	1
deslocamento de dados	float,int	1
operações lógicas	shift, modulo	1
divisões	()=()/()	18
raiz quadrada	()=sqrt()	25
funções transcendentais	seno,log,exp	25
testes aritméticos	if, if-then-else	2

Tabela 4.7: Pesos para diferentes operações [4]. Por exemplo, uma sigmóide pode ser calculada com uma função exponencial, uma adição e uma divisão, tendo peso total igual a 44.

ficadores de maneira a inferir seus tempos de processamento. Isto ocorre, dentre outras causas, porque um mesmo algoritmo de aprendizado pode ser implementado de várias maneiras. Por exemplo, a classe que implementa o algoritmo da árvore de decisão J4.8 no WEKA oferece recursos adicionais que não fariam parte de um algoritmo otimizado para o tratamento de atributos numéricos. No caso do naïve Bayes, o WEKA não usa nenhum recurso de simplificação em seu processamento como, por exemplo, ao invés de calcular as likelihoods, calcular o log neperiano destas (Equação 4.1).

4.4 Resultados para classificação pós-falta

A Figura 4.7 mostra os resultados obtidos para classificação de seqüências utilizando arquitetura baseada em quadros (FBSC). Os resultados obtidos para os dois *front ends wavelet* não são melhores que obtidos pelo *front end raw*. Nos dois tipos de normalização, verifica-se que os classificadores RNA e J4.8 são os que apresentam melhores resultados e neste último, para $L = 11$, a taxa de erro é abaixo dos 10%. Nota-se ainda que KNN e GMM apresentam melhores resultados para a normalização *allfault* e naïve Bayes para a *prefault*.

Para efeito de comparação, a Figura 4.8 mostra $E_f - E_s$, a redução absoluta na taxa de erro ao comparar a classificação de seqüência utilizando a arquitetura FBSC e o correspondente classificador convencional. Por exemplo, GMM apresenta $E_f = 51.3\%$ e $E_s = 31.6\%$ levando

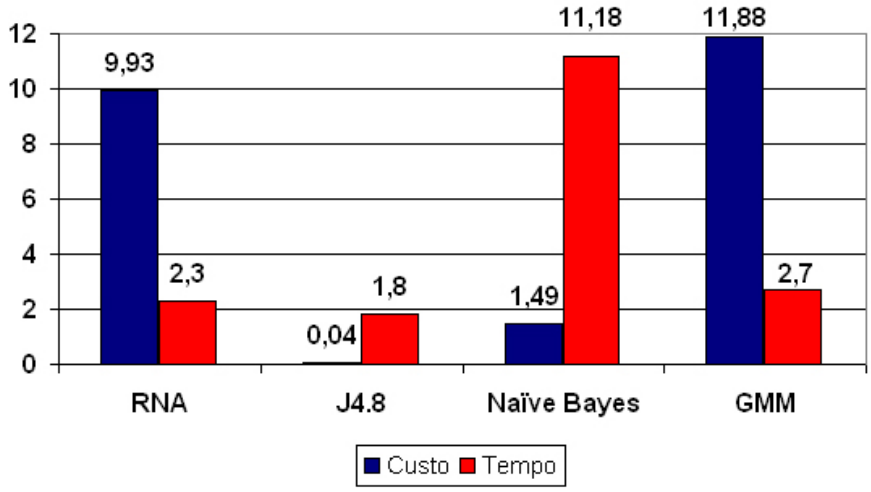

Figura 4.6: Comparação do tempo para o estágio de teste e do custo computacional estimado usando a Tabela 4.6 para os melhores classificadores encontrados pela seleção de modelos, para a normalizações *prefault* com $L = 9$. Os valores estimados do custo dos classificadores foram divididos por 400 para efeito de comparação. O custo computacional estimado do classificador KNN foi em torno de 150 (já dividido por 400) e seu tempo de processamento igual a 148 s, o que o deixa fora da escala dos demais.

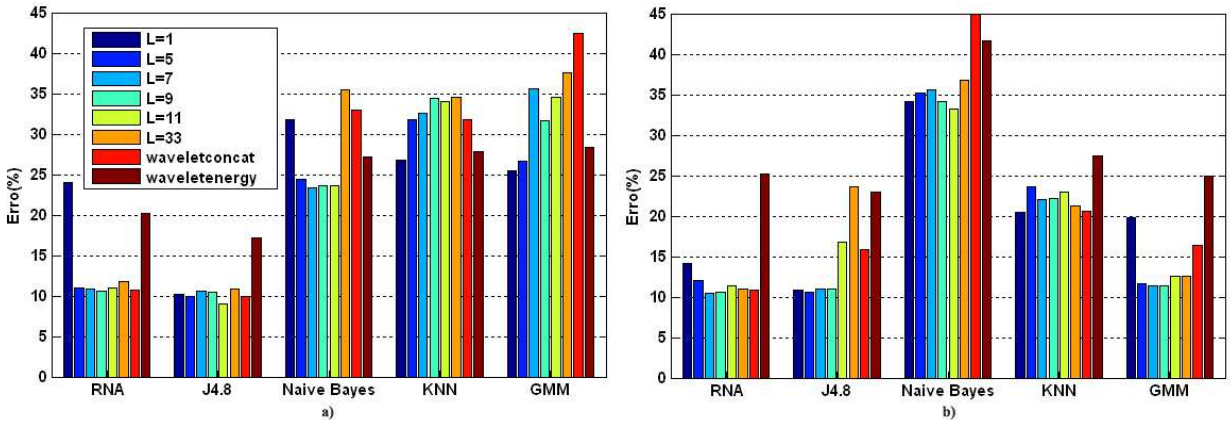

Figura 4.7: Erro E_s para classificação de seqüência baseada em quadros usando os *front ends raw, waveletconcat* ($L_{min} = 9$ *e* $S_{min} = 4$) *e waveletenergy* e as estratégias de normalização *prefault*(a) e *allfault*(b).

a uma diferença de 19.7%.

Foram também registrados resultados preliminares utilizando HMM e segmental KNN como mostra a Figura 4.9.

Os HMMs adotados neste trabalho tiveram uma topologia esquerda-direita sem *skips*

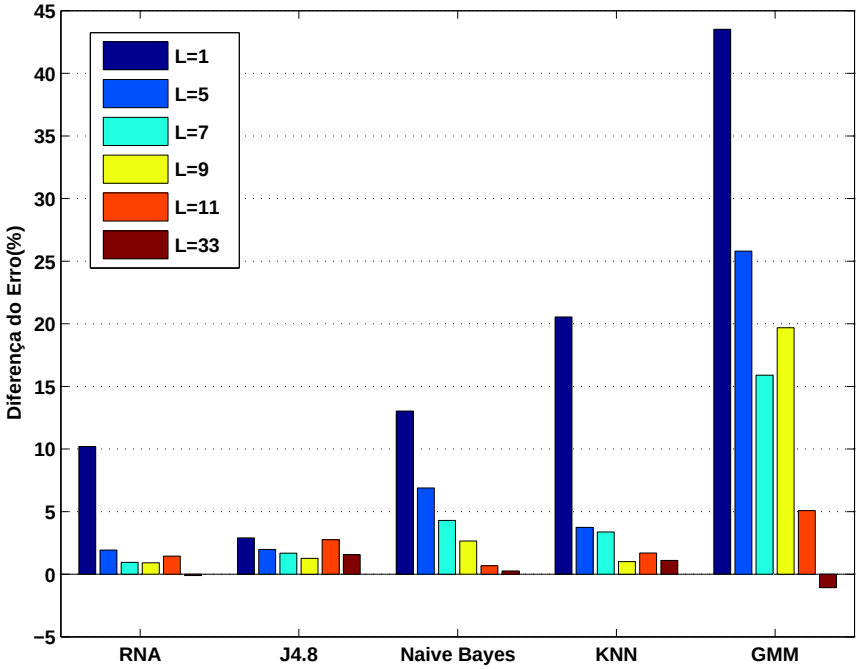

Figura 4.8: Diferença $E_f - E_s$ entre as taxas de erro para classificação quadro-a-quadro e seqüência usando a normalização *prefault* e *front-end raw*.

[38]. O espaço de dimensão foi determinado pelo tipo de *front end*. Para o *front end raw* a dimensão K foi igual $Q = 6$, enquanto que para os *front ends wavelet* a dimensão foi igual ao número de sinais de decomposição $4 \times Q$ totalizando $K = 24$. O número de estados foi arbitrariamente fixado em 13 (onze estados emissores no HTK). O número de Gaussianas por mistura foi variada de 1 a 8. Os melhores resultados foram obtidos com 8 Gaussianas por mistura.

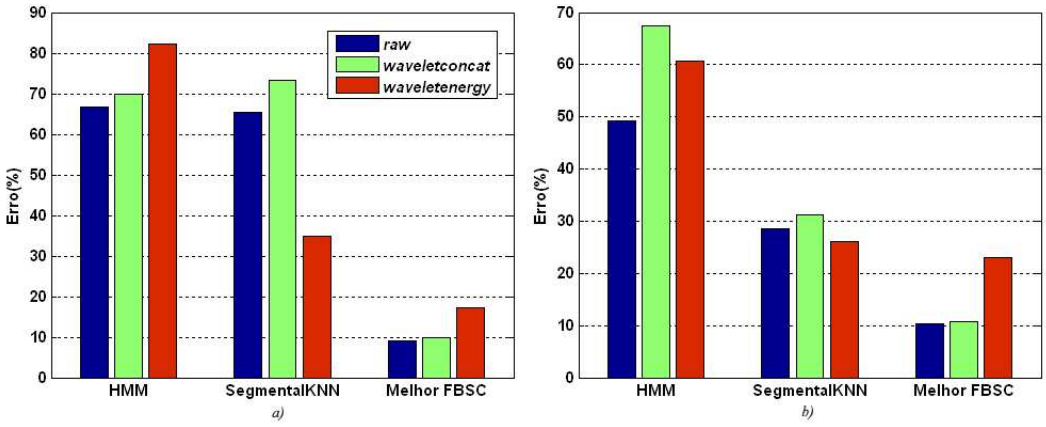

Figura 4.9: Erro E_s para classificação de seqüências utilizando HMM, segmental KNN e FBSC.

O classificador segmental KNN foi treinado com o algoritmo segmental K-means, no

qual o número de centróides K foi igual a 500 seqüências.

Para efeito de comparação, os melhores resultados obtidos pela classificação FBSC (no caso os resultados dos classificadores J4.8 com $L = 11$ e RNA com $L = 7$ para as normalizações *prefault* e *allfault*,respectivamente) foram repetidos por conveniência. Observa-se que os resultados dos classificadores HMM e segmental KNN foram inferiores em relação aos do FBSC. Contudo, é claro que a seleção de modelo no caso da HMM e segmental KNN ainda pode ser bastante melhorada.

4.5 Conclusões do capítulo

O presente capítulo apresentou os principais resultados obtidos para as classificações *on-line* e pós-falta. Foram utilizados nos experimentos as normalizações *prefault* e *allfault* bem como os três tipos de *front end raw*, *waveletconcat* e *waveletenergy*.

Avaliando os dois tipos de normalização foi visto que a *prefault* obteve os melhores resultados, porém para classificadores como GMM e KNN seus resultados foram inferiores em relação à normalização *allfault*.

Entre os *front ends*, o *raw* foi o que obteve melhores resultados em relação aos outros dois *front ends wavelet*. Entretanto, tanto o *front end waveletconcat*, quanto *waveletenergy* poderiam ser mais aproveitados a partir de um melhor ajuste de seus parâmetros. Deve-se também ressaltar que a escolha da *wavelet* db4 pertencente à família de *Daubechies* foi em consideração ao grande número de trabalhos que utilizam tal wavelet na análise de sistemas de energia elétrica [20, 21, 49], mas outras wavelets poderiam ser testadas.

Os resultados obtidos neste capítulo permitiram compreender o comportamento dos classificadores na classificação de séries temporais. Entre os classificadores convencionais, a rede neural foi a que menos sofreu influência quanto ao tipo de normalização adotado. A RNA mostrou-se ainda robusta mesmo após a adição de ruído, caracterizando seu alto nível de generalização. Por outro lado, a árvore de decisão J4.8 também apresentou um bom desempenho com a vantagem de possuir um grau elevado de interpretabilidade.

Os classificadores GMM, KNN e naïve Bayes mostraram-se muito sensíveis quanto ao *front end* e ao tipo de normalização adotados. GMM e KNN, diferente dos outros classificadores, apresentaram melhores resultados utilizando o *front end waveletenergy*.

Na classificação de seqüências utilizando a arquitetura FBSC, assim como na classificação quadro-a-quadro, os resultados obtidos usando o *front end raw* foram melhores do que os resultados com wavelet. Foi feita uma comparação entre os erros E_s e E_f e, a partir desta

comparação, observou-se que o classificador GMM apresentou uma grande discrepância na diferença entre esses dois erros.

As taxas de erro tanto para HMM, quanto para segmental KNN, foram abaixo da expectativa, ao ser comparado, por exemplo, com os resultados da arquitetura FBSC.

Devido as diferentes implementações dos algoritmos utilizados neste trabalho, foi visto que não é uma tarefa fácil avaliar o custo computacional dos classificadores a partir da análise do tempo de seus processamentos.

Foi observado, neste capítulo, que os classificadores não conseguiram distinguir as faltas do tipo ABC e ABCT, visto a grande semelhança nas formas de onda de tensão e corrente dessas duas faltas. Para uma maior diferenciação entre tais faltas poderiam ser analisados outros tipos de parâmetros, além das formas de onda de tensão e corrente, como por exemplo a componente de seqüência zero [20, 21].

Capítulo 5

Conclusões

5.1 Conclusões

Este trabalho apresentou uma descrição dos aspectos relacionados ao projeto de módulos de classificação de faltas em linhas de transmissão para sistemas elétricos de potência. As soluções para este problema envolvem processamento digital de sinais e algoritmos de aprendizagem de máquina. Conseqüentemente, existem muitos graus de liberdade na concepção de um classificador. Por exemplo, os *front ends* wavelet poderiam, provavelmente, ser mais aproveitados a partir de um ajuste melhor dos seus parâmetros.

Na definição do escopo do trabalho, pelo mesmo ser pioneiro no âmbito do LaPS da UFPA, optou-se por enfatizar a diversidade dos assuntos, ao invés de focalizar em alguns pontos. Em outras palavras, no que diz respeito à relação de compromisso entre *exploration* and *exploitation*, optou-se pela primeira. Assim, há diversos pontos que foram apenas superficialmente estudados, tais como HMM, a seleção da melhor *wavelet* e uma estimativa mais confiável da complexidade computacional dos algoritmos do WEKA.

Estudos em classificação fazem parte de um conjunto mais geral de tarefas de mineração de dados aplicado à séries temporais representando eventos de QEE. Investigações atuais também incluem agrupamento com DTW e regras de associação. Em todos esses casos, a avaliação dos algoritmos é problemática devido à falta de dados rotulados. As simulações com ATP ajudam a evitar este problema.

No que diz respeito ao lançamento da base de dados UFPAFaults e a revisão do assunto sobre classificação de faltas certamente ajudarão os especialistas em inteligência computacional a aplicarem seus algoritmos com mais propriedade no campo da QEE. Este trabalho estabeleceu alguns resultados básicos usando UFPAFaults, mas que não são conclusivos.

Os resultados experimentais indicam que as redes neurais e árvores de decisão apresentam uma melhor performance com relação a outros tipos de classificadores. Quando o objetivo é alcançar um bom nível de performance e robustez as redes neurais são atrativas. Já as árvores de decisão tornam-se interessantes a partir do momento que o objetivo é tentar minimizar o custo computacional, como, por exemplo, no desenvolvimento de dispositivos embarcados.

A classificação pós-falta carece de uma investigação mais aprofundada. A arquitetura de classificação seqüencia baseada em quadros FBSC é somente uma dentre as muitas opções de classificação. A adoção de novos parâmetros além das formas de onda de tensão e corrente e até mesmo um aumento na freqüência de amostragem para uma melhor caracterização das faltas ABC e ABCT talvez sejam necessárias para melhorar a performance dos classificadores, principalmente os que tratam diretamente com a forma de onda no caso HMM e segmental Knn. É interessante também que um classificador GMM, que apresente como saída log de likelihoods, obtenha uma grande discrepância entre E_s e E_f, enquanto que essa diferença para o classificador RNA é muito pequena. Trata-se de uma problemática relevante para se discutir futuramente.

Em resumo, os resultados obtidos até o presente momento mostram-se encorajadores e, as considerações acima citadas sobre *front ends*, tipos de normalização de dados e arquitetura de classificação servirão de ponto de partida para obtenção de resultados mais promissores.

5.2 Sugestões de trabalhos futuros

Como recomendações para trabalhos futuros pode-se citar:

- Utilizar os dados digitais disponíveis nos registros oscilográficos para testar a robustez dos classificadores.
- Aplicar outras técnicas de normalização como, por exemplo, obter o valor médio dos picos no intervalo pré-falta e utilizá-lo como base para transformação dos dados em *pu*.
- Adotar outros tipos de *front ends* tais como: calcular os valores rms (*Root Mean Square*) das formas de onda e em seguida concatená-los, calcular os valores rms ou as entropias dos coeficientes wavelet [21] ou aplicar técnicas como Transformada de Fourier [22].
- Utilizar outros classificadores como *Support Vector Machine*(SVM) [23] no processo de classificação de faltas.
- Estender a metodologia para classificação de outros eventos relacionados à qualidade de energia elétrica.

Referências Bibliográficas

[1] A. C. Alves. Algoritmos genéticos aplicados ao planejamento da distribuição de energia elétrica em curitiba e região metropolitana. Master's thesis, Universidade Federal do Paraná, Departamento de Matemática, 2002.

[2] http://htk.eng.cam.ac.uk/, Visitado em março, 2007.

[3] E. Franco. Qualidade de energia, disponível em: http://www.engecomp.com.br/, visitado em jun. 2007.

[4] J. R. Boisson de Marca. An LSF quantizer for the north-american half-rate speech coder. *IEEE Transactions on Vehicular Technology*, 43:413–419, 1994.

[5] S. Santoso and J.D. Power Lamoree. Power quality data analysis: From raw data to knowledge using knowledge discovery approach. In *Engineering Society Summer Meeting, IEEE*, pages 172–177, 2000.

[6] Slavko Vasilic. *Fuzzy Neural Network Pattern Recognition Algorithm for Classification of the Events in Power System Networks*. PhD thesis, Texas A&M University, 2004.

[7] Math H.J. Bollen. *Understanding power quality problems: Voltage sags and interruptions*. IEEE Press Series on Power Engineering, 2000.

[8] M. Silva, M. Oleskovicz, and D. V. Coury. Uma nova ferramenta baseada na transformada wavelet para localização digital de faltas. *Sba Controle & Automação*, 16, 2005.

[9] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2nd Edition, 2005.

[10] X. Luo and M. Kezunovic. Fault analysis based on integration of digital relay and dfr. *Power Engineering Society General Meeting*, 1:746 – 751, 2005.

[11] M. Kezunovic and I. Rikalo. Detect and classify faults using neural networks. *Computer Applications in Power*, 9:42–47, 1996.

[12] EMTP. *Alternative Transients Program (ATP) Rule Book*. Canadian/American EMTP User's Group, 1995.

[13] Yomara Pires, Andreia Santos, Jose Borges, Antonio Carvalho, Marcus Vinicius Alves Nunes, Surya Santoso, and Aldebaro Klautau. A framework for evaluating data mining techniques applied to power quality. In *Brazilian Conference on Neural Networks (CBRN)*, 2005.

[14] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. In *8th ACM SIGKDD International Conf. Knowledge Discovery and Data Mining.*, pages 102–111, 2002.

[15] S. Vasilic and M. Kezunovic. New design of a neural network algorithm for detecting and classifying the transmission line faults. In *IEEE PES Transmission and Distribution Conference*, 2001.

[16] S. Vasilic and M. Kezunovic. An improved neural network algorithm for classifying the transmission line faults. In *IEEE PES Winter Meeting*, 2002.

[17] S.Vasilic and M. Kezunovic. Fuzzy art neural network algorithm for classifying the power system faults. *IEEE Transactions on Power Delivery*, 20:1306–1314, 2005.

[18] N. Zhang and M. Kezunovic. A real time fault analysis tool for monitoring operation of transmission line protective relay. *Electric Power Systems Research*, 77:361–70, 2007.

[19] O. Dag and C. Ucak. Fault classification for power distribution systems via combined wavelet-neural approach. In *Internationl Conference on Power System Technology - POWERCON*, 2004.

[20] K. M. Silva, B.A. Souza, and N.S.D. Brito. Fault detection and classification in transmission lines based on wavelet transform and ann. *IEEE TRANSACTIONS ON POWER DELIVERY*, 21 no.4:2058– 2063, 2006.

[21] B. Zhang, H. Zhengyou, and Q. Qian. Application of wavelet entropy and adaptive nerve-fuzzy inference to fault classification. In *Power System Technology*, pages 1–6, 2006.

[22] D. Das, N. K. Singh, and A. K. Sinha. A comparison of fourier transform and wavelet transform methods for detection and classification of faults on transmission lines. *IEEE Power India Conference*, 2006.

[23] P. G. V. Axelberg, I. Y. H. Gu, and M. H. J. Bollen. Support vector machine for classification of voltage disturbances. *IEEE Trans. Power Delivery*, 22:1297 – 1303, 2007.

[24] Monedero. Classification of electrical disturbances in real time using neural networks. *IEEE Trans. Power Delivery*, 22:1288 – 1296, 2007.

[25] R. S. Ehlers. Análise de séries temporais. Technical report, Departamento de Estatística, UFPR, 2005.

[26] E. Passos and R. Goldschmidt. *Data Mining: um guia prático*. Campus, 2005.

[27] M. Vetterli and J. Kovačević. *Wavelets and Subband Coding*. Prentice Hall, 1995.

[28] I. N. Sneddon. *Fourier transforms*. New York: Dover, 1995.

[29] C.Aguilera, E.Orduna, and G.Rattá. Fault detection, classification and faulted phase selection approach based on high-frequency voltage signals applied to a series-compensated line. *IEEE Proc.- Gener. Transm. Distrib.*, 153:469–475, 2006.

[30] B. A. de Souza et al. Classificacao de faltas via redes neurais artificiais. In *V SBQEE*, pages 163–8, 2003.

[31] M. Oleskovicz, R. K. Aggarwal, and D. V. Coury. O emprego de redes neurais artificiais na detecção, classificação e localização de faltas em linhas de transmissão. *Sba Controle & Automação*, 14, 2003.

[32] R. Duda, P. Hart, and D. Stork. *Pattern classification*. Wiley, 2001.

[33] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, 2nd Edition, 1999.

[34] A. Braga, A. P. Carvalho, and T. B. Ludemir. *Redes Neurais Artificiais: teoria e aplicações*. LTC, 2000.

[35] J. R. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, pages 77 – 90, 1996.

[36] B. H. Juang and L. R. Rabiner. The segmental K-means algorithm for estimating parameters of hidden Markov models. *IEEE Transactions on Signal Processing*, 38:1639 – 1641, 1990.

[37] A. Klautau, N. Jevtić, and A. Orlitsky. Discriminative Gaussian mixture models: A comparison with kernel classifiers. In *ICML*, pages 353–360, 2003.

[38] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–86, Feb. 1989.

[39] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. on ASSP*, 26(1):43–49, 1978.

[40] H. Englert and J. Stenzel. Automated classification of power quality events using speech recognition techniques. In *14th Power Systems Computation Conference*, 2002.

[41] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics*, 41:164 – 171, 1970.

[42] L. Rabiner and B. Juang. *Fundamentals of speech recognition*. PTR Prentice Hall, Englewood Cliffs, N.J., 1993.

[43] Xiaopeng, E. Keogh, C. Shelton, and L. Wei. Fast time series classification using numerosity reduction. In *Proceedings of the 23nd international conference on machine learning*, 2006.

[44] J. Morais, Y. Pires, C. Cardoso, and A. Klautau. Data mining applied to the electric power industry: Classification of short-circuit in transmission lines. In *IEEE International Conference on Intelligent Systems Design and Applications (ISDA)*, 2007.

[45] A. Carvalho. Metodologia para estudos de sobretensões originadas por manobras em sistemas elétricos de potência. Master's thesis, Universidade Federal do Pará, Departamento de Engenharia Elétrica, 2007.

[46] C.E. Morais and L.C. Zanetta Jr. Fault location in transmission lines using one-terminal postfault voltage data. *IEEE Trans. Power Delivery*, 19:570–575, 2004.

[47] A. McEachern. A floating-window algorithm for detecting certain power line faults that disrupt sensitive electronic loads. *Instrumentation and Measurement, IEEE Trans.*, 39:112–115, 1990.

[48] J. Liang, S. Elangovan, and J. B. X. Devotta. A wavelet multiresolution analysis approach to fault detection and classification in transmission lines. *International Journal of Electrical Power and Energy Systems*, 20:327–332, 1998.

[49] A. Gaouda, M. Salama, and M. Sultan. Power quality detection and classification using wavelet-multiresolution signal decomposition. *IEEE Trans. Power Delivery*, 14:1469 – 1476, 1999.

[50] M. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, University of Waikato, 1999.

Apêndice A

Parâmetros dos classificadores

Os parâmetros dos classificadores são designados pelo procedimento de seleção automática de modelos. Para este procedimento, foi desenvolvida uma classe em Java, denominada de ParameterSelectionUsingTestFile. O código fonte desta classe e uma descrição de como adicioná-la ao pacote WEKA podem ser obtidas em *http://www.ufpa.br/freedatasets/UfpaFaults/scripts*.

O WEKA também possui uma classe que realiza a seleção automática de modelos denominada de CVParameterSelection. As classes ParameterSelectionUsingTestFile e CVParameterSelection diferem em dois aspectos, como explicado a seguir.

Na classe CVParameterSelection utiliza-se o próprio conjunto de teste para realizar a seleção do modelo, o que pode provocar *overfitting* caso os vetores sejam extraídos de uma mesma falta (veja Seção 3.3.3 para mais detalhes), enquanto que a ParameterSelectionUsingTestFile usa um arquivo disjunto dos arquivos de treino e teste, denominado de validação.

Outro aspecto é com relação à combinação dos possíveis parâmetros dos classificadores. Na classe CVParameterSelection os parâmetros do grid são obtidos a partir de um incremento linear usando-se a opção -P da classe. Já na classe ParameterSelectionUsingTestFile os parâmetros do grid podem ser obtidos tanto a partir de um incremento linear (usando-se adição), representado pela opção -P, quanto de um geométrico (usando-se multiplicação), representado pela opção -Q.

Este trabalho utilizou a classe ParameterSelectionUsingTestFile para gerar o grid de alguns parâmetros dos classificadores.

Para a rede neural foi utilizada a classe FastNeuralNetwork não pertencente ao pacote WEKA, disponível em *http://www.laps.ufpa.br/aldebaro/weka*. Os parâmetros variados desta classe foram: a taxa de aprendizagem -L e a taxa de momentum -M. O parâmetro -L variou de

acordo com um incremento linear (-P “L 0.2 0.9 8”), enquanto que o parâmetro -M a partir de um incremento geométrico (-Q “M 0.1 0.9 3”). O número de neurônios na camada escondida, representado pelo parâmetro -H na classe FastNeuralNetwork, foi especificado pela opção “a” (-H a) que corresponde ao (números de atributos + número de classes)/2. O número de iterações -N foi fixado arbitrariamente em 1500.

Em relação à árvore de decisão foi utilizada a classe do WEKA, denominada de J4.8. Os parâmetros -C (fator de confiança para poda) e -M (o número mínimo de exemplos por folha) desta classe foram variados, respectivamente, a partir de um incremento linear (-P “C 0.35 0.94 4 ”) e um incremento geométrico (-Q “M 10 270 3”).

Assim como para a rede neural, para o classificador misturas de Gaussianas (GMM) foi utilizada uma classe, denominada de GaussiansMixture (*http://www.laps.ufpa.br/freedatasets/ufpafaults/scripts*) que não faz parte do pacote WEKA. Os parâmetros -I (número de Gaussianas) e -C (valor mínimo para os elementos da matriz de covariância) da classe GaussiansMixture variaram, respectivamente, de acordo com um incremento geométrico (-Q “I 1 10 2”) e linear (-P “C 0.1 0.9 3”).

A classe IBK do WEKA foi utilizada para implementar o classificador K-Nearest Neighbors (KNN). O parâmetro número de vizinhos mais próximos, representado pelo parâmetro -K nesta classe, variou de acordo com um incremento linear (-P (“K 1 7 4”)).

Front end	L ou L_{min}	S ou S_{min}	K	**RNA**	**J4.8**	**GMM**
Raw	1	1	6	-H 8 -N 1500 -L 0.2 -M 0.3	-C 0.35 -M 10	-I 8 -C 0.1
	5	5	30	-H 20 -N 1500 -L 0.2 -M 0.3	-C 0.5467 -M 10	-I -C 0.1
	7	7	42	-H 26 -N 1500 -L 0.2 -M 0.3	-C 0.7433 -M 10	-I 7 -C 0.1
	9	9	54	-H 32 -N 1500 -L 0.2 -M 0.3	-C 0.35 -M 10	-I 8 -C 0.1
	11	11	66	-H 38 -N 1500 -L 0.2 -M 0.3	-C 0.5467 -M 10	-I 1 -C 0.1
	33	33	198	-H 104 -N 1500 -L 0.2 -M 0.3	-C 0.35 -M 10	-I 1 -C 0.1
Wavelet-concat	4	2	192	-H 100 -N 1500 -L 0.2 -M 0.2	-C 0.5467 -M 10	-I 1 -C 0.9
	5	2	240	-H 125 -N 1500 -L 0.2 -M 0.3	-C 0.35 -M 10	-I 1 -C 0.9
	7	3	336	-H 173 -N 1500 -L 0.2 -M 0.3	-C 0.54 -M 10	-I 1 -C 0.5
	9	4	486	-H 248 -N 1500 -L 0.5 -M 0.4	-C 0.54 -M 10	-I 1 -C 0.9
Wavelet-energy	1	1	24	-H 17 -N 1500 -L 0.2 -M 0.3	-C 0.5467 -M 10	-I 4 -C 0.1

Tabela A.1: Resumo do conjunto dos parâmetros para os classificadores.

O algoritmo de agrupamento K-means foi utilizado com objetivo de reduzir o tamanho da base de dados de treino, e com isso diminuir o custo computacional do KNN. A classe do WEKA que implementa esse algoritmo é denominada de SimpleKMeans. O número de *clusters* (centros), especificado pelo parâmetro -N nesta classe, foi variado sem o auxílio da seleção automática de modelos. Os valores arbitrariamente especificados para este parâmetro

foram iguais a 121, 220, 550, 990 e 1111.

Um resumo do conjunto dos parâmetros para os classificadores pode ser encontrado na Tabela A.1, onde L e L_{min} indicam, respectivamente, o número de amostras (tamanho) do quadro (*frame*) para o *front end raw* e o tamanho do quadro para os coeficientes *wavelet* que têm menor freqüência de amostragem (f_s) (*front ends waveletconcat e waveletenergy*). S e S_{min} representam, respectivamente, o deslocamento do quadro (*front end raw*) e o deslocamento do quadro para os coeficientes *wavelet* que têm menor f_s. K indica a dimensão de entrada.

Os parâmetros dos classificadores não citados assumiram valores padrões fornecidos pelas classes. O parâmetro -K do classificador KNN foi igual a 1 e o parâmetro -N do K-means foi igual a 1111 para todos os *front ends*.

Apêndice B

Scripts utilizados

Todos os scripts aqui mencionados podem ser obtidos no endereço eletrônico *www.laps.ufpa.br/freedatasets/UFPAFaults/scripts*.

1 - Pré-processamento

A classe que implementa o pré-processamento (PreProcessor.java) possui os seguintes parâmetros: o diretório com o arquivo labels, o qual armazena informações sobre as simulações; a razão sinal ruído (Se igual 0 não se adiciona ruído); o fator de decimação; o número de ciclos que devem ser coletados antes e depois da falta; o tipo de normalização (0 para não normalizar, 1 para normalização *prefault* e 2 para normalização *allfault*); e o último parâmetro permite ao usuário definir se serão coletadas somente as seqüências no intervalo da falta (true). Em seguida temos um exemplo de como executar a classe.

```
java scripts.PreProcessor . 30 20 0 1 true > preprocessor.log
```

A classe PreProcessor.java realiza a leitura dos arquivos de simulações em formato txt e transforma-os em arquivos binários (.bin) com os dados pré-processados. Tais arquivos binários são nomeados de acordo com as informações contidas no arquivo labels. Por exemplo, se no arquivo labels temos uma linha indicando que o número da simulação é 20, a base de dados é a Tramoeste, a linha 00 foi onde ocorreu a falta e a falta é do tipo AB, então o nome do arquivo binário é: tramoeste_00_ab_20.bin.

2 - *Front ends*

Para aplicar o *front end Raw* nos dados das simulações utiliza-se a seguinte linha de comando:

```
java scripts.Raw . bin 9 true true > arffile.arff
```

onde [.] indica que o diretório de entrada é diretório corrente, bin a extensão dos arquivos, 9

o tamanho do quadro L e as opções true e true indicam, respectivamente, que os valores de tensão e corrente devem ser coletados, logo o número de sinais $Q = 6$ e a dimensão de entrada $K = 54$. Note que a classe Raw.java fornece um arquivo no formato arff do WEKA a partir do comando de redirecionamento de saída >.

No caso do *front end waveletconcat* executa-se a classe WaveletDecomposition.java para calcular os coeficientes wavelets e concatená-los:

java scripts.WaveletDecomposition . bin db4 3 2000 4 2 > arffile.arff

onde indica que o diretório de entrada é diretório corrente, bin a extensão dos arquivos de entrada, db4 é o nome da wavelet, 3 é o número de níveis de decomposição, 2000 é freqüência de amostragem e 4 e 2 são, respectivamente, o tamanho mínimo do quadro L_{min} e o deslocamento mínimo do quadro S_{min}. Note que o script gera dois tipos de arquivos: para cada arquivo bin é gerado um arquivo .coefs com os coeficientes *wavelet* concatenados e a partir do comando de redirecionamento de saída > é gerado um arquivo ASCII no formato do WEKA.

O *front end waveletenergy* é implementado pela classe WaveletPower.java e executado a através da linha de comando:

java scripts.WaveletPower . bin db4 3 2000 > arffile.arff

onde [.] e bin são, respectivamente, o diretório (corrente) e a extensão dos arquivos de entrada, db4 é o nome da *wavelet*, 3 é número de níveis de decomposição e 2000 é a freqüência de amostragem. Note que a classe WaveletPower gera dois tipos de arquivos: para cada arquivo bin é gerado um arquivo .ene com a energia dos coeficientes *wavelet* e a partir do comando de redirecionamento de saída > é gerado um arquivo ASCII no formato do WEKA.