Projeto Final

Machine Learning Engineer
Nanodegree

Gabriel Costa

Outubro, 2016

Detecção de Falhas em Circuitos Elétricos

Definição

Visão Geral do Projeto

O desenvolvimento de estratégias de teste para detectar e diagnosticar falhas em circuitos analógicos e de sinais mistos é uma tarefa desafiadora que tem encorajado uma boa quantidade de pesquisas, devido ao aumento do número de aplicações destes circuitos e ao alto custo dos testes. Muitas áreas, tais como, telecomunicações, multimídia e aplicações biomédicas, precisam de bom desempenho em aplicações de alta frequência, baixo ruído e baixa potência, o que só se pode ser alcançado através do uso de circuitos integrados analógicos e de sinais mistos. Assim, uma estratégia para detectar e diagnosticar falhas nesse tipo de circuitos é muito importante (Albustani, 2004). No passado, um circuito integrado era apenas um componente em um sistema, mas hoje o circuito integrado em si é o sistema inteiro (SoC - *system on a chip*). Com esse nível de integração, problemas difíceis de teste e projeto foram gerados. Dentre os vários fatores que aumentam as dificuldades, pode-se citar: a falta de bons modelos de falhas, falta de um padrão de projeto com vistas à facilidade de testes e o aumento da importância das falhas relacionadas ao tempo de vida dos componentes (Claasen, 2003). Assim, a estratégia de testes para detecção e diagnóstico de falhas ainda é severamente dependente da perícia e da experiência que os engenheiros têm sobre as características do circuito. Então a detecção e a identificação de falhas ainda são um processo interativo e que consome bastante tempo. Um estudo na área de detecção e diagnóstico (Fenton, 2001) mostrou que, nas últimas décadas, uma boa quantidade de pesquisa em diagnósticos de falhas foi concentrada em desenvolver ferramentas que facilitassem as tarefas de diagnóstico. Embora progressos importantes tenham sido alcançados, essas novas tecnologias não têm sido largamente aceitas. Isso deve motivar os pesquisadores a investigar outros paradigmas e desenvolver novas estratégias para diagnósticos de falhas.

O uso de técnicas de inteligência computacional para diagnóstico é normalmente baseado na construção de modelos ou no uso de classificadores, cujo sucesso e desempenho depende da qualidade do modelo obtido, o que, no caso de um sistema complexo, pode ser difícil de obter. Os classificadores multiclasse procuram por comportamentos específicos de falhas e se mostram vulneráveis a superposição de padrões de falha ou a padrões de falha que não foram apresentados durante a fase de treinamento. Classificadores de classe única podem ser treinados para resolver problemas de classificação binária onde apenas uma das classes é bem conhecida (Tax, 2001). Estes podem ser organizados na forma de conjunto

(ensemble) de classificadores e com isso reduzir alguns dos problemas encontrados com classificadores multiclasse citados anteriormente. Esta proposta de projeto apresenta o esboço do desenvolvimento de um sistema de detecção de falhas em circuitos lineares e invariantes no tempo, onde os resultados de diferentes métodos serão comparados para a determinação do melhor tipo de classificador.

Descrição do Problema

O termo *falha* é definido como uma condição anormal ou defeito (ISO/CD 10303), em um componente, equipamento ou sistema que pode conduzir ao mau funcionamento, isto é, uma diminuição parcial ou total na capacidade de desempenhar a função desejada por certo intervalo de tempo.

Em circuitos analógicos, as falhas podem ser classificadas usando diferentes critérios. O tipo de falha abordada neste projeto é aquela que se dá em função do desvio do parâmetro de um sistema (ou componente do sistema) no tempo, designada falha *paramétrica*, forçando-o a assumir um valor que está fora de sua faixa nominal. Quando existe um desvio repentino muito grande do valor do parâmetro desejado, este é chamado de falha catastrófica. Este tipo de falha está associado à mudança da estrutura do sistema. Alguns exemplos de falhas catastróficas em circuitos elétricos seriam o circuito aberto e o curto-circuito (DUHAMEL E RAULT, 1979).

Métricas

Como a alteração causadora da falha no circuito é previamente determinada, é possível usar um "gabarito" para qualificar e quantificar a precisão do modelo de predição. Desta forma é possível empregar as métricas do próprio scikit-learn, como o fbeta_score.

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

O F-beta score é um método de avaliação de precisão que representa o desempenho do modelo de predição com maior precisão por considerar não apenas os acertos e a quantidade total de elementos, mas também os falsos positivos e os falsos negativos.

A matriz de confusão foi plotada após a aplicação de cada método a fim de comparar e avaliar o resultado do próprio F-beta score.

Análise

Exploração e Visualização de Dados

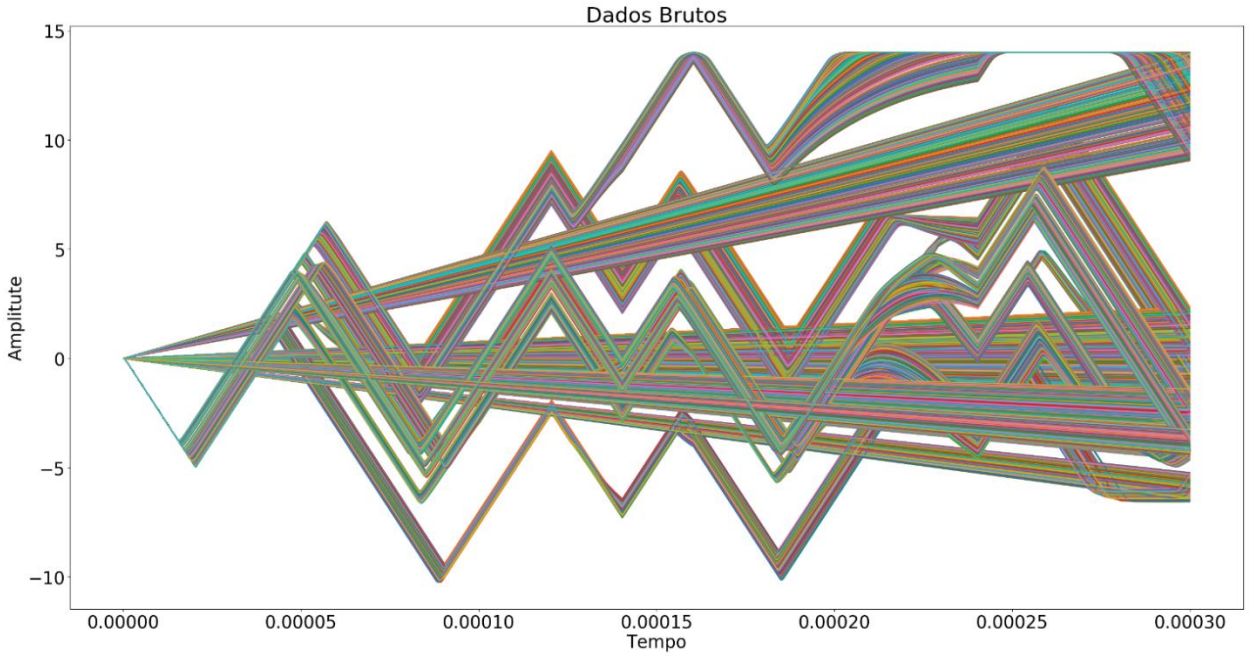
Os dados de simulação ordinários do LTSpiceIV são salvos em um arquivo ".raw" que contém informações de todas as grandezas do circuito para todos os passos de simulação, bem como dos passos de tempo referente aos passos de simulação, um cabeçalho com título do arquivo de simulação e outros metadados, e até alguns dados criptografados. Segue um trecho do arquivo .raw referente ao circuito Nonlinear Rectfier:

```
Title: * <caminho do sistema>\Nonlinear Rectfier + 4bit PRBS [FALHA] - 300 - 0.2s.asc
Date: Sat Oct 06 16:15:11 2018
Plotname: Transient Analysis
Flags: real forward stepped
No. Variables: 32
No. Points:     3325566
Offset:   0.0000000000000000e+000
Command: Linear Technology Corporation LTspice IV
Variables:
        0       time    time
        1       V(vin)  voltage
        2       V(n001)         voltage
        3       V(v1)   voltage
        4       V(vout)         voltage
        ...
        29      I8(A1)  device_current
        30      I7(A1)  device_current
        31      I6(A1)  device_current
Binary:
    ¶'¤¡SØKŽ›ÏKŽ› + [1352054 linhas de dados binários representados, quando possível, em ascii]
```

O arquivo raw é bem completo e este, por exemplo, possui cerca de 429MBytes de informação. Assim seria necessário um projeto só para extrair as informações necessárias do arquivo, e então foi usado um toolchain existente para a extração dos dados. Através do "LTSpiceRaw_Reader.py", as informações pertinentes extraídas do arquivo raw foram obtidas e salvas em um arquivo ".csv" de tamanho menor (65MBytes) e já no formato de dataframe, que será usado ao longo do projeto:

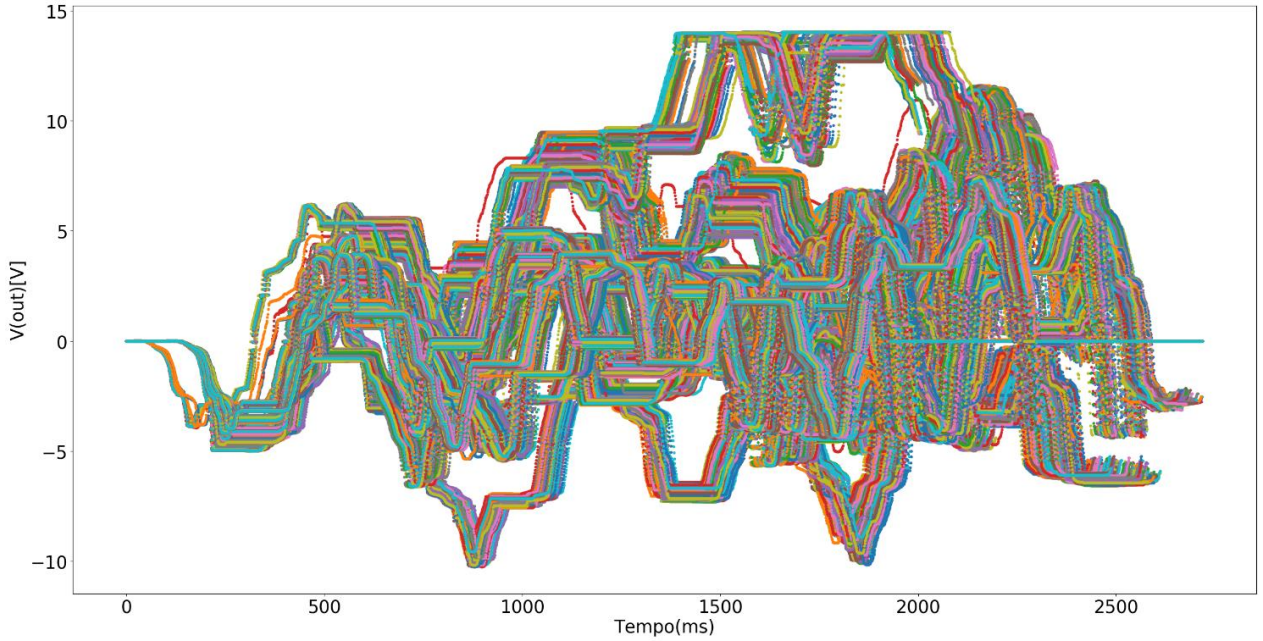
	0	1	2	...	3297	3298	3299
0	-2,35E-06	1,77E-06	1,01E-06	...	-8,97E-08	-2,17E-07	-2,48E-07
...	...	...	...	...	...	...	...
16	-2,35E-06	1,78E-06	1,01E-06	...	-8,37E-09	-2,11E-06	-2,42E-06
18	-2,35E-06	1,79E-06	1,01E-06	...	7,16E-03	7,16E-02	9,12E-03
19	1,20E-01	1,85E-06	1,02E-07	...	5,69E-01	5,69E-01	5,75E-01
20	9,59E-03	1,36E-04	1,68E-01	...	4,53E+00	4,53E+00	4,57E+00

O gráfico a seguir traz a plotagem dos dados da saída da simulação sobre os quais serão aplicados os métodos de aprendizagem:



(Dados originais extraídos do circuito Biquad Highpass Filter)

E os dataframe populado com estes dados possui a seguinte característica:



(Valores de Vout de cada passo de simulação plotados individualmente, do circuito Biquad Highpass Filter)

Através destes gráficos (para este e demais circuitos) podemos observar que existem alguns valores que aparentam estar fora do comportamento padrão das demais simulações. Porém, devo ressaltar que estes são dados resultantes de simulações de falhas elétricas e, portanto, estes não devem ser tratados como outliers pois carregam informações importantes sobre o comportamento das falhas.

Algoritmos e Técnicas

Para este conjunto de dados é esperado que métodos de aglomeração por semelhança de vizinhança, ensemble, árvores de decisão, regressão linear e *support vector machines* alcancem um bom desempenho. Por este motivo diversos algoritmos foram implementados neste projeto, pertencentes a diversas famílias de modelos, para fins de comparação.

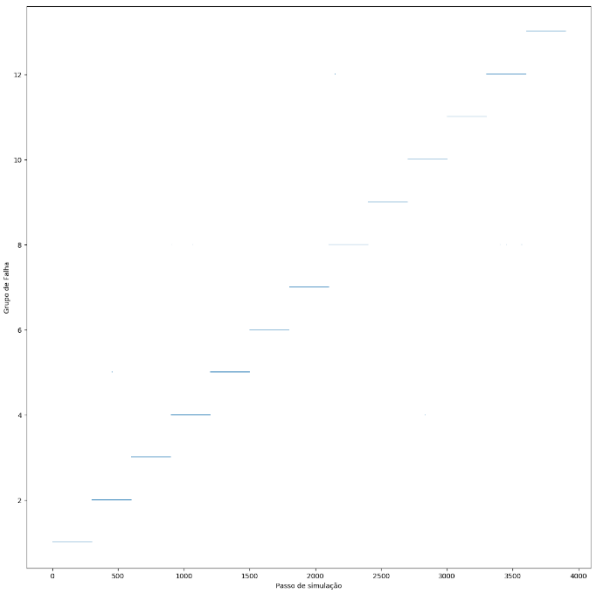
A precisão será avaliada através do F-beta score, e o tempo que cada algoritmo leva para tratar os dados por completo também será observado.

Benchmark

Os circuitos são simulados inicialmente em estado de funcionamento normal, e cada estado de falha é causado forçadamente, isto é, existe uma função no LTSpiceIV que causa a alteração dos parâmetros dos componentes de forma ordenada, prevista. Desta forma, para um dado conjunto de passos de simulação é possível prever qual foi o componente causador da falha, e qual o valor que levou àquela falha.
No caso específico do circuito "Sallen Key mc + 4bitPRBS [FALHA]" o algoritmo gerador das falhas em tempo de simulação é:

```
ac dec 100 10k 1meg
.step param run 1 3300 1
.tran 300us
.function falhaR1(baixo,alto,mc) if((run>X)&(run<=2*X), alto,if (run<=X,baixo,mc))
.function falhaR2(baixo,alto,mc) if((run>3*X)&(run<=4*X), alto,
if ((run<=3*X)&(run>2*X),baixo,mc))
.function falhaR3(baixo,alto,mc) if((run>5*X)&(run<=6*X), alto,
if ((run<=5*X)&(run>4*X),baixo,mc))
.function falhaC2(curto,aberto,normal) if((run>9*X)&(run<=10*X),aberto,
if((run<=9*X)&(run>8*X),curto,normal))
.function falhaC1(curto,aberto,normal) if((run>7*X)&(run<=8*X),aberto,
if((run<=7*X)&(run>6*X),curto,normal))
.param X=300
```

Com o objetivo de obter um retorno visual, de avaliação humanamente imediata, do desempenho da solução, os resultados dos treinos e das classificações serão plotados. O gabarito com estes para a comparação avaliação destes resultados se assemelha a uma escada, onde os degraus são os grupos de falhas. A imagem a seguir apresenta o gabarito para o circuito Biquad Highpass Filter mc + 4bitPRBS [FALHA].



(Figura: Gabarito gráfico do circuito Biquad Highpass Filter mc + 4bitPRBS [FALHA])

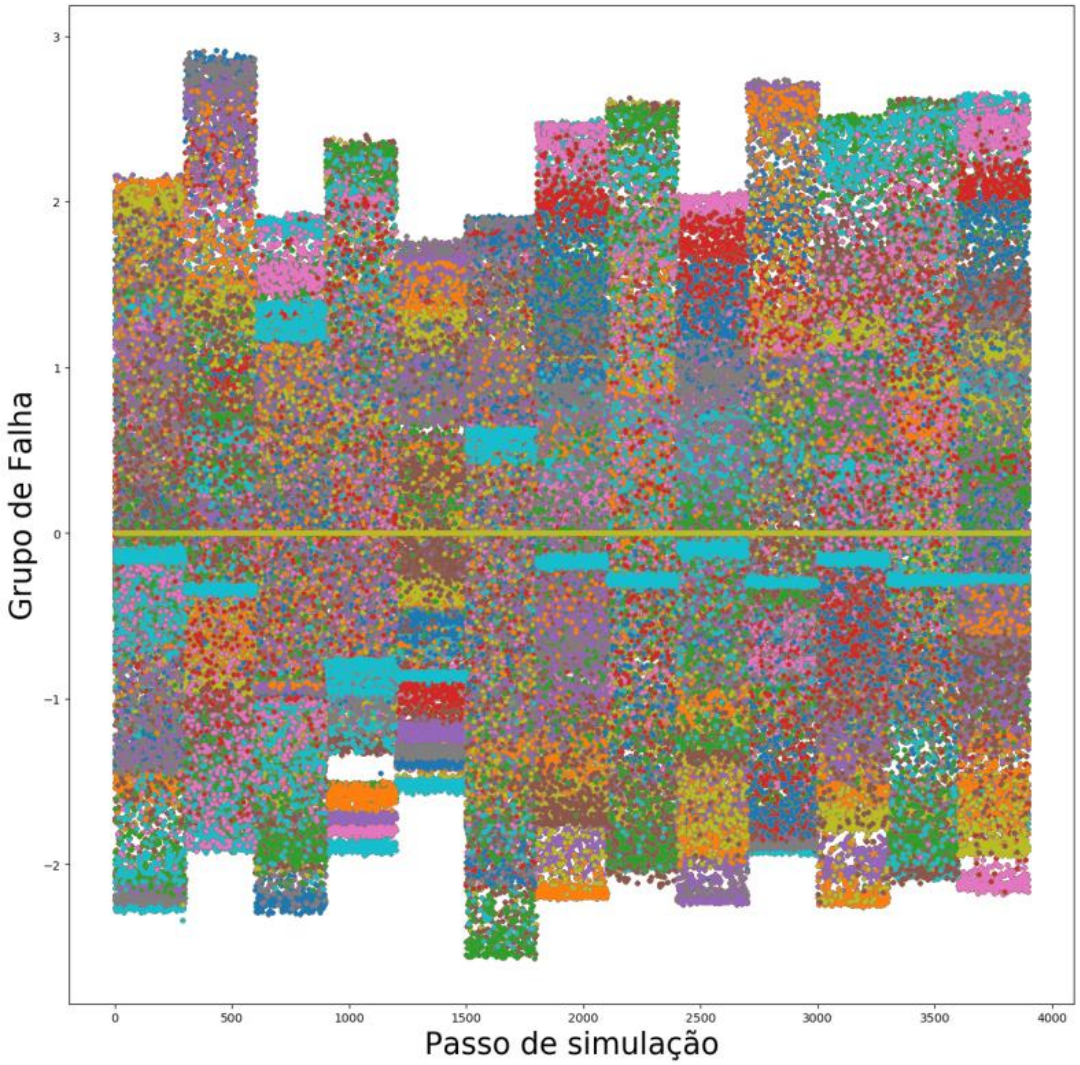
Metodologia

Pré-processamento de Dados

O pré-processamento dos dados se resumiu à extração de dados do objeto retornado após a leitura do arquivo ".raw". Dentre todos os dados fornecidos foi necessário retirar, especificamente, o "trace" referente a grandeza de interesse (no caso, tensão de saída) e organizar cada elemento do trace (isto é, passos de simulação dentro de cada simulação) em uma célula de um dataframe.

Não foi necessário remover outliers ou normalizar dados, nem recuperar dados perdidos, embora tenha sido aplicado o PAA (Piecewise Aggragate Aproximation) para reduzir o tempo de processamento através da redução da quantidade de informações suficientemente semelhantes.

O dataframe obtido após a aplicação do PAA possui o seguinte comportamento:



(Valores de Vout de cada passo de simulação plotados individualmente, do circuito Biquad Highpass Filter)

Na figura acima a separação das simulações por grupos de falhas se torna bem evidente.

Neste dataframe não há dados categóricos, todos os dados são números reais (float), logo não há a necessidade de conversão de dados, e os métodos de classificação podem ser aplicados diretamente sobre eles.

Implementação

Este projeto tem uma visão geral de aplicação e funcionamento bem simples e direta:

1. Em primeiro lugar é feita a leitura de dados:
    - Se for a primeira vez que o código for aplicado a um circuito, os dados são obtidos diretamente do arquivo raw exportado pelo LTSpiceIV;
    - Caso contrário, os dados são lidos do arquivo csv exportado anteriormente pelo próprio código;
2. Os dados são submetidos ao PAA;
3. Os dados do ao PAA são submetidos aos métodos de classificação;
4. Um classificador escolhido é submetido à otimização de parâmetros;
5. O classificador otimizado é aplicado aos dados originais;
6. Os resultados são salvos.

Como o objetivo aqui é comparar o desempenho de diversos métodos de classificação para poder escolher qual o melhor para a resolução do problema, o código foi estruturado de forma reutilizável, sendo separado em dois arquivos principais:

- AuxiliaryFunctions.py: que contém código reutilizável e trabalho pesado, com foco em processamento e organização de dados.
- Main.py: que contém a estruturação do fluxo de processamento e divulgação de resultados.

Aqui vale a pena ressaltar as classes:

- LTSpiceReader(): realiza a seleção dos dados que devem ser extraídos do arquivo raw de simulação pelo LTSpiceRawReader, plota os dados brutos e retorna os dados da importantes da simulação para o escopo principal;
- ApplyPaa(): prepara e submete os dados ao PAA;
- SupervisedPreds(): segmenta os dados entre grupos de treino e teste, aplica os métodos ou otimiza, conforme solicitado, e gera a matriz de confusão;

Os métodos aplicados foram:

- DecisionTreeClassifier;
- AdaBoostClassifier;
- SVC;
- RandomForestClassifier;
- GaussianNB;
- kNeighborsClassifier;
- SGDClassifier;
- AdaBoostClassifier com RandomForestClassifier como classificador base;
- LogisticRegression;
- BaggingClassifier;

Todos os métodos foram usados “out of the box”, ou seja, sem configuração prévia de Parâmetros, exceto pelo parâmetro random_state quando presente, e exceto pelo AdaBoost com RandomForest como base, para fins de comparação. O conjunto de teste foi definido como um total de 25% do total de dados para todos os métodos.

Refinamento

Como a disposição e tipo dos dados facilitam a classificação em diversos métodos, como os métodos não compartilham completamente dos mesmos parâmetros, e com a finalidade de exaltar a importância da otimização de parâmetros, o método de pior desempenho foi escolhido manualmente após a aplicação automática dos métodos, e sua otimização foi incorporada ao código.

A otimização deste método foi feita através da aplicação do GridSearchCV, que realiza a busca exaustiva sobre valores especificados de parâmetros de um estimador.

Uma grade de parâmetros em forma de dicionários é passada para um objeto do GridSearchCV, como por exemplo:

```
param_grid = [{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
              {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
             ]
```

Que especifica que duas grades de parâmetros devem ser exploradas:

- Uma com ‘kernel’ linear e valores de C na faixa [1,10,100,1000]
- E outra com ‘kernel’ RBF e o produto cruzado de C na faixa [1,10,100,1000] e valores de ‘gamma’ na faixa na faixa dentre os valores [0.001, 0.0001];

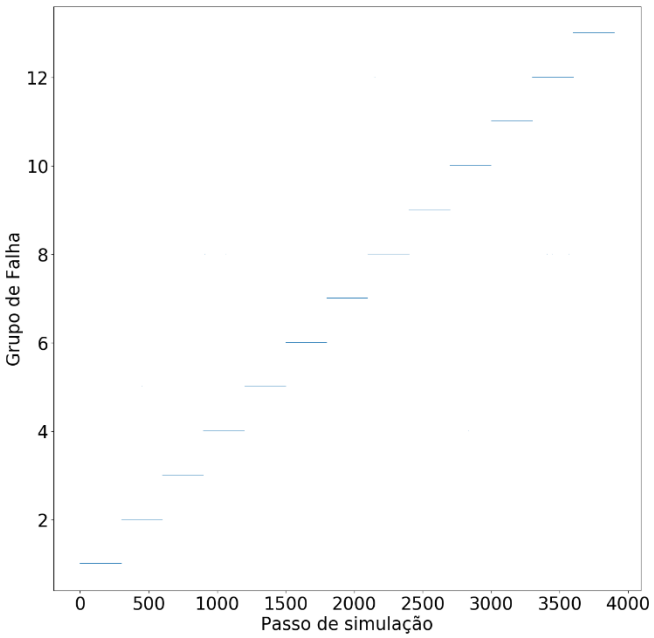
A instância do GridSearchCV implementa a API comum do estimador: quando “encaixando” em um conjunto de dados todas as combinações possíveis dos valores dos parâmetros são avaliadas e a melhor cominação e retida.

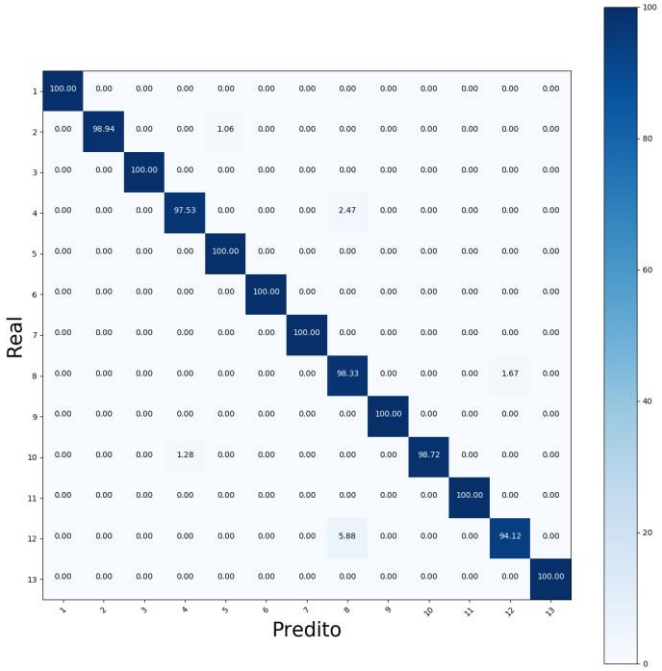
Resultados

Validação e Avaliação de Modelo

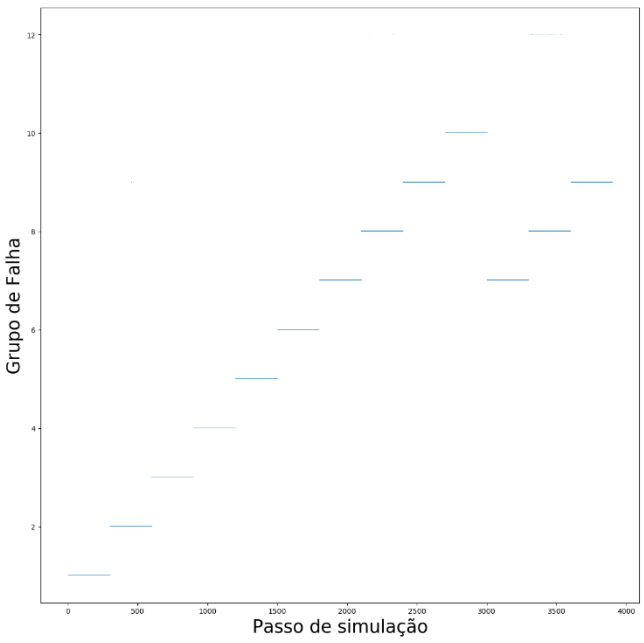
O código gera mais de 150 figuras como saída, então para fins práticos somente os gráficos referentes ao circuito Biquad Highpass Filter serão apresentados. Os resultados para os outros circuitos podem ser encontrados em:
https://drive.google.com/drive/folders/1aj-4SWuYvI32WoE9ZGpv3DejWCLAckw8?usp=sharing

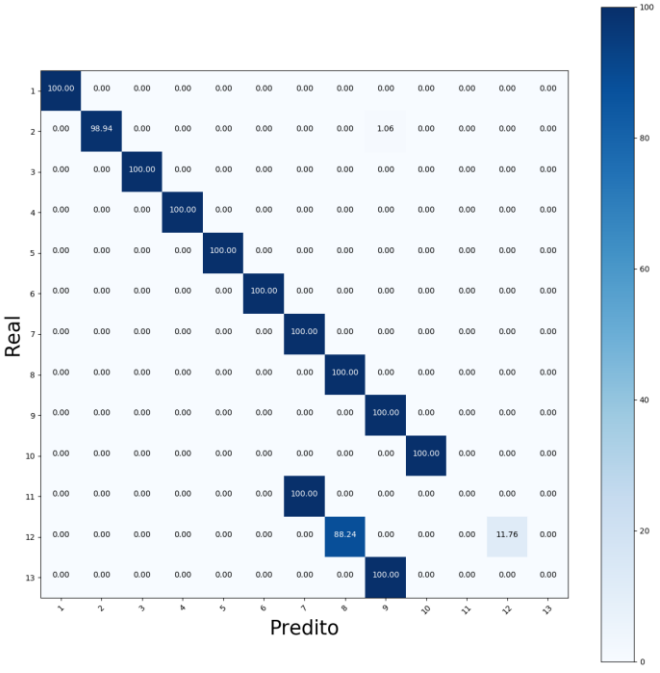
1. Resultados do Decision Tree Classifier:
    - Score: 98,98%;
    - Tempo de processamento: 5,99 segundos;



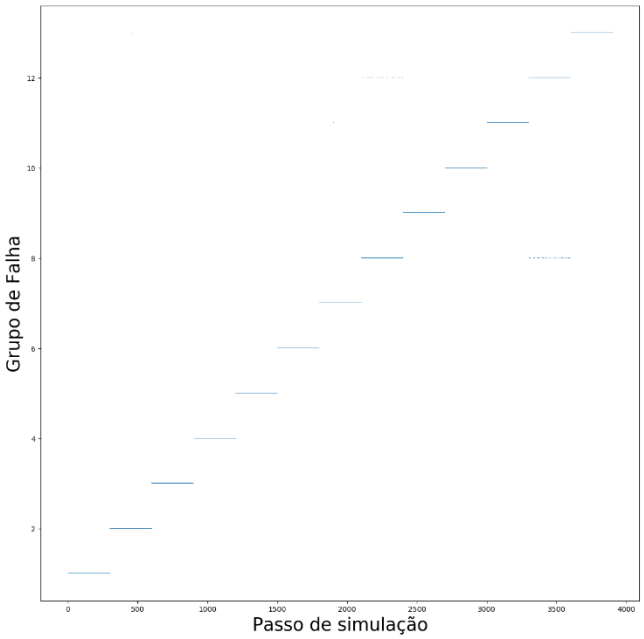


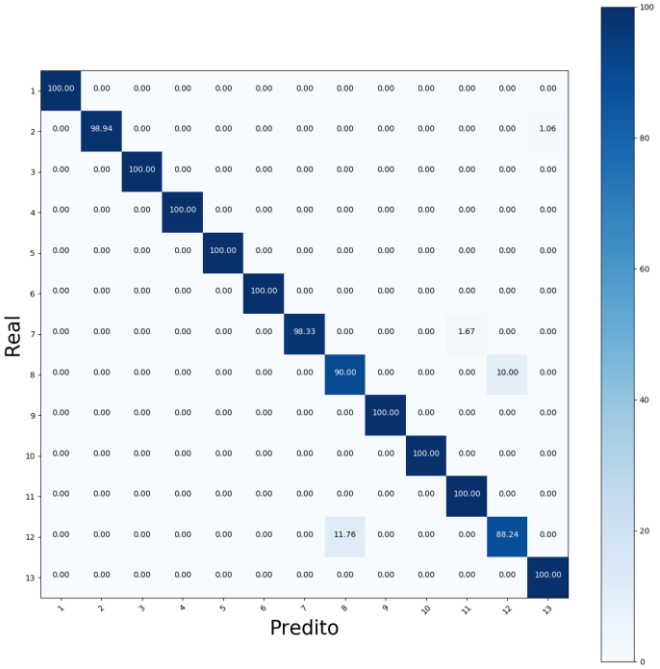
2. Resultados do AdaBoost Classifier;
    - Score: 68,81%;
    - Tempo de processamento: 54,72 segundos;



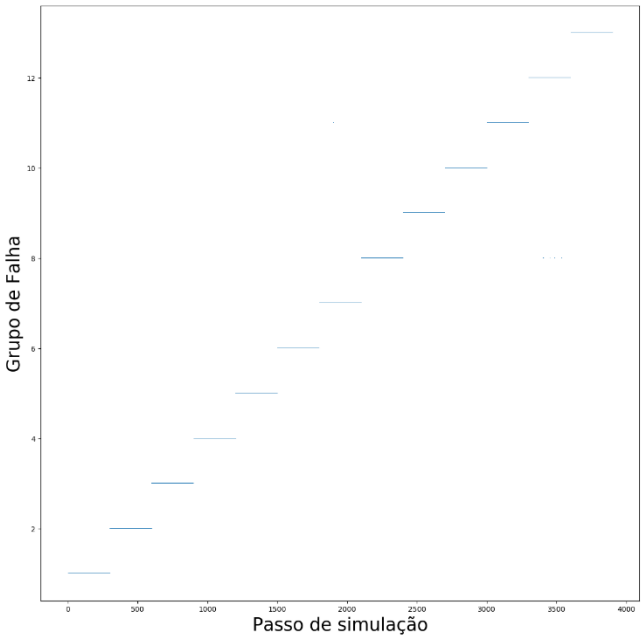


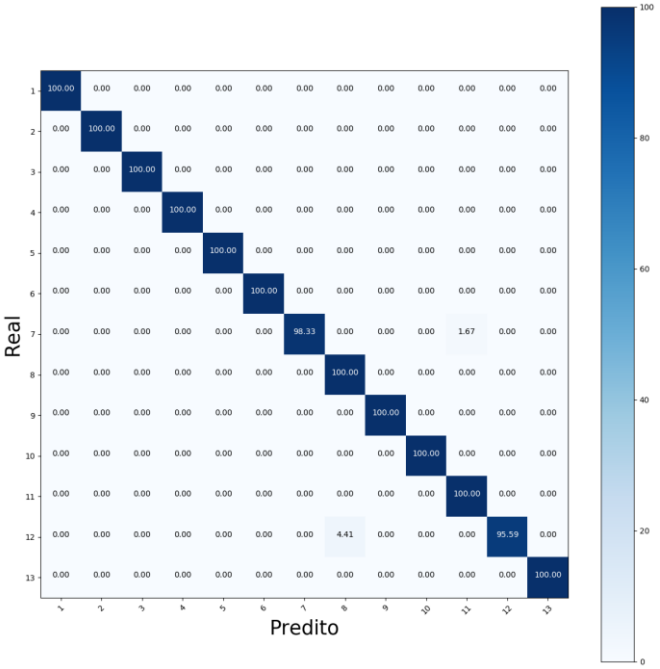
3. Resultados do SVC:
    - Score: 98,12%;
    - Tempo de processamento: 10,82 segundos;



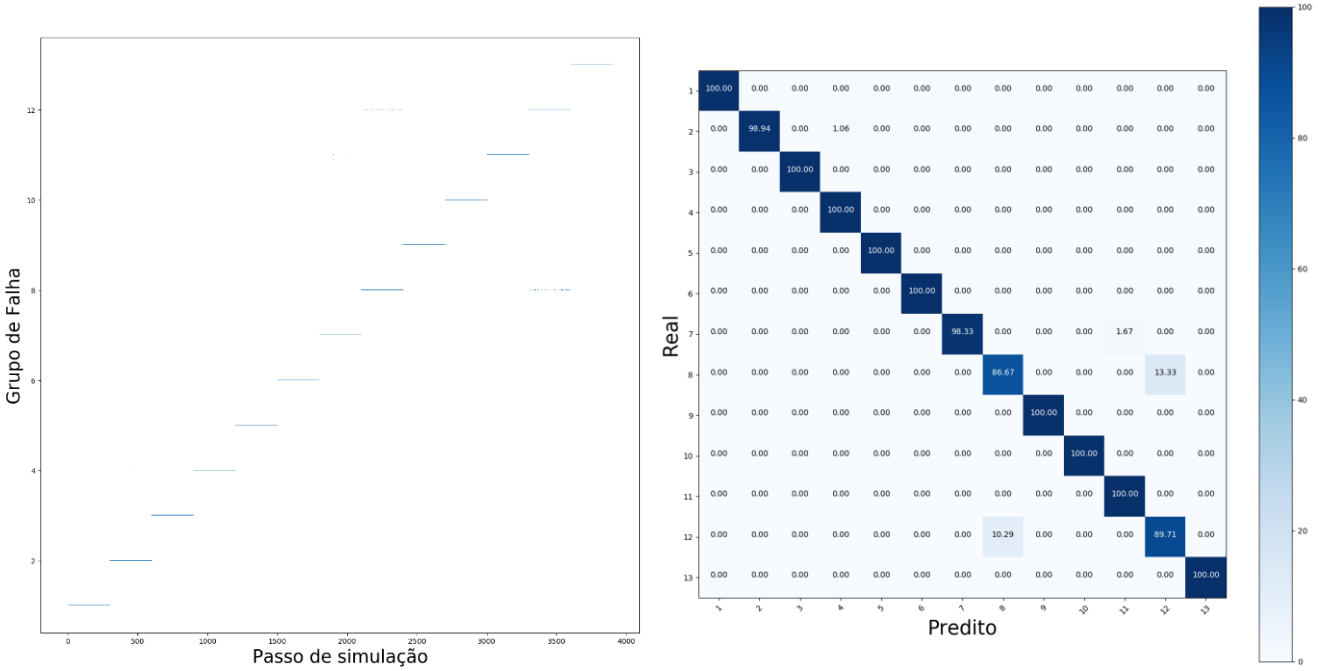


4. Resultados do Random Forest Classifier:
    - Score: 99,53%;
    - Tempo de processamento: 4,32 segundos;

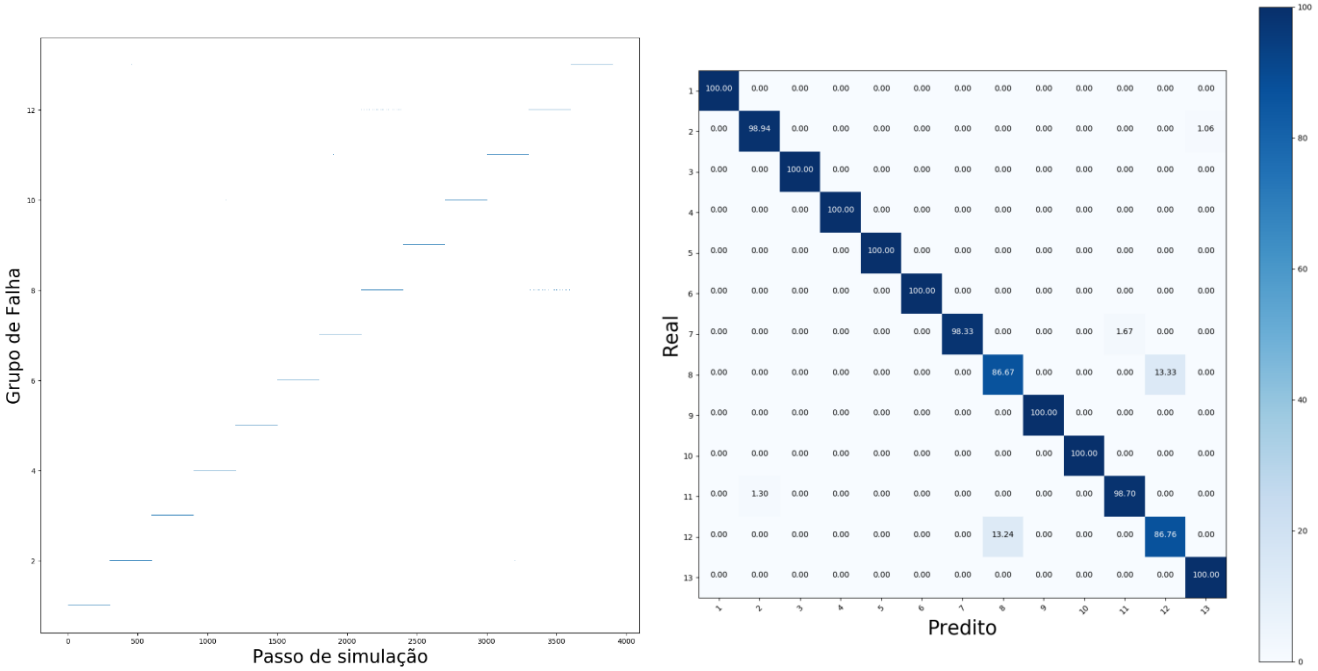




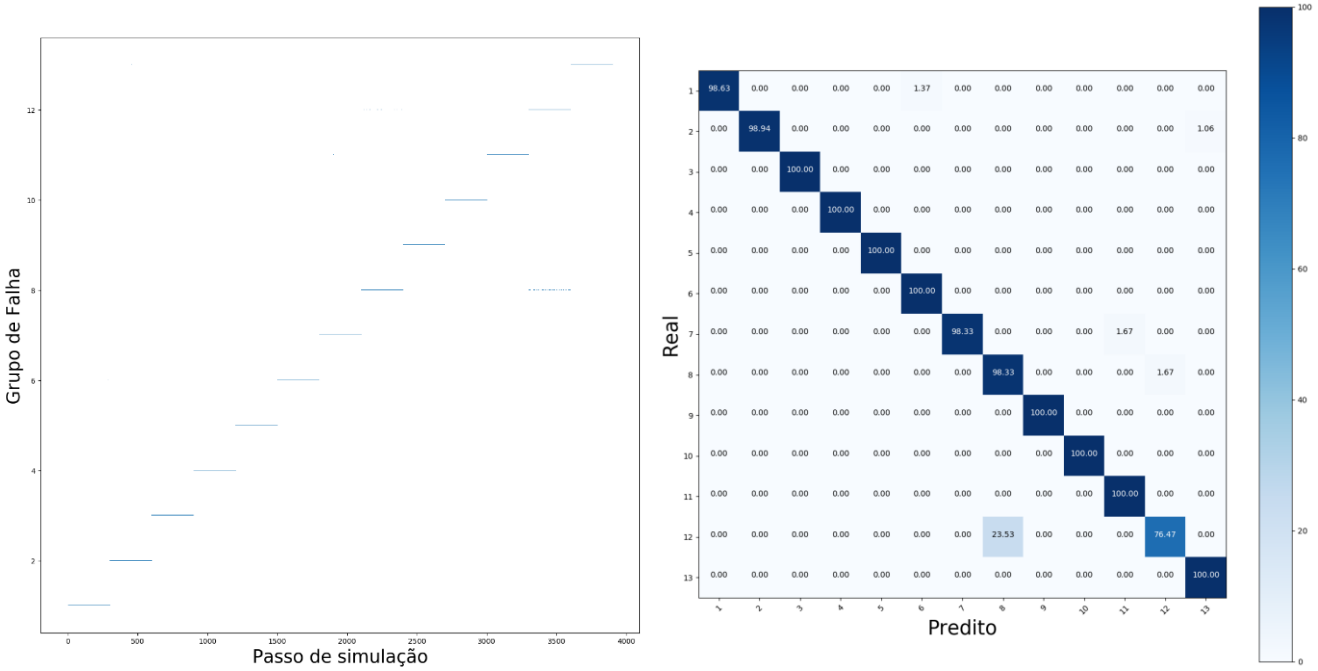
5. Resultados do Gaussian Naive Bayes:
    - Score: 97,99%;
    - Tempo de processamento: 5,72 segundos;



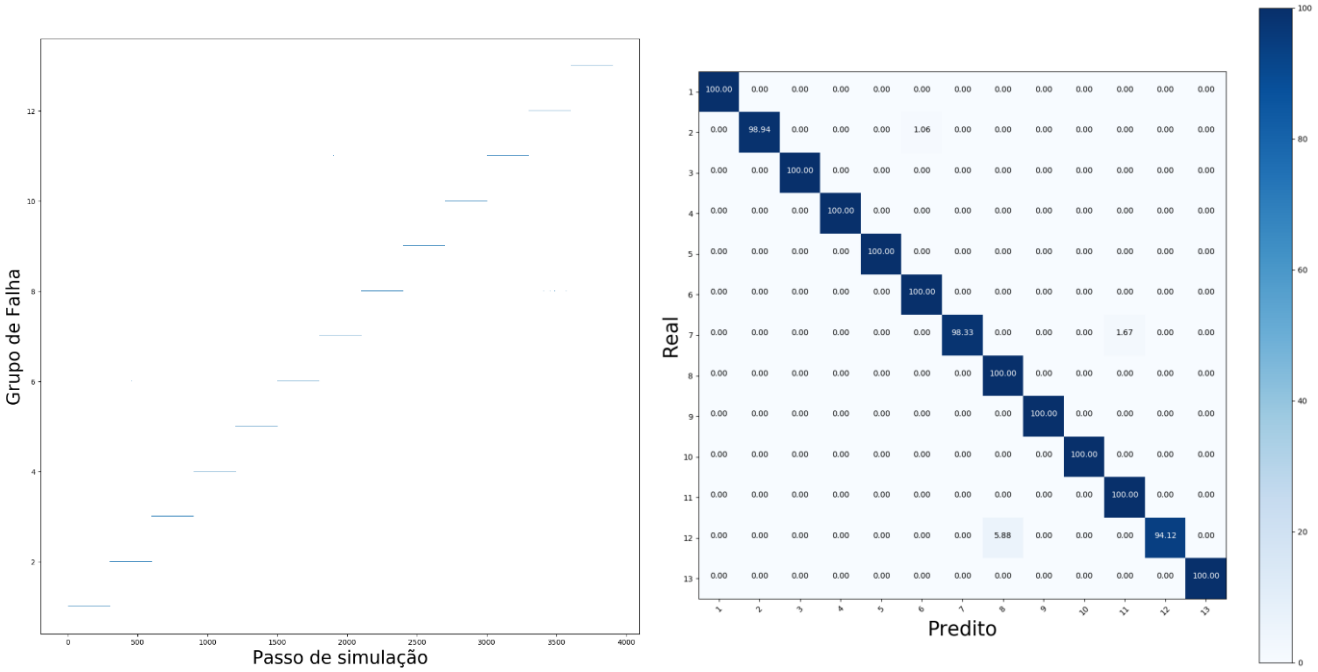
6. Resultados do KNeighbors Classifier:
    - Score: 97,67%;
    - Tempo de processamento: 9,20 segundos;



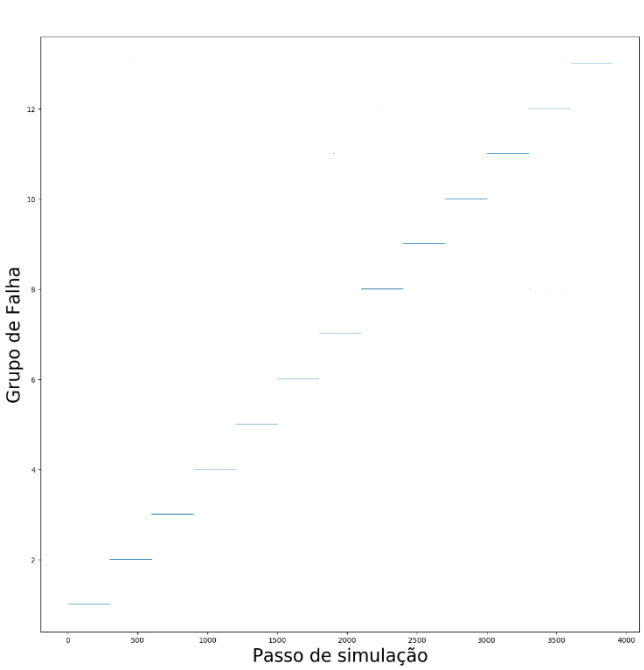
7. Resultados do Stochastic Gradient Descent:
    - Score: 97,76%;
    - Tempo de processamento: 3,09 segundos;

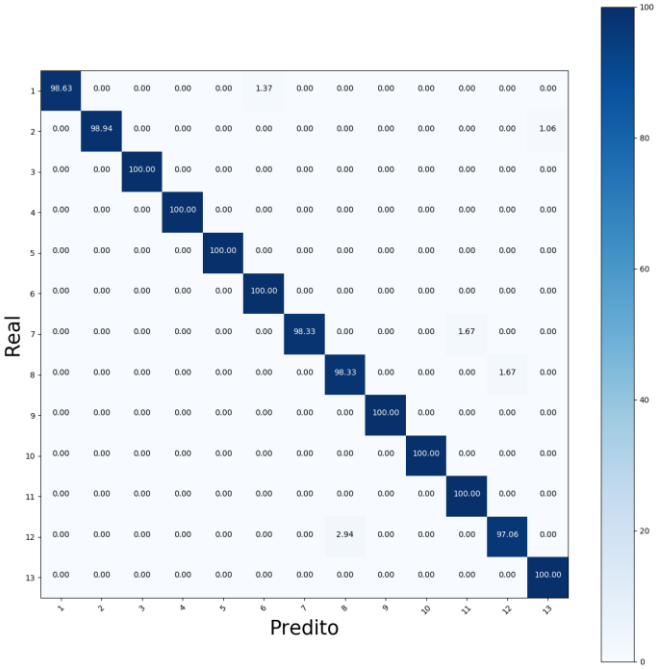


8. Resultados do AdaBoost com estimador base Random Forest Classifier:
    - Score: 99,30%;
    - Tempo de processamento: 4,55 segundos;

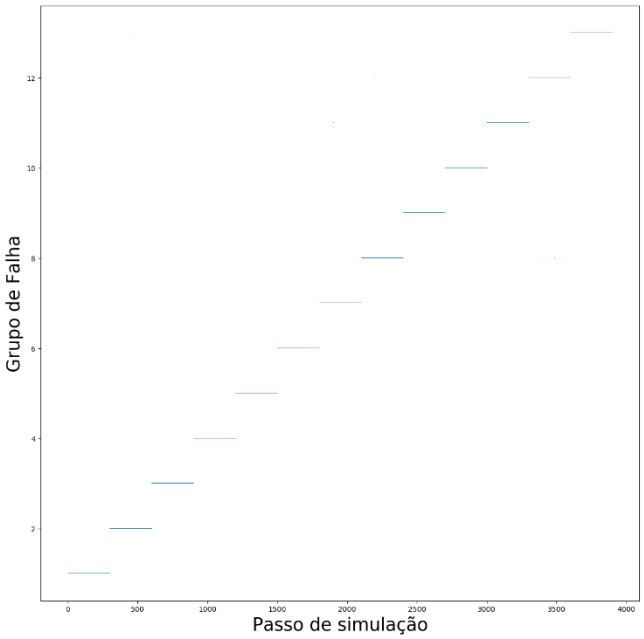


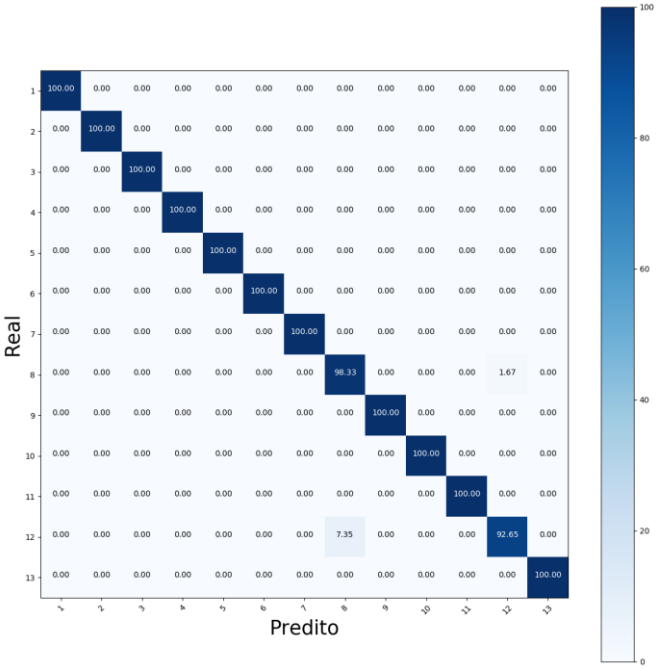
9. Resultados do Logistic Regression:
    - Score: 99,33%;
    - Tempo de processamento: 32,22 segundos;





10. Resultados do Bagging Classifier:
    - Score: 99,07%;
    - Tempo de processamento: 38,00 segundos;





Justificativa

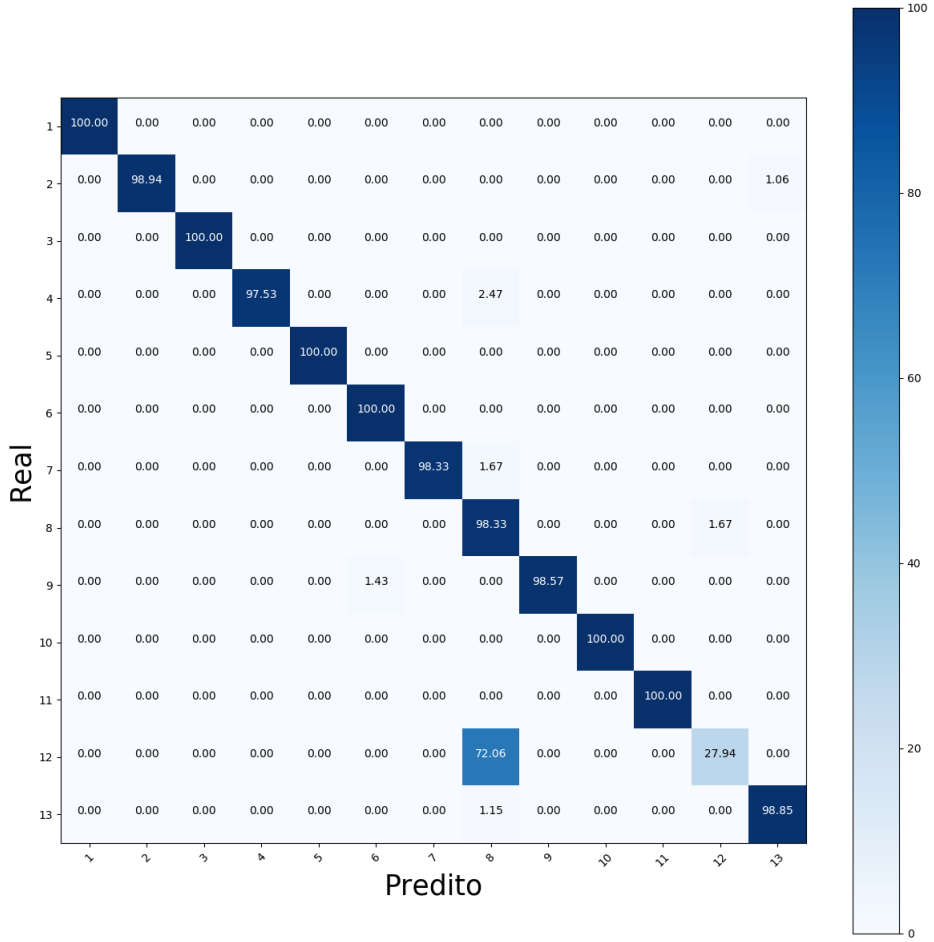
Todos os métodos alcançaram resultados satisfatórios, como esperado, e, também como esperado, alguns levaram um tempo muito longo para realizar as classificações. O AdaBoost, quando empregado sem que nenhum estimador base fosse especificado apresentou um resultado bem pobre, tanto em precisão quanto em tempo de processamento. Isto reflete o fato, aplicável a todos os modelos, de que apesar de todos os métodos apresentarem um bom resultado "out of the box" a otimização de parâmetros se faz necessária para que seja possível alcançar um resultado perfeito.

Como proposto na metodologia, e como AdaBoost sem estimador base apresentou o pior resultado, foi otimizado e apresentou os seguintes resultados:

- Otimização de parâmetros executada em: 1072.0006394386292 segundos
- Melhor Score: 0.894844496823169
- Melhores Parâmetros: {'learning_rate': 0.5, 'random_state': 40}
- Score de teste pré-otimização: 0.6881
- Score de teste pós-otimização: 0.9374

Quando a grade de valores de parâmetros para varredura de otimização foi tão simples quanto:

parameters = {'learning_rate': [0.1, 0.5, 1.],
'random_state': [20, 40, 120, 360]}



Se observarmos o gráfico com os dados após a aplicação do PAA veremos a semelhança entre os dados do oitavo grupo de falha e o décimo segundo, o que causou a confusão do AdaBoost.

Conclusão

Reflexão

O desenvolvimento de estratégias de teste para detectar e diagnosticar falhas em circuitos analógicos e de sinais mistos é uma tarefa complexa. Existem muitos fatores que contribuem para o aumento da dificuldade no teste destes circuitos tais como: a necessidade de alterar conexões para medir correntes ou a falta de bons modelos de falha. Os métodos clássicos necessitam de grande poder computacional se a identificação de parâmetros for utilizada ou caso haja um grande número de simulações, como no caso de um dicionário de falhas.

Esse desafio tem estimulado o desenvolvimento de ferramentas que buscam facilitar os procedimentos de detecção de falhas. Em particular o uso de técnicas de Inteligência Computacional tem sido amplamente empregado, sobretudo através da utilização de classificadores para identificação de componentes defeituosos.

Este projeto apresentou um sistema de detecção de falhas para circuitos lineares utilizando dez diferentes algoritmos de aprendizagem de máquinas, algoritmos estes que foram escolhidos devido a sua recorrência na literatura científica. No processo de classificação foram elaboradas algumas etapas: Simulação dos circuitos, extração e limpeza dos dados, processamento de redução de dimensão para finalmente o conjunto de treino ser submetido ao algoritmo de classificação e finalizando com a predição com o conjunto de teste.

Os quatro diferentes circuitos utilizados são o Sallen key, Filtro Passa Alta (Biquad), Filtro universal (CTSV) e o retificador não linear. O método de limpeza e redução foi o PAA, além de técnicas de ETL com Pandas.

O resultado da predição foi verificado, e então calculado o percentual de precisão de cada um dos algoritmos para cada um dos circuitos utilizando o F-beta score.

Após uma análise geral podemos afirmar quais métodos apresentam uma boa taxa de acerto ou não. O único que precisou de métodos de otimização de parâmetro foi o AdaBoost com a árvore de decisão como estimador base, os demais obtiveram bons desempenhos para a predição chegando perto do 100% em alguns casos.

Dessa forma podemos garantir a eficiência da metodologia para o diagnostico de falhas. Podendo replicar o método para outras fontes de circuitos.

Visualização de Forma Livre e Melhorias

Este projeto atinge seu proposto final com êxito dado os classificadores atingiram altas taxas de precisão. Mesmo se considerarmos os algoritmos que levaram um tempo muito longo para processar os dados o fizeram em um tempo humanamente impraticável. Assim, torna-se possível identificar causadores de falhas e até predizer um comportamento de falha, se optar-se por um sistema de monitoramento em tempo real.

Além disso, fica comprovada a eficácia dos métodos de aprendizado de máquina quando aplicados a dados de circuitos lineares. A quantidade de informações contidas em um circuito linear, por mais simples que seja é muito grande, e quase sempre a melhor abordagem para a solução de circuitos elétrico é a álgebra matricial. O que torna a aplicação de dataframes um metodologia convidativa, pois neste projeto pudemos classificar as falhas com base apenas em no comportamento de uma das grandezas, a tensão de saída.

Devo voltar a ressaltar que os métodos foram implementados sem nenhuma calibração, e algumas pequenas alterações, como a modificação do *kernel* no SVC para linear ou a definição de um estimador base pra o AdaBoost, que por si e só já implicaram em melhoria de resultados. A ideia ao começar a desenvolver o projeto era a de aplicar a otimização de parâmetros através do GridSearchCV a todos os métodos, mas o código como se apresenta leva mais de quarenta minutos para processar por completo.