Projeto Final

Machine Learning Engineer Nanodegree

Gabriel Costa

Outubro, 2018

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 optado pelo uso de 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:

Tabela 1 - Relação entre passo de simulação e passo de variação da falha nos componentes

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

Na Tabela 1 acima, cada amostra de linha é uma simulação e cada amostra de coluna é um nível de variação do erro.

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

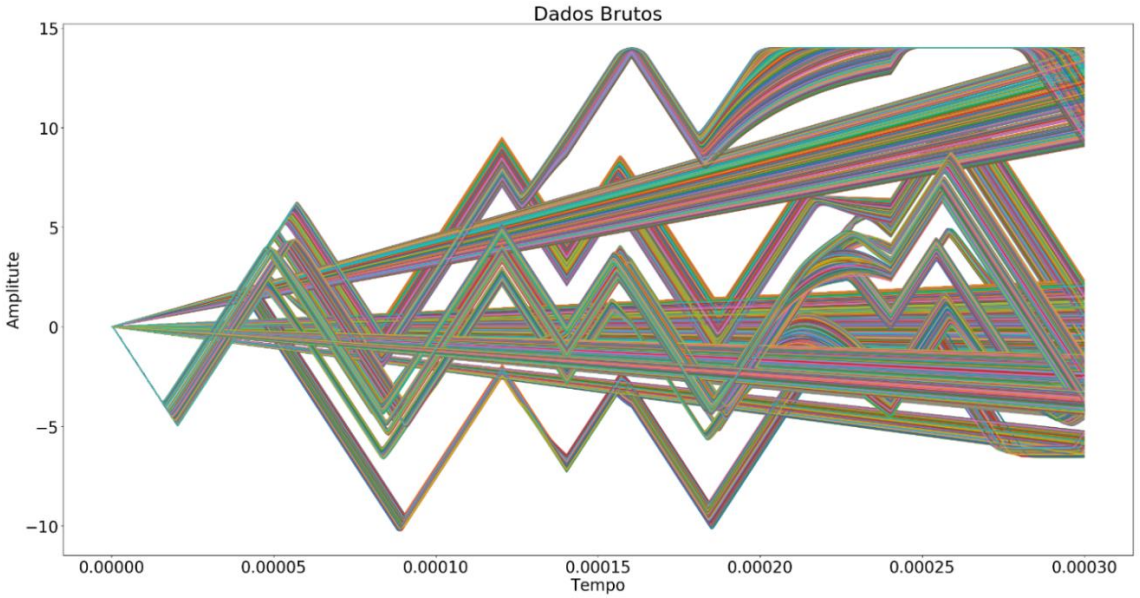


Figura 1 - Dados originais extraídos do circuito Biquad Highpass Filter

Este gráfico traz a representação em linhas contínuas o resultado de cada simulação ao longo dos passos de simulação (tempo). Aqui cada uma das simulações é representada em uma cor diferente o que torna inviável a geração, por código, de legendas que as identifiquem no gráfico. Este, e o próximo gráfico, servem para que possamos observar como algumas das falhas resultam em saídas facilmente discerníveis de outras, e como outras falhas resultam em comportamentos semelhantes.

O dataframe populado com estes dados possui a seguinte característica, onde cada ponto plotado representa a tensão de saída (variável de interesse do circuito) gerada quando a simulação força cada uma das variações dos componentes, forçando-os a gerar uma falha:

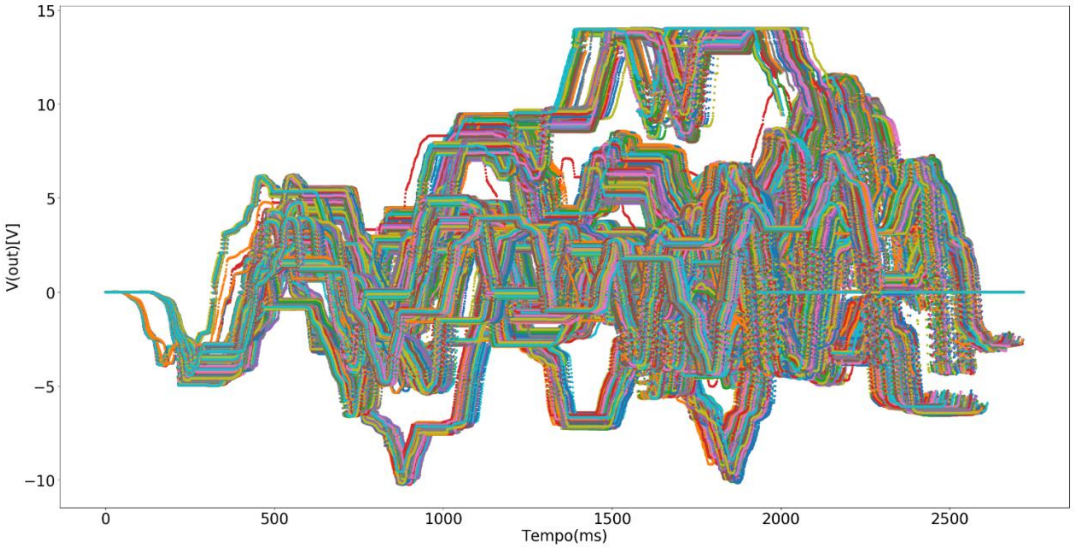


Figura 2 - Valores de Vout de cada passo de simulação plotados individualmente, do circuito Biquad Highpass Filter

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.

Identificação das variáveis

Embora o dataframe com os dados de entrada faça bem em relacionar as simulações com as falhas, caraterística principal da simulação, ele falha em identificar ou relacionar o tipo de falha (componente ou faixa de valor). Assim, se faz necessário uma terceira variável que correlacione o resultado da simulação com o tipo de falha, ou mesmo o passo de variação da falha com o tipo de falha. Esta variável oculta é o *tempo* de simulação que, por não apresentar importância quantitativamente será usada qualitativamente, isto é, o tempo em si (i.e. $0{,}004x10^{-9}$s) não será usado. Ao invés disso usaremos a ordenação de ocorrência (*step*), onde, por exemplo $0{,}004x10^{-9}$s seria o "step 1", $0{,}005x10^{-9}$s o "step 2", $0{,}006x10^{-9}$s o "step 3" e assim sucessivamente, para simulações sucessivas, de forma que para uma segunda simulação um novo tempo t = $0{,}004x10^{-9}$s será identificado como step 1 + n, onde n é o último step da simulação anterior. Desta forma cada valor gerado pela simulação poderá ser tratado exclusivamente.

Esta associação entre step e tipo de falha será tratada posteriormente neste documento como um "gabarito" de identificação de falhas e será fornecido um gráfico característico de seu comportamento. Aqui, uma vez esclarecida esta relação entre simulação, step, falha e tipo de falha, ele será tratado como *INPUT* (dado entrada do algoritmo).

Conforme será apresentado posteriormente na seção "Benchmark", as falhas dos circuitos são geradas de forma premeditada, o que nos permite montar a seguinte tabela relacionando step e tipo de falha:

Tabela 2 – Associação do tipo de falha o passo de simulação

STEP	TIPO DE FALHA	COMPONENTE	FALHA
1	1	R1	BAIXO
2	1	R1	BAIXO
3	1	R1	BAIXO
...	...	...	...
301	2	R1	ALTO
302	2	R1	ALTO
302	2	R1	ALTO
...	...	...	...
2700	10	C2	ALTO
2701	10	C2	ALTO
2702	10	C2	ALTO
...	...	...	...
3298	11	NORMAL	NA
3299	11	NORMAL	NA
3300	11	NORMAL	NA

Na Tabela 2 cada step desta tabela, referente ao circuito “Sallen Key mc + 4bitPRBS [FALHA]”, é referente a uma célula do dataframe de entrada do projeto (que relaciona simulação com o nível de variação da falha). Esta tabela é referente a um único passo de simulação do circuito, e cada um dos 1230 passos de simulação terá uma igual a esta, exceto pela identificação por step (que contará até 4059000, número total de elementos no dataframe).

Assim, o projeto pode tratar como entrada diretamente os valores de simulação contidos células do dataframe lido do arquivo *.raw* (Vout), ou os valores de step, e como saída o tipo de falha em si.

Tabela 3 – Relação de possíveis entradas e saídas alvo para os algoritmos

ENTRADA		SAÍDA		
STEP	**V(out)**	**TIPO DE FALHA**	**COMPONENTE**	**FALHA**
1	8,30E-03	1	R1	BAIXO
2	4,45E-04	1	R1	BAIXO
3	-2,36E-02	1	R1	BAIXO
...	...	...	...	...
2700	-4,38E-03	10	C2	ALTO
2701	3,95E-03	10	C2	ALTO
2702	-3,47E-02	10	C2	ALTO
...	...	...	...	...
3298	7,31E-03	11	NORMAL	NA
3299	1,24E-01	11	NORMAL	NA
3300	-3,03E-02	11	NORMAL	NA

E então é possível categorizar os tipos de variáveis do projeto como:

Tabela 4 – Categorização dos tipos de variáveis do projeto

VARIÁVEL	TIPO	TIPO DE DADO	CATEGORIA
STEP	PREDITOR	NUMÉRICO	CONTÍNUO
V(out)	PREDITOR	NUMÉRICO	CONTÍNUO
TIPO DE FALHA	ALVO	NUMÉRICO	CONTÍNUO
COMPONENTE	ALVO	TEXTO	CATEGÓRICO
FALHA	ALVO	TEXTO	CATEGÓRICO

Aqui ressalto que as variáveis “STEP”, “COMPONENTE”, “FALHA” e “TIPO DE FALHA” não são fornecidos diretamente pela simulação e teriam de ser geradas por código. Então as varáveis escolhidas para serem tratadas pelos algoritmos, para que sejam poupados tempo e recursos computacionais, foram as variáveis “V(out)” e “TIPO DE FALHA”. A variável “TIPO DE FALHA” pode ser extraída facilmente através da fórmula de geração de falhas empregada na simulação, que será discutida nos próximos tópicos. Além disso, este projeto visa ser expandido para aplicações em tempo real, o que torna a escolha do valor real do resultado da simulação (V(out)) a escolha mais sensata.

Análise Comportamental

Embora seja inviável gerar um histograma, e uma análise de dispersão e medida central para cada coluna do dataframe (o circuito *Biquad Highpass Filter*, por exemplo, possui 3300 colunas e mais de 2500 linhas), para efeito de levantamento do comportamento dos circuitos foi usada o valor médio da soma de todos os valores resultantes de um passo de simulação, o que nos permite trabalhar com uma quantidade menor de informação sem perder a essência do dataframe:

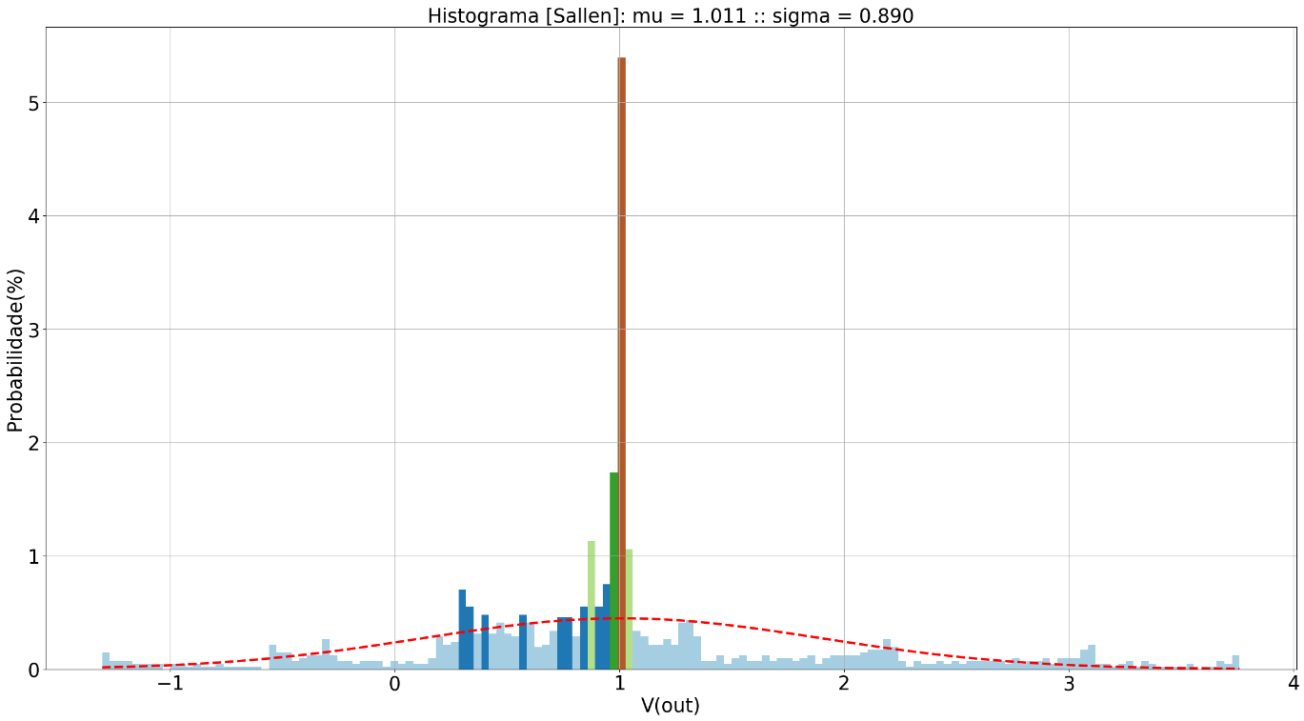


Figura 3 - Histograma dos valores médios do circuito Sallen Key

Tabela 5 - Análises de tendência central e dispersão dos dados

TENDÊNCIA CENTRAL	
Média	1. 0109963395097832
Mediana	0. 9933713759504794
Moda	1.0
Min	-1. 3019416195966982
Max	3. 7606591033032446

DISPERSÃO	
Desvio Padrão	0. 8895876919386501
Variância	0. 7913662616487345
Range	5. 062600722899942
Assimetria	0. 5655721899001163
Curtose	1. 303825233873654

Os histogramas e dados dos demais circuitos encontram-se no repositório.

Pelos dados fornecidos pelo histograma e tabelas podemos observar que os resultados das simulações do circuito *Sallen Key* tendem a se concentrar em torno do valor unitário, e que, apesar de possuir uma gama de valores maior para o lado mais positivo (direito), os valores que se encontram no lado menos positivo (esquerdo) são mais numerosos. Isto faz com que a assimetria tenda a zero. A grande concentração de valores iguais ou bem próximos de “1” faz com que a distribuição tenha uma forma “pontuda”, onde decai rapidamente (68,2% das amostras, ou σ, se encontram na faixa de 0,12V à 1,88V), justificando a curtose (kurtosis) de 1,304.

O “boxplot” a seguir realça as observações anteriores e permite a visualização de possíveis “outliers”:

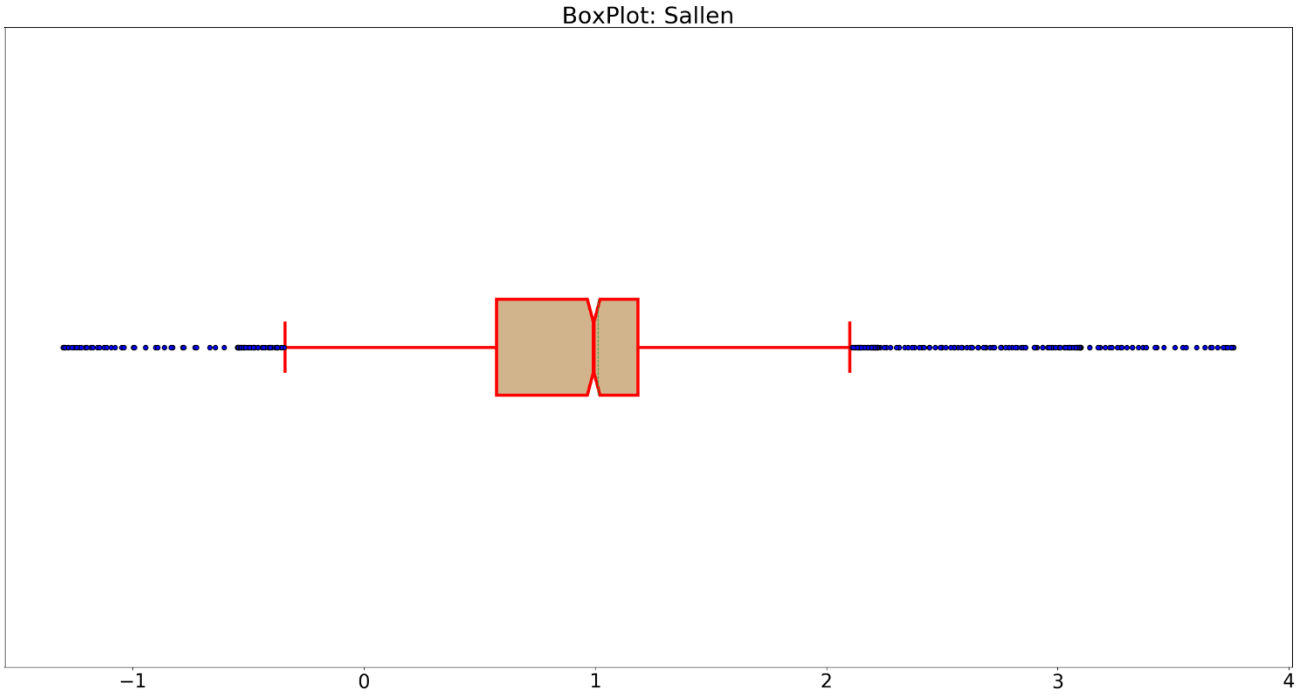


Figura 4 - Boxplot do circuito Sallen Key

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.

Classificadores

Gaussian Naive Bayes

Métodos Naive Bayes se baseiam na aplicação do teorema de Bayes com a suposição *naive* (ingênua) da independência condicional entre pares sucessivos de características, dado o valor da classe.

Referente às suposições aparentemente simplificadas, classificadores Naive Bayes Têm mostrado um bom desempenho em vários problemas do mundo real, famosamente classificação de documentos e filtragem de spam. Eles requerem baixa quantidade de dados para estimar os parâmetros necessários. Para maiores referências teóricas sobre a aplicabilidade do Naive Bayes, veja http://www.cs.unb.ca/~hzhang/publications/FLAIRS04ZhangH.pdf.

Em particular, no caso do Gaussian Naive Bayes é assumido que o comportamento da distribuição dos dados seja gaussiano.

Como neste projeto os dados de entrada apresentam um comportamento próximo ao Gaussiano, este método deve apresentar bons resultados.

K Nearest Neighbors

Classificação baseada em Vizinhança (Neighbors) é um tipo de aprendizado baseado em instâncias, ou aprendizado não-generalizador: não tenta criar um modelo genérico interno, mas simplesmente armazena instâncias dos dados treinados. A classificação é computada através do simples voto majoritário do vizinho mais próximo de cada ponto: um ponto de consulta é atribuído à classe de dados que possui mais representantes dentre os vizinhos mais próximos do ponto.

O classificador K-Neighbor implementa o aprendizado baseando-se nos *k* vizinhos mais próximos de cada ponto de consulta, onde *k* é um valor inteiro especificado pelo usuário. O melhor valor de *k* é extremamente dependente do formato dos dados: em geral, um valor alto de *k* suprime o ruído, mas torna as fronteiras de classificação mais genéricas.

A classificação básica por vizinhança utiliza pesos. Em alguns casos é melhor atribuir maior importância à vizinhos de forma que vizinhos mais próximos entre si contribuam mais para a classificação. Estes pesos podem ser configurados para priorizar de forma uniforma cada um dos pontos da vizinhança, ou para priorizar a proximidade.

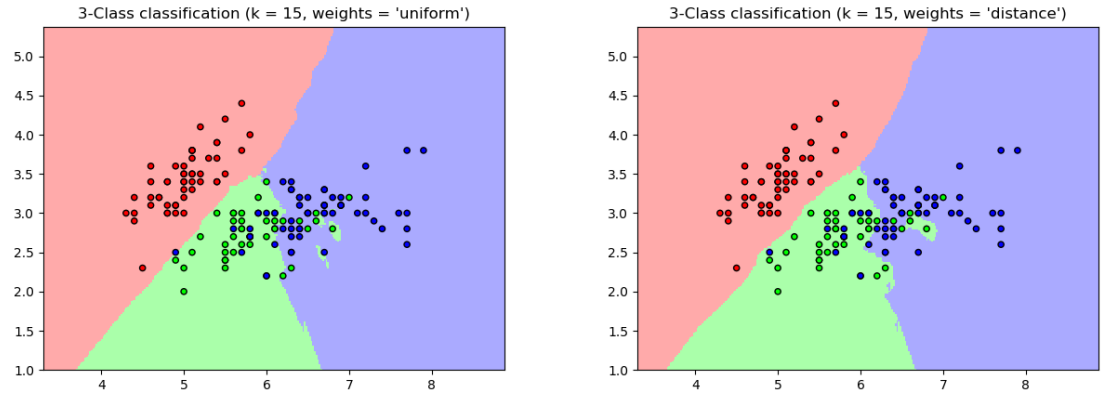


Figura 5 – Comparativo da classificação do KNeighbors para tipos de pesos diferentes

As simulações são compostas de passos de tempo sucessivos, onde um dado par de simulações frequentemente produz resultados suficientemente próximos na saída. Mesmo considerando as variações forçadas nos desvios dos valores dos componentes, que vêm a gerar as falhas, as falhas resultantes também produzem valores subsequentes muito próximos. Por isso, é esperado que mesmo uma aplicação "out of the box" do KNeighbors produza um bom resultado.

Linear Models

São chamados modelos lineares (Linear Models) aqueles que apresentam uma relação entre variáveis que seja linear nos parâmetros. Essa linearidade implica que matematicamente a variação de cada um dos parâmetros é independente dos demais parâmetros do modelo.

Logistic Regression

A regressão logística (logistic regression), apesar do nome, é um método de classificação e não de regressão, e também é conhecida na literatura como Classificação de Máxima entropia (MaxEnt) ou classificador log-linear. Neste modelo a probabilidade descrevendo os possíveis desfechos de um único teste são modeladas usando uma função logística.

A Regressão Logística prevê a probabilidade de uma saída que só pode assumir dois valores (dicotômica). A predição é baseada no uso de diversos preditores (numéricos e categóricos). A Regressão Linear não é apropriada para a predição do valor de uma variável binária por dois motivos:

- Produzirá predições fora do alcance aceitável (além de 0 e 1);
- Dado que experimento dicotômicos só podem assumir um dentre dois valores para o experimento, os resíduos não se distribuirão normalmente sobre a linha de predição.

Por outro lado, uma regressão logística produz uma curva logística, que é limitada a valores entre 0 e 1. A regressão logística é semelhante a uma regressão linear, mas a curva é construída usando o logaritmo natural das "desigualdades" da variável alvo, em vez da probabilidade. Além disso, os preditores não precisam ser distribuídos normalmente ou ter variância igual em cada grupo.

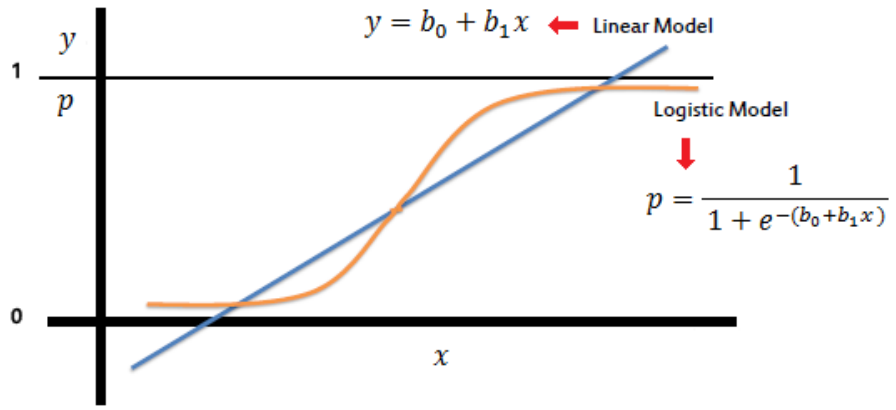


Figura 6 – Comparação entre as curvas da Regressão Linear (azul) e da Regressão Logística (amarela)

Todos os circuitos analisados são lineares, então a relação entre característica e alvo das predições é linear. Além disso, apesar de os alvos serem múltiplos, a comparação para classificação das variáveis só possui dois estados: pertence ao grupo de falha em questão ou não pertence ao grupo de falha em questão. Logo, é esperado que uma aplicação *out of the box* da Regressão Logística atinja bons resultados neste projeto.

Stochastic Gradient Descent (SGD)

SGD é uma abordagem simples, mas muito eficiente, para a aprendizagem discriminativa de classificadores lineares sob funções de perda convexa, como SVM (lineares) e Regressão Logística. Embora o SGD esteja presente há muito tempo na comunidade de aprendizado de máquina, ele recebeu considerável atenção recentemente, no contexto do aprendizado em grande escala por ser particularmente útil quando o número de amostras (e de características) é muito grande.

Tem sido aplicado com sucesso em problemas de aprendizado de máquina em grande escala e esparsos, frequentemente encontrados na classificação de texto e no processamento de linguagem natural. Dado que os dados são escassos, os classificadores neste módulo escalam facilmente para problemas com mais de 10^5 exemplos de treinamento e mais de 10^5 recursos.

As vantagens da descida de gradiente estocástica são:

- Eficiência;
- Facilidade de implementação (fácil ajuste de código);

As desvantagens da Descida de Gradiente Estocástica incluem:

- Requer um número de hiperparâmetros, como o parâmetro de regularização e o número de iterações;
- É sensível ao dimensionamento de recursos;

A prerrogativa da aplicação deste método neste projeto é a mesma da Regressão Logísitica: a linearidade dos dados. Aqui é esperado que o desempenho do SGD *out of the box* não seja tão satisfatório, dada a semelhança dos dados e dada a sua necessidade por configuração dos hiperparâmetros.

Ensemble

Um classificador *ensemble* (também chamado de comitê de *learners*, mistura de especialistas ou sistema de classificadores múltiplo), consiste em um conjunto de classificadores treinados individualmente, classificadores de base, cujas decisões são de alguma forma combinadas (Marques et al., 2012).

Classificadores Ensemble apresentam melhor desempenho em conjuntos de dados maiores e mais complexos, embora exijam maiores quantidades de recursos computacionais, e por serem compostos de grupos de classificadores costumam apresentar menores taxas de *overfitting*.

Bagging

O *Bagging (Bootstrap Aggregating)*, um método proposto por Breiman em 1996, gera um conjunto de dados por amostragem *bootstrap* dos dados originais. *Bootstrapping* consiste em "reamostrar" um conjunto de dados, diretamente ou via um modelo ajustado, a fim de criar réplicas dos dados, a partir das quais podemos avaliar a variabilidade de quantidades de interesse, sem usar cálculos analíticos. O conjunto de dados gera um conjunto de modelos utilizando um algoritmo de aprendizagem simples por meio da combinação por votos para classificação. O seu uso é particularmente atraente quando a informação disponível é de tamanho limitado.

No *Bagging* os classificadores são treinados de forma independente por diferentes conjuntos de treinamento através do método de inicialização. Para construí-los é necessário montar *k* conjuntos de treinamento idênticos e replicar esses dados de treinamento de forma aleatória para construir *k* redes independentes por re-amostragem com reposição. Em seguida, deve-se agregar as *k* redes através de um método de combinação apropriada, tal como a maioria de votos (Breiman, 1996).

Para garantir que há amostras de treinamento suficientes em cada subconjunto, grandes porções de amostras (75-100%) são colocadas em cada subconjunto. Com isso, os subconjuntos individuais de formação se sobrepõem de forma significativa, com muitos casos fazendo parte da maioria dos subconjuntos e podendo até mesmo aparecer várias vezes num mesmo subconjunto. A fim de assegurar a diversidade de situações, um *learner* de base relativamente instável é usado para que limites de decisão diferentes possam ser obtidos, considerando-se pequenas perturbações em diferentes amostras de treinamento (Wang, 2011).

AdaBoost

No *Boosting*, de forma semelhante ao *Bagging*, cada classificador é treinado usando um conjunto de treinamento diferente. A abordagem por *Boosting* original foi proposta por Schapire em 1990. A principal diferença em relação ao *Bagging* é que os conjuntos de dados re-amostrados são construídos especificamente para gerar aprendizados complementares e a importância do voto é ponderado com base no desempenho de cada modelo, em vez da atribuição de mesmo peso para todos os votos. Essencialmente, esse procedimento permite aumentar o desempenho de um limiar arbitrário simplesmente adicionando *learners* mais fracos. Dada a utilidade desse achado, *Boosting* é considerado uma das descobertas mais significativas em aprendizado de máquina (LANTZ, 2013).

O AdaBoost, "*Adaptive Boosting*", é uma combinação das ideias de *Bagging* e *Boosting* e não exige um grande conjunto de treinamento como o *Boosting*. Inicialmente, cada exemplo de formação de um determinado conjunto de treinamento tem o mesmo peso.

Para treinar o k−ésimo classificador como um "modelo de aprendizagem fraca", *n* conjuntos de amostras de treinamento entre S são usadas para treinar o k−ésimo classificador. Em seguida, o classificador treinado é avaliado por S para identificar os exemplos de treinamento que não foram classificados corretamente (TSAI, 2014). A rede k+1 é então treinada por um conjunto treinado modificado que reforça a importância desses exemplos classificados incorretamente.

Este procedimento de amostragem será repetido até que *k* amostras de treinamento sejam construídas para a construção da k−ésima rede. Portanto, a decisão final baseia-se na votação ponderada dos classificadores individuais.

Random Forest

Random Forest (Floresta Aleatória) é um algoritmo de aprendizagem de máquina flexível e fácil de usar que produz excelentes resultados na maioria das vezes, mesmo sem ajuste de hiperparâmetros. É também um dos algoritmos mais utilizados, devido à sua simplicidade e o fato de que pode ser utilizado para tarefas de classificação e também de regressão. Colocando de forma simples, o algoritmo de florestas aleatórias cria várias árvores de decisão e as combina para obter uma predição com maior acurácia e mais estável.

Com raras exceções um classificador de floresta aleatória tem todos os hiperparâmetros de uma árvore de decisão e também todos os hiperparâmetros de um classificador *bagging*, para controlar a combinação de árvores. Ao invés de construir um classificador *bagging* e passá-lo para um classificador de árvore de decisão, você pode usar a classe da floresta aleatória, que é mais conveniente e otimizada para árvore de decisões. Este algoritmo adiciona aleatoriedade extra ao modelo, quando está criando as árvores. Ao invés de procurar pela melhor característica ao fazer a partição de nós, ele busca a melhor característica em um subconjunto aleatório das características. Este processo cria uma grande diversidade, o que geralmente leva a geração de modelos melhores.

Portanto, quando você está criando uma árvore na floresta aleatória, apenas um subconjunto aleatório das características é considerado na partição de um nó. E é possível até fazer as árvores ficarem mais aleatórias utilizando limiares (*thresholds*) aleatórios para cada característica, ao invés de procurar pelo melhor limiar (como uma árvore de decisão geralmente faz).

Decision Tree Classifier

Decision Trees (árvores de decisão) são métodos não-paramétricos não-supervisionados de aprendizagem usados para classificação e regressão. O objetivo é criar um modelo capaz de predizer o valor de uma variável alvo através do aprendizado de regras simples de decisão inferidas das características dos dados.

No exemplo abaixo Decision Trees aprendem a partir dos dados a aproximar uma senóide através de um conjunto de regras de decisão "if-then-else". Quanto mais profunda a árvore, mais complexas as regras de decisão e mais preciso o modelo.

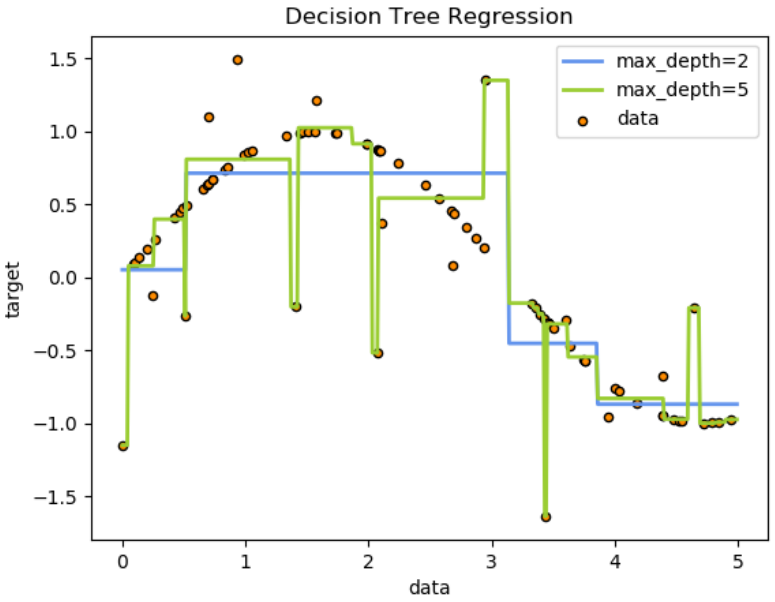


Figura 7 – Aproximação do comportamento de uma senóide por profundidade de Árvores de Decisão

Algumas de suas vantagens são:

- Simplicidade de entender e implementar. Árvores podem ser visualizadas.
- Demanda pouca preparação dos dados. Outas técnicas frequentemente requerem a normalização dos dados, criação de variáveis auxiliares ou remoção de campos vazios. Este método, porém, não suporta valores faltantes;
- O custo computacional da predição da Árvores de Decisão é logaritmicamente proporcional á quantidade de dados usados para treiná-la;
- Pode processar tanto dados numéricos como categóricos;
- É capaz de lidar com problemas multivariáveis;
- Usa o modelo “white box”, onde a observabilidade de uma dada situação em um dado modelo pode ser explicada por lógica booleana. Em contrapartida, em um modelo “black box” (redes neurais artificiais, por exemplo) os resultados podem se tornar complicados de interpretar.
- Confiável por ser possível de validar e um modelo através de testes estatísticos;
- Alcança bom desempenho mesmo se suas suposições forem de alguma forma violadas pelo modelo real, de onde os dados foram gerados;

Algumas desvantagens do modelo:

- Pode se tornar muito complexo e causar overfitting;
- Pequenas variações nos dados podem criar árvores completamente diferentes;
- Classes dominantes induzem a criação de learners polarizados;

Support Vector Machine – SVM

O SVM e uma técnica de aprendizado de máquina que se baseia em aprendizado estatístico que foi proposta por Vapnik em 1995 (Cortes and Vapnik, 1995). Desde então vem recebendo constante atenção, principalmente pelo fato de que a aplicação dessa técnica tem provido resultados melhores que a aplicação de redes neurais. Consiste resumidamente em encontrar um hiperplano ótimo capaz de separar conjuntos, isto é, que possua a maior margem de diferenciação entre eles possível. SVM originalmente foi desenvolvido como um método de separação linear (Cortes and Vapnik, 1995), mas é possível estendê-lo de forma a separar classes que são não lineares, a ideia principal é transportar os dados para um espaço de dimensão maior no qual eles possam ser separados linearmente.

Quando tentamos reconhecer padrões estamos tentando estimar funções, usando dados de treinamento, que associem padrões multidimensionais (características) a uma classe, de forma que novos padrões (fora do conjunto de treinamento) possam ser corretamente qualificadas através da semelhança de suas probabilidades com as do conjunto de treinamento. Se não colocarmos nenhuma restrição sobre a classe das funções a partir da qual escolhemos nossa função de estimativa, mesmo que ela resolva muito bem para os dados de treinamento, pode ser que ela não generalize bem para exemplos desconhecidos. SVMs são uma implementação aproximada do método de minimização do risco estrutural, que vem da teoria do aprendizado estatístico (Vapnik, 1995). Este princípio de indução é baseado no fato de que a taxa de erro do aprendizado de máquina nos dados de teste (isto é, a taxa de erro da generalização) é limitada pela soma da taxa de erro no treinamento com um termo que depende da dimensão Vapnik-Chervonenkis (VC) (Haykin, 1998).

De forma simples o SVM realiza a separação de um conjunto de objetos com diferentes classes, ou seja, utiliza o conceito de planos de decisão. Podemos analisar o uso do SVM com o seguinte exemplo. Na Figura a seguir é possível observar duas classes de objetos: azul ou laranja. Essa linha que os separa define o limite em que se encontram os pontos azuis e os pontos laranjas. Ao entrarem novos objetos na análise, estes serão classificados como laranjas se estiverem à direita e como azuis caso situem-se à esquerda. Neste caso, conseguimos separar, por meio de uma linha, o conjunto de objetos em seu respectivo grupo, o que caracteriza um classificador linear.

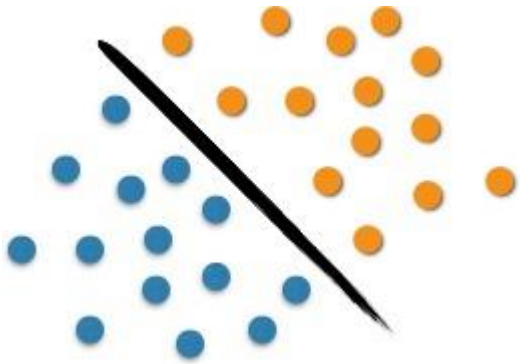


Figura 8 – Exemplo de dados de separação simples

No entanto, problemas de classificação costumam ser mais elaborados, sendo necessário realizar a separação ótima por meio de estruturas mais complexas. O SVM propõe a classificação de novos objetos (teste) com base em dados disponíveis (treinamento). Como pode-se observar na Figura 2. A separação ótima nesse caso ocorreria com a utilização de uma curva.

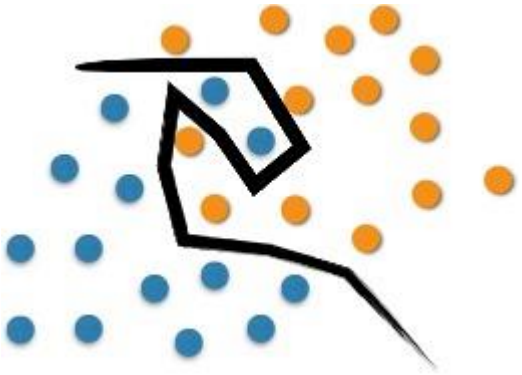


Figura 8 – Exemplo de dados de separação complexa

Na esquerda da Figura 3 observamos a complexidade em separar os objetos originais. Para mapeá-los, utilizamos um conjunto de funções matemáticas, conhecido como *Kernels*. Esse mapeamento é conhecido como o processo de reorganização dos objetos.

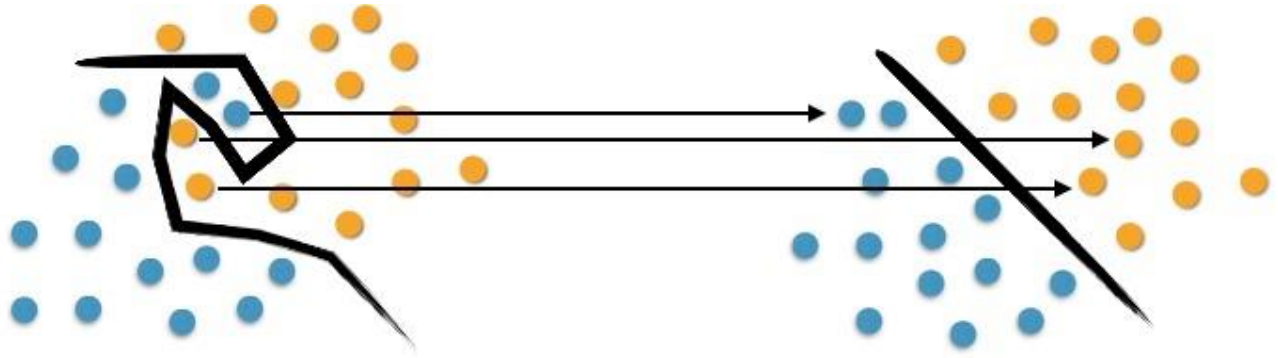


Figura 8 – Simplificação provida pelo SVM

Observa-se que à direita da Figura 3 há uma separação linear dos objetos. Assim, ao invés de construir uma curva complexa, como no esquema à esquerda; o SVM propõe, nesse caso, essa linha ótima capaz de separar os pontos azuis dos alaranjados.

Algumas vantagens do modelo:

- Eficaz em espaços dimensionais elevado;
- Também é eficiente em casos onde o número de dimensões é maior que o número de amostras;
- Usa um subconjunto de pontos de treino na função de decisão, então também é eficiente quanto ao consumo de memória;
- Versátil: diferentes funções de *kernel* podem ser especificadas para a função de decisão.

Algumas Desvantagens do modelo:

- Se a quantidade de características for muito maior que op número de amostras é mandatória a escolha da melhor função *kernel* e do melhor termo de regularização;
- SVMs não proveem estimativas de probabilidade diretamente, estas são fornecidas por métodos de *cross-validation* pesados;

Dada a complexidade de separação dos dados criada pela semelhança entre dados, é esperado que o SVM produza bons resultados neste projeto.

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].

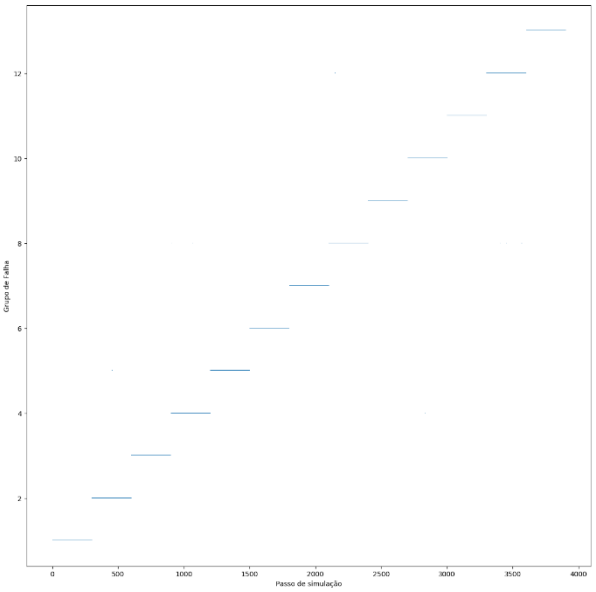


Figura 9 – Gabarito gráfico do circuito Biquad Highpass Filter mc + 4bitPRBS [FALHA]

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:

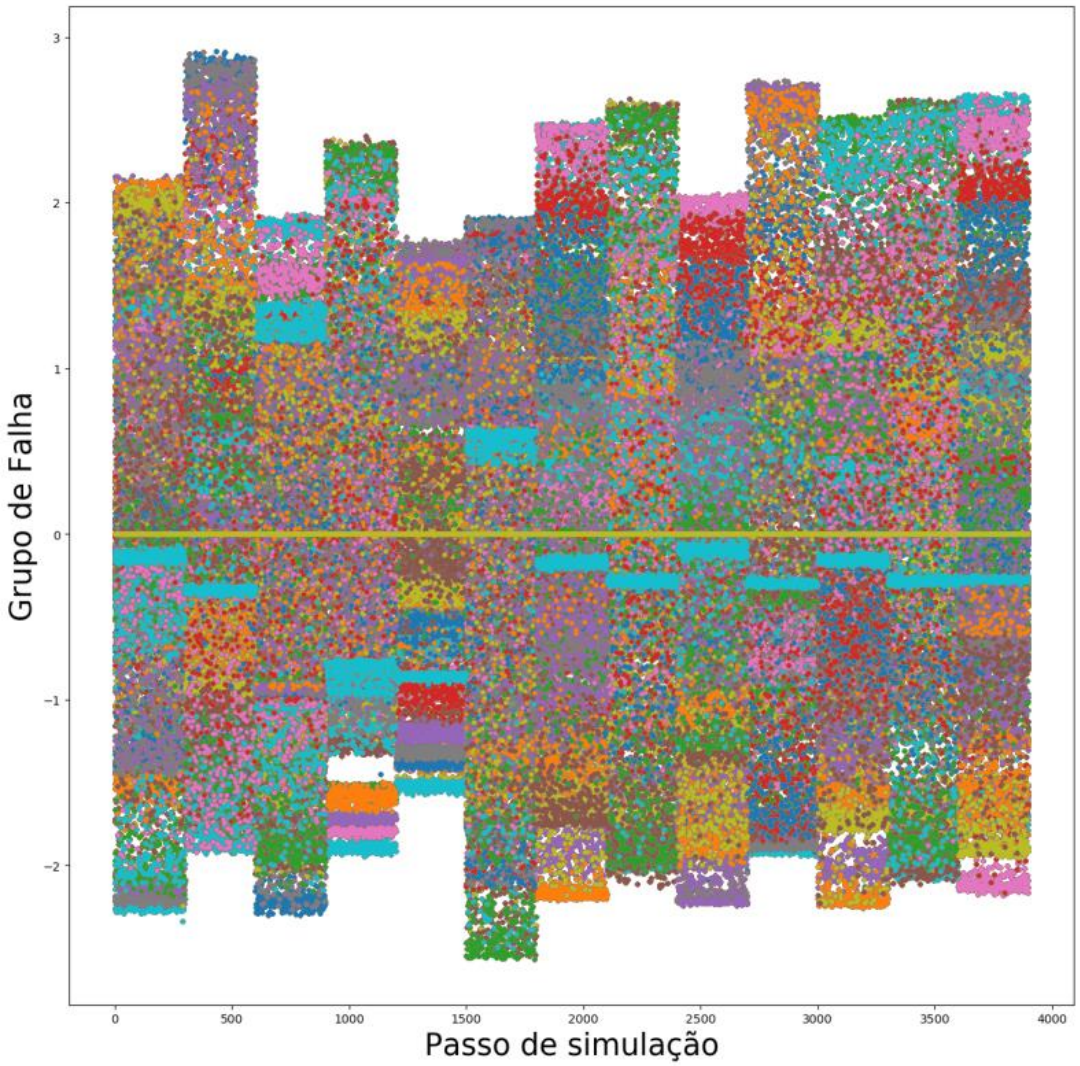


Figura 10 – Valores de Vout de cada passo de simulação após aplicação do PAA, do circuito Biquad Highpass Filter

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];

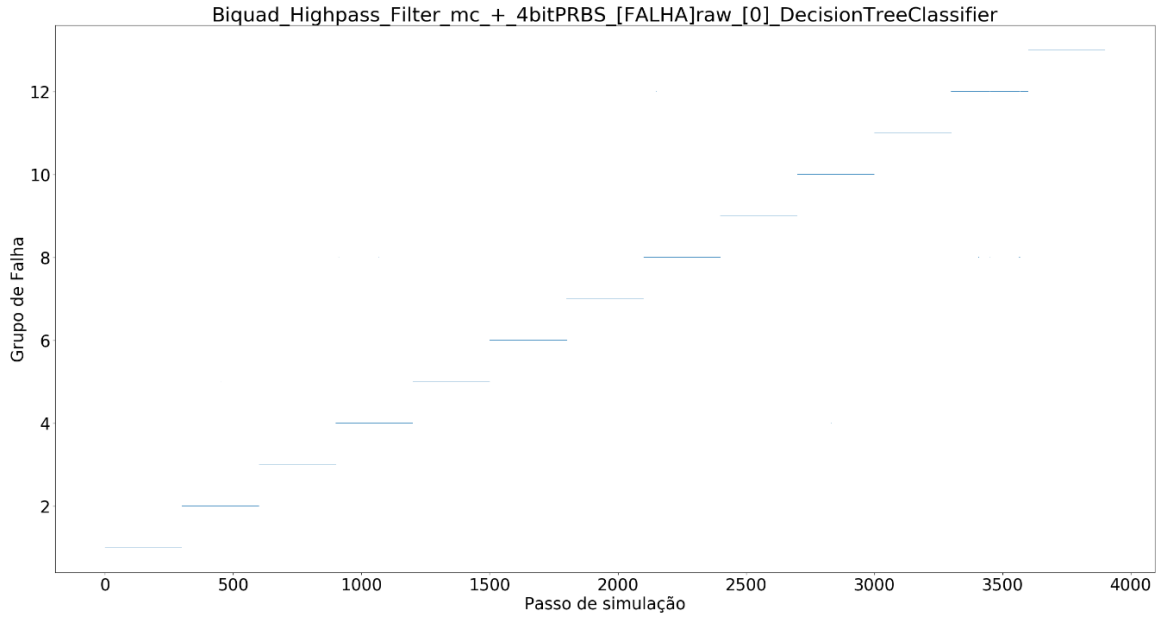
A instância do GridSearchCV implementa a API comum do estimador: quando “encaixando” em um conjunto de dados todas as combinações possíveis dos valores dos parâmetros são avaliadas e a melhor cominação e retida.

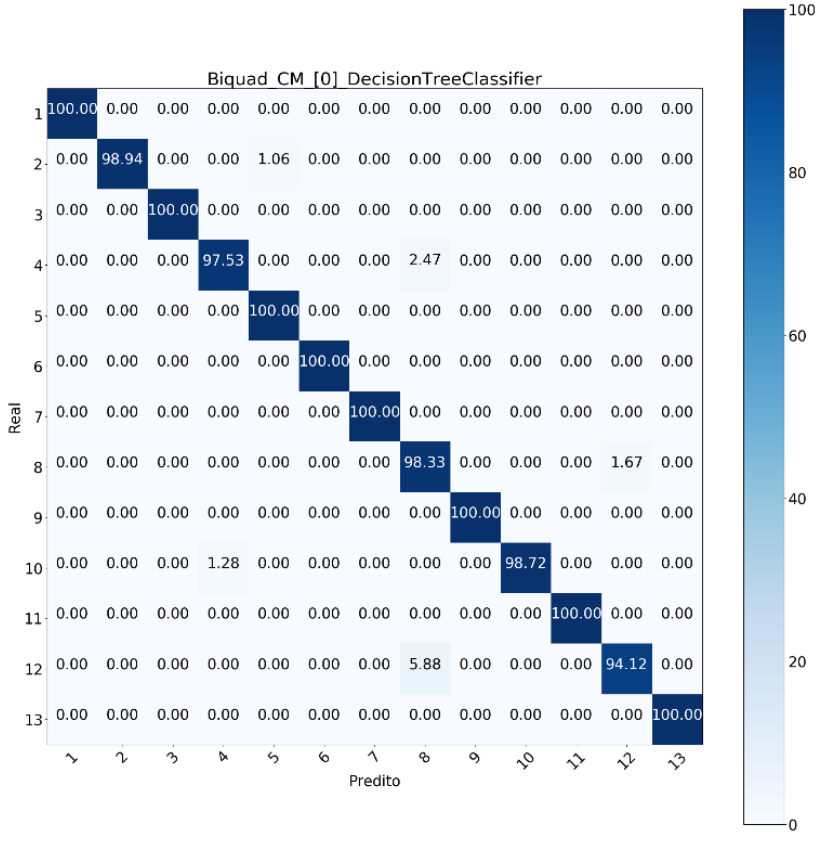
Resultados

Validação e Avaliação de Modelo

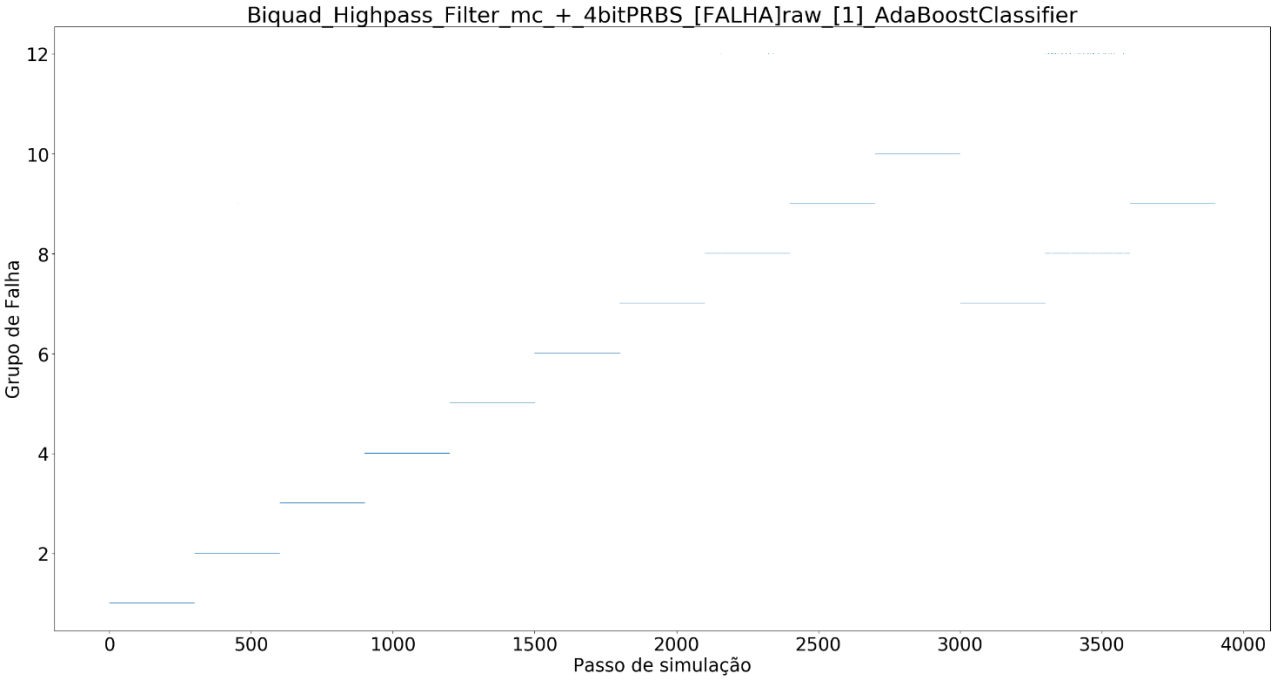
O código gera mais de 150 figuras como saída, então para fins práticos somente os gráficos referentes ao circuito Biquad Highpass Filter serão apresentados. Os resultados para os outros circuitos podem ser encontrados em:
https://drive.google.com/drive/folders/1aj-4SWuYvI32WoE9ZGpv3DejWCLAckw8?usp=sharing

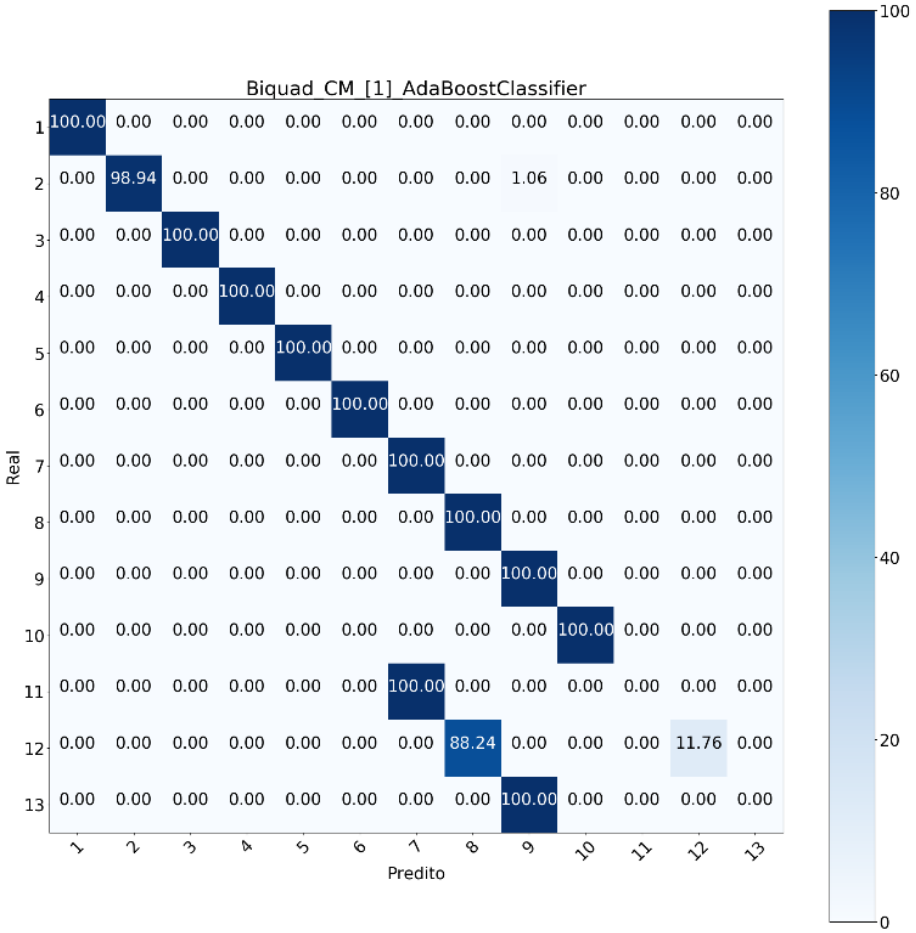
1. Resultados do Decision Tree Classifier:
    - Score: 98,98%;
    - Tempo de processamento: 5,99 segundos;



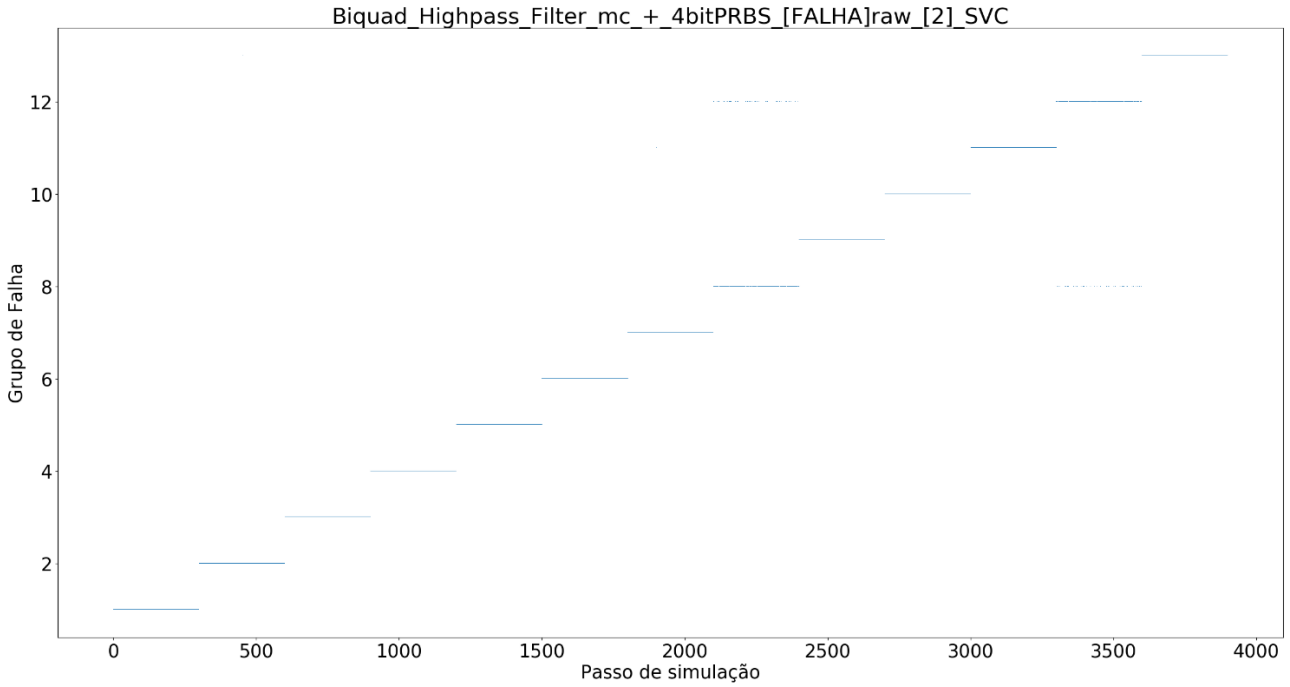


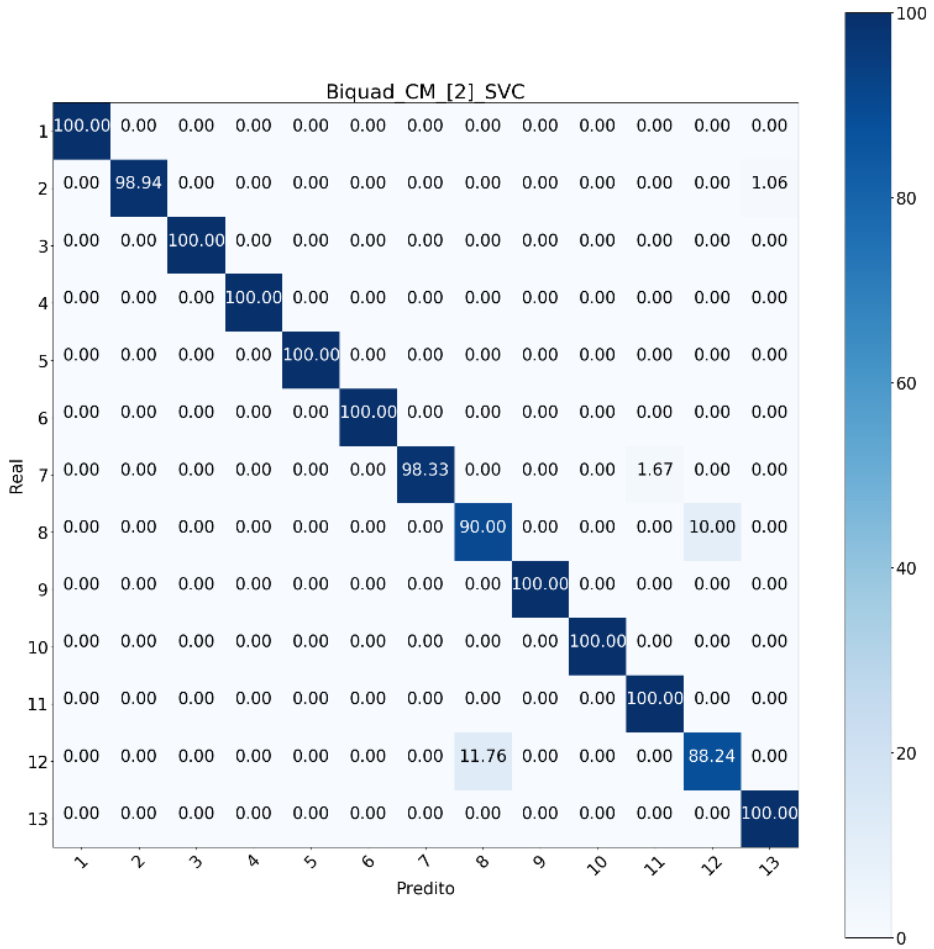
2. Resultados do AdaBoost Classifier;
    - Score: 68,81%;
    - Tempo de processamento: 54,72 segundos;



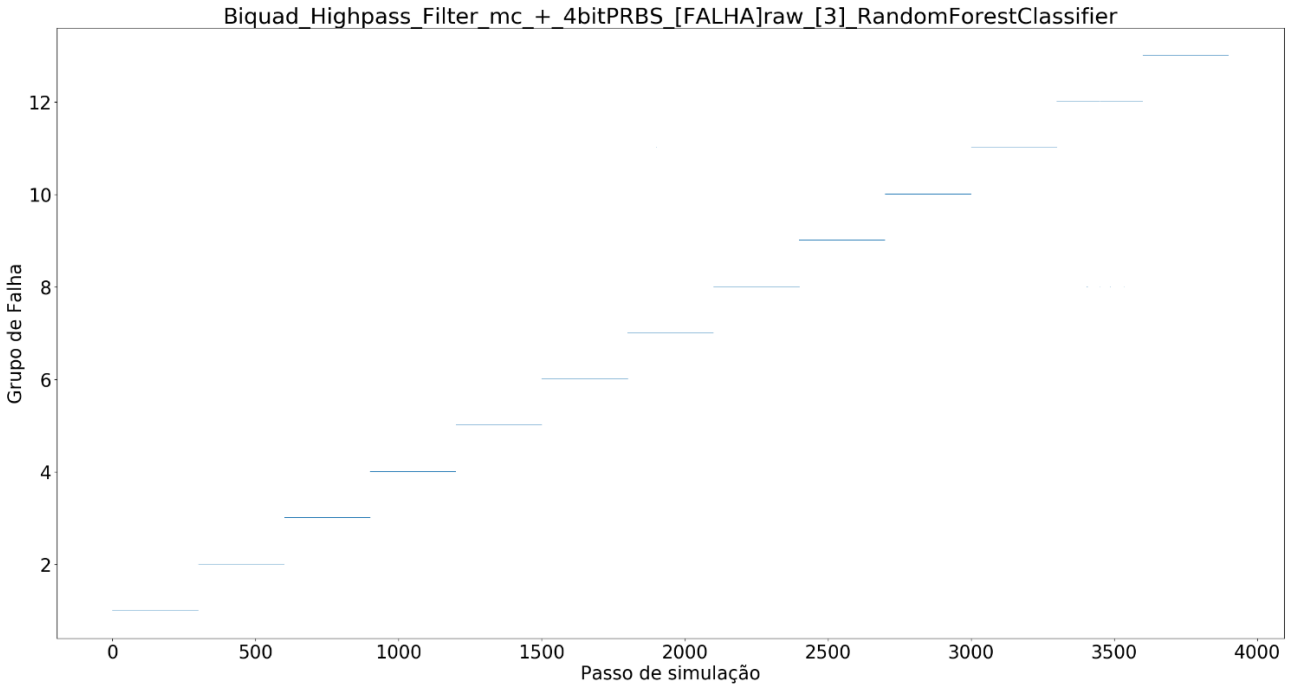


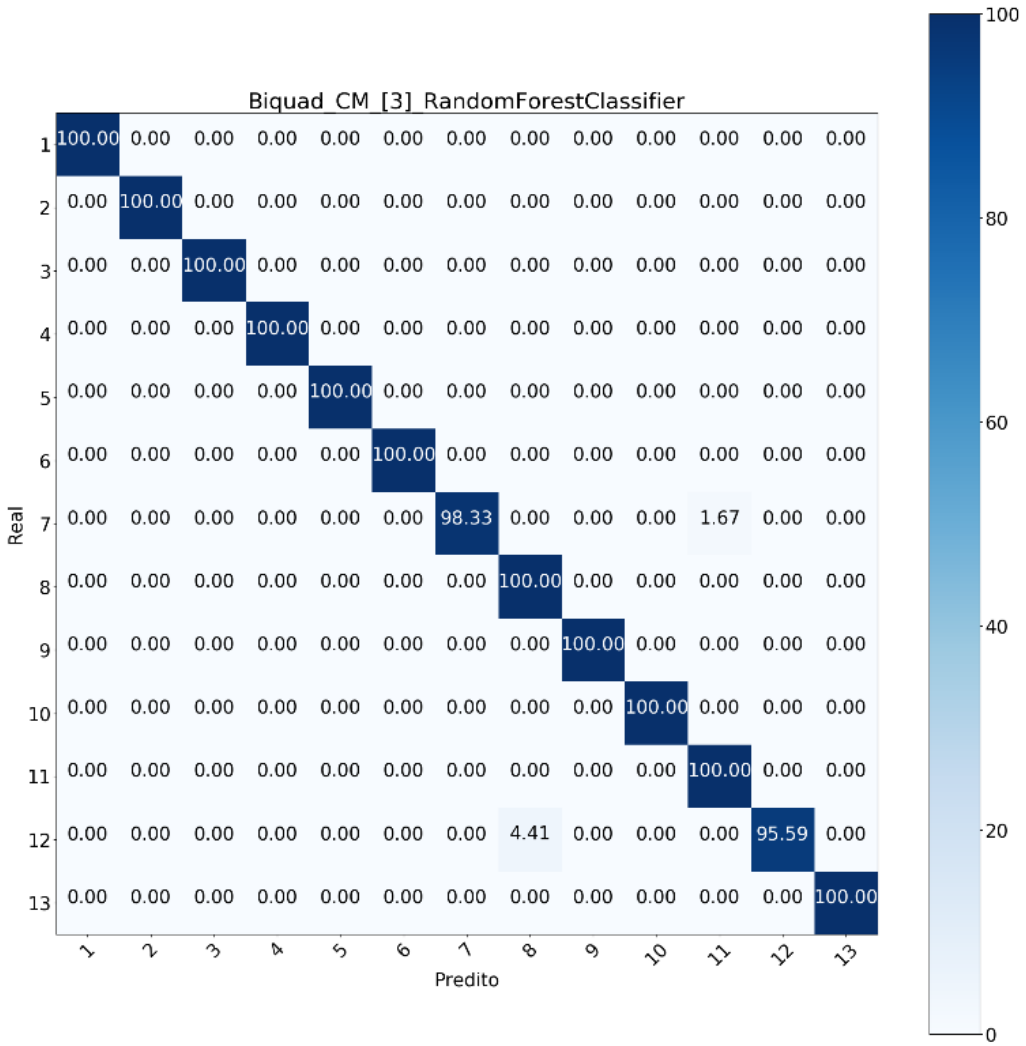
3. Resultados do SVC:
    - Score: 98,12%;
    - Tempo de processamento: 10,82 segundos;



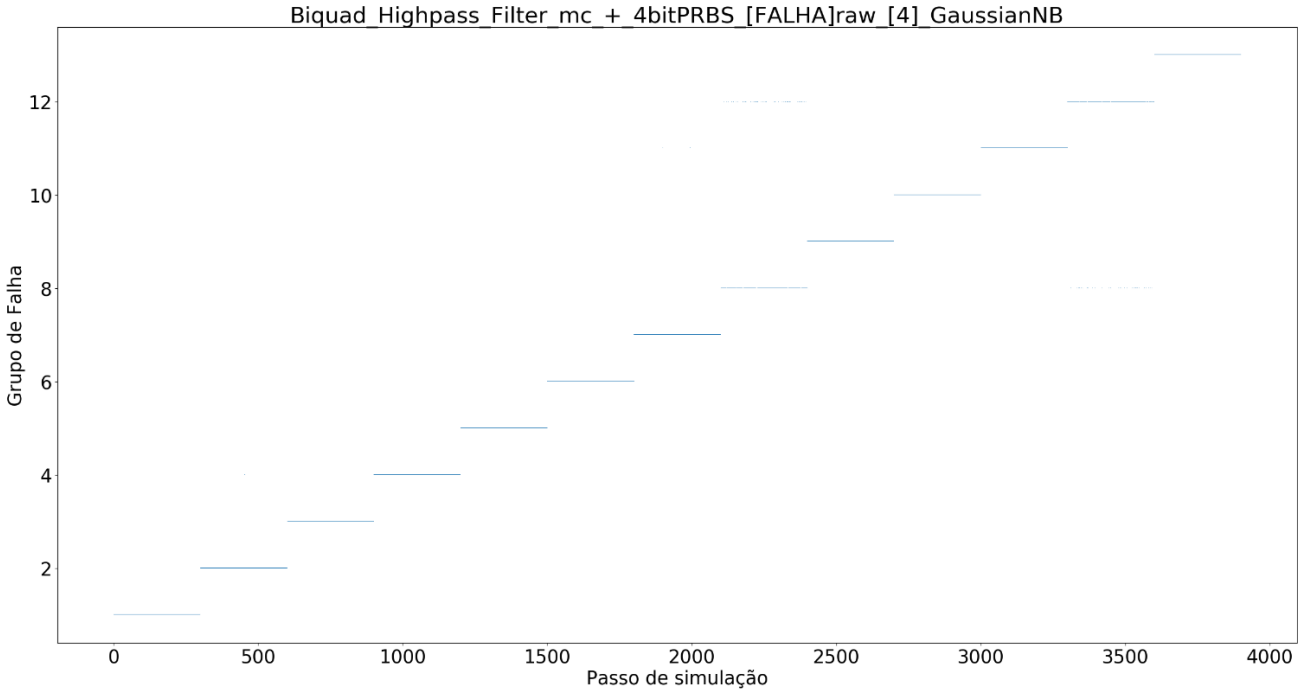


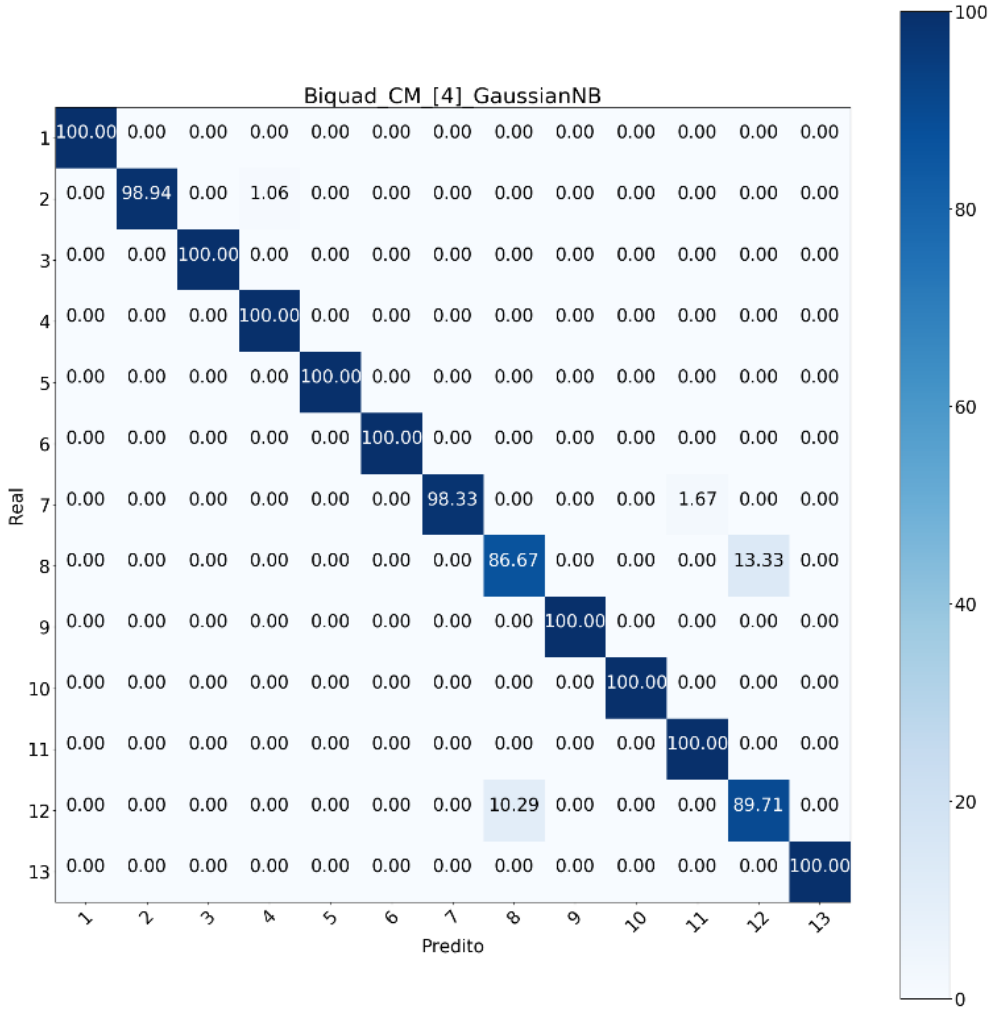
4. Resultados do Random Forest Classifier:
    - Score: 99,53%;
    - Tempo de processamento: 4,32 segundos;



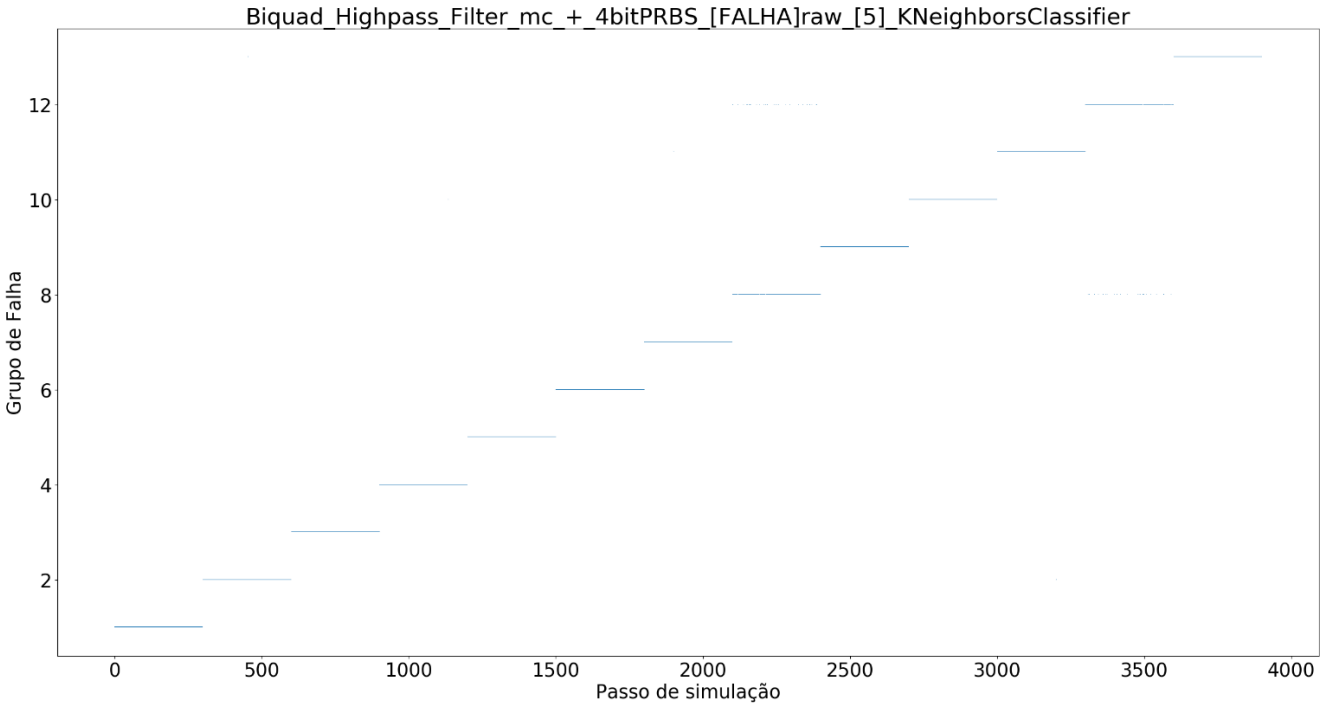


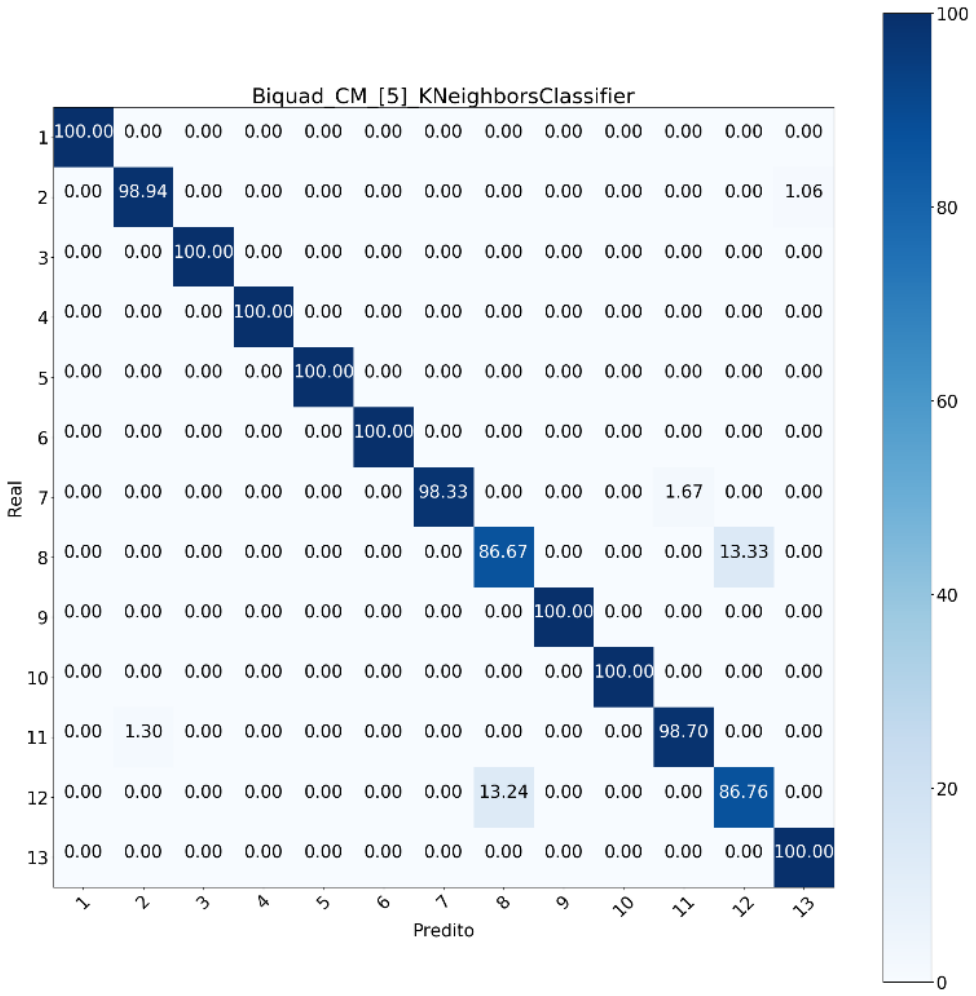
5. Resultados do Gaussian Naive Bayes:
    - Score: 97,99%;
    - Tempo de processamento: 5,72 segundos;



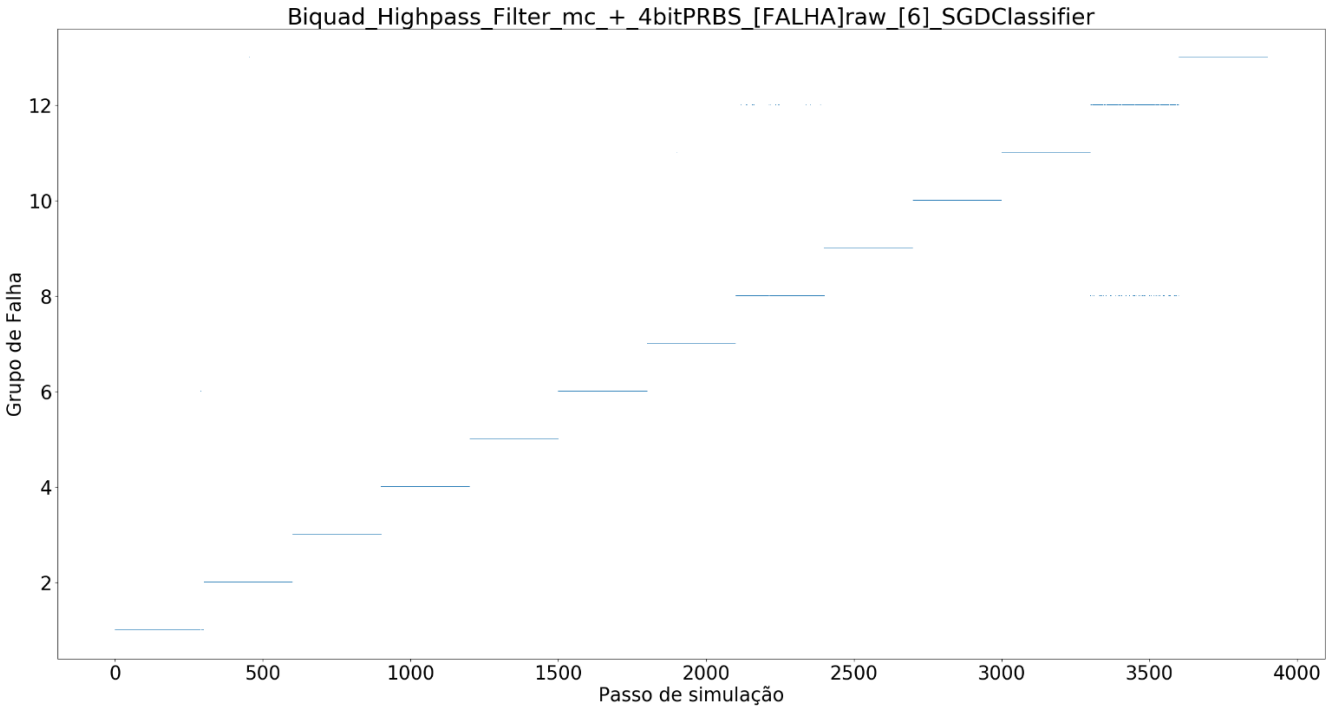


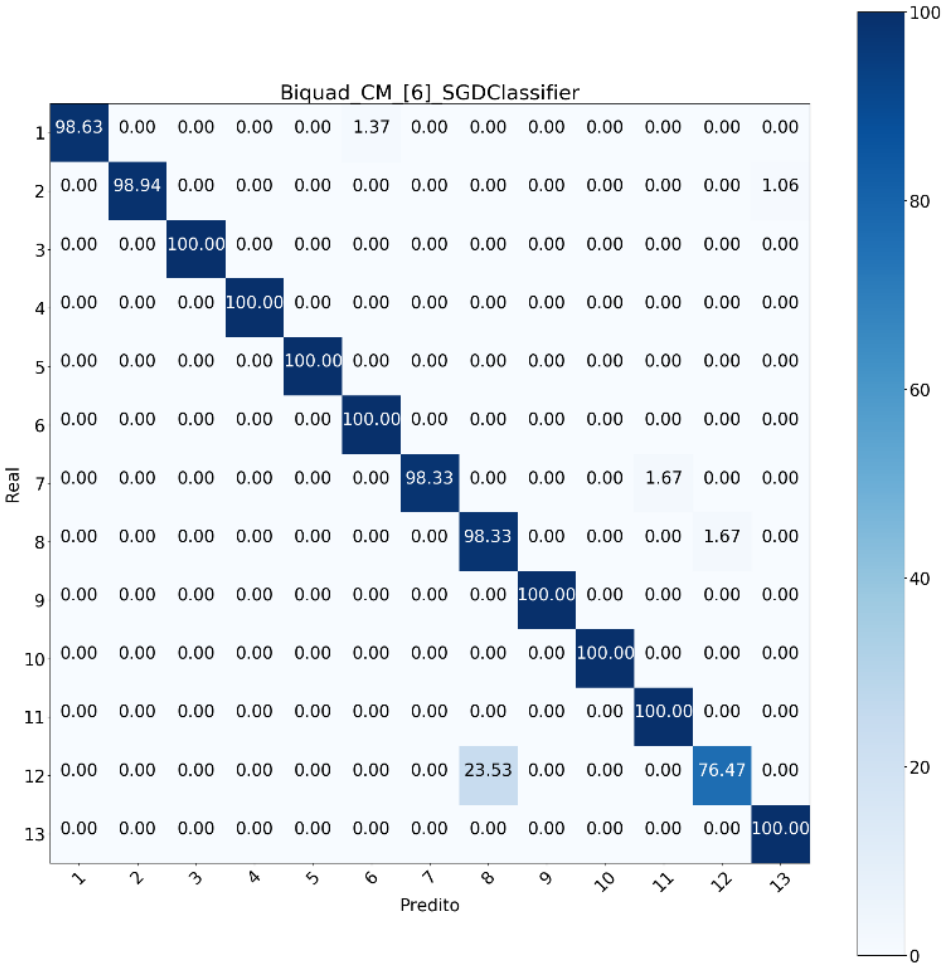
6. Resultados do KNeighbors Classifier:
    - Score: 97,67%;
    - Tempo de processamento: 9,20 segundos;



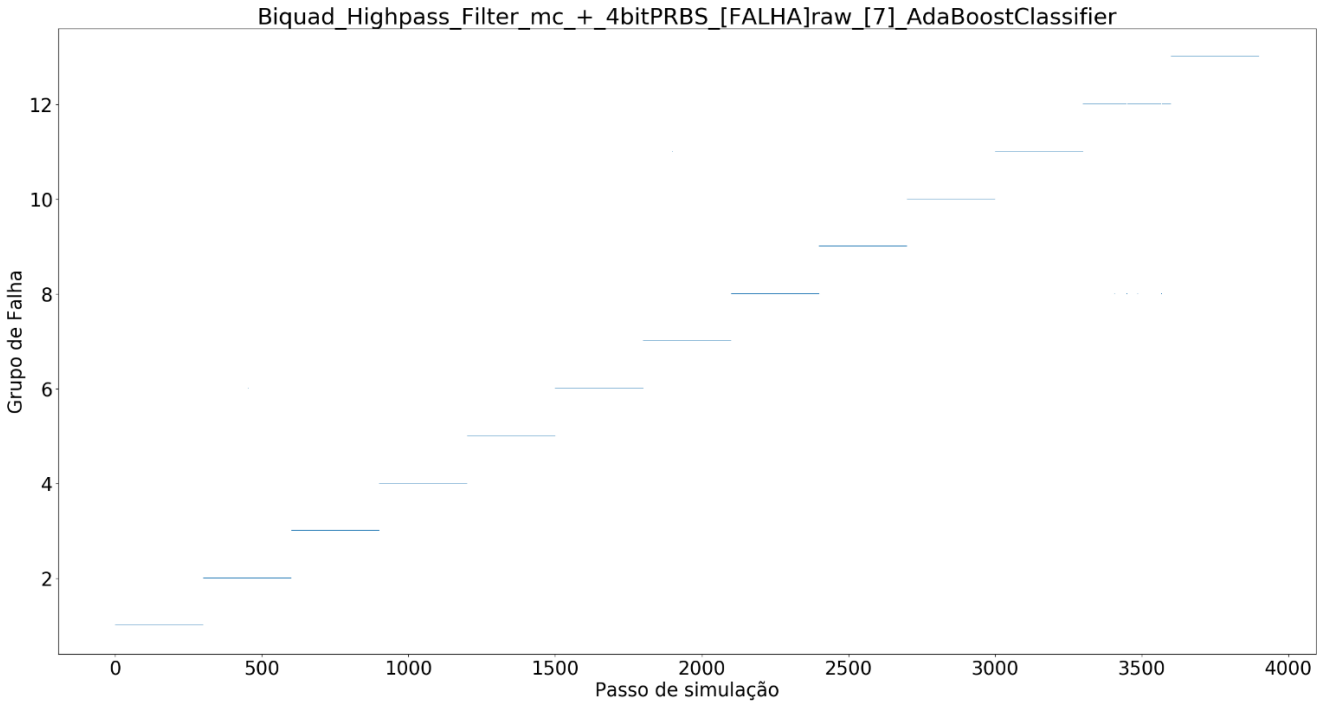


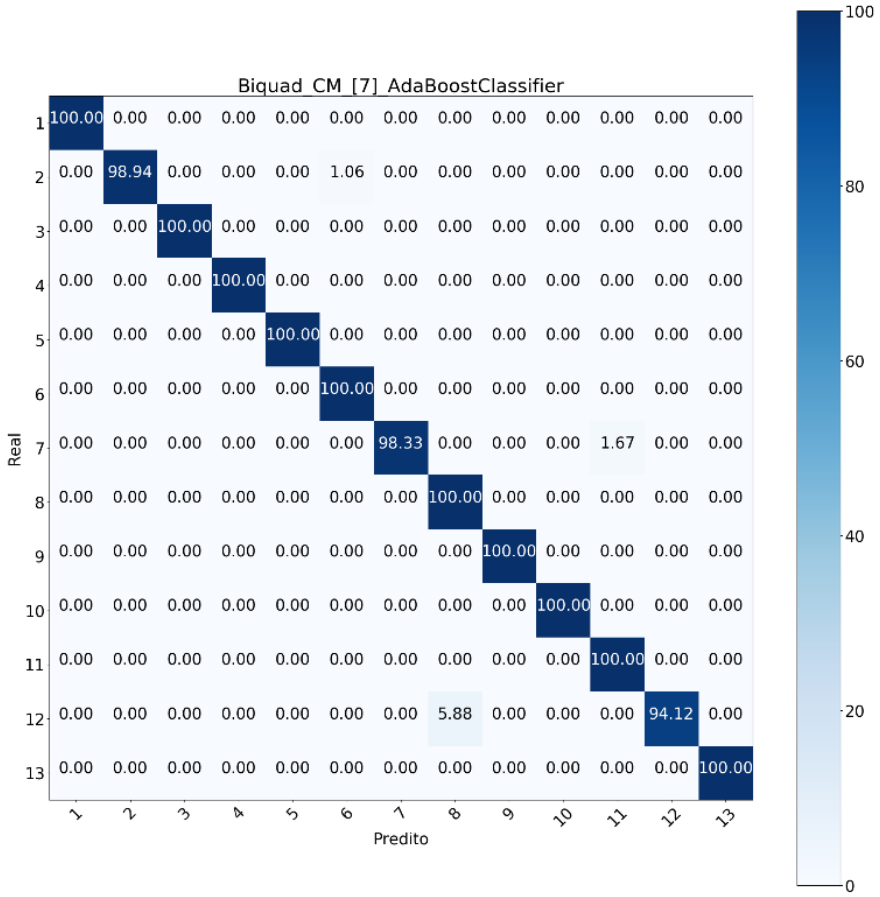
7. Resultados do Stochastic Gradient Descent:
    - Score: 97,76%;
    - Tempo de processamento: 3,09 segundos;



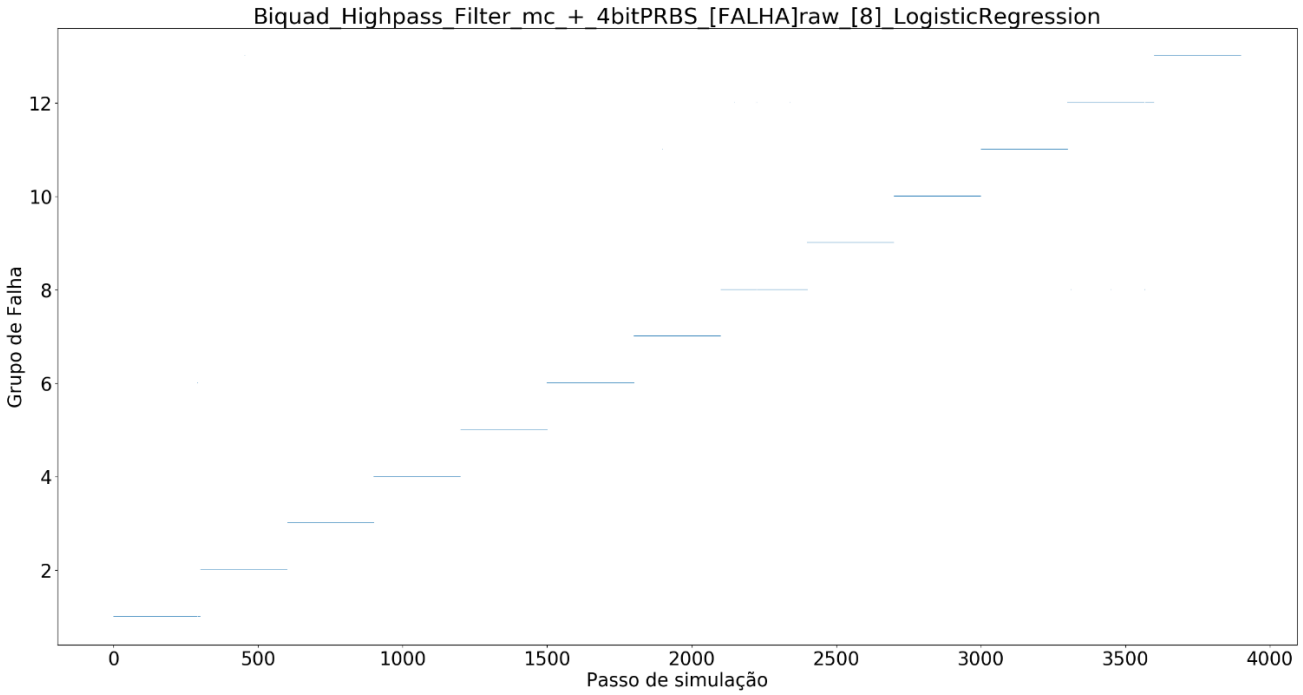


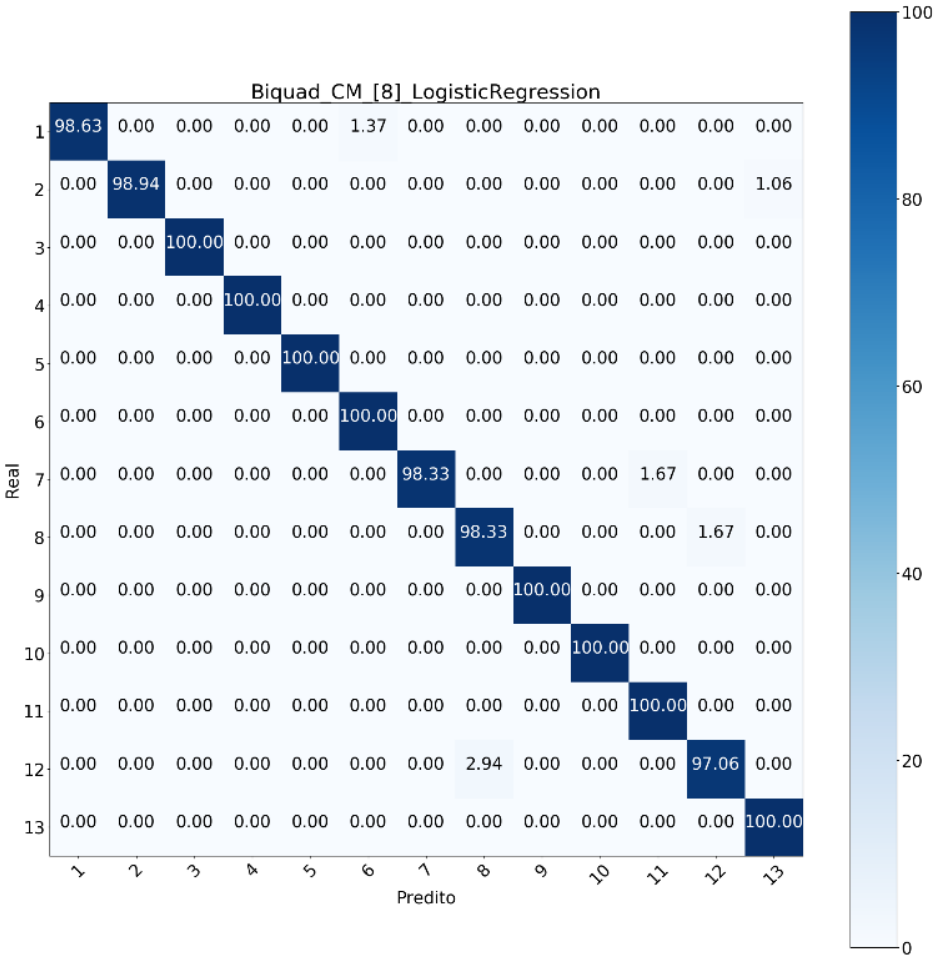
8. Resultados do AdaBoost com estimador base Random Forest Classifier:
    - Score: 99,30%;
    - Tempo de processamento: 4,55 segundos;



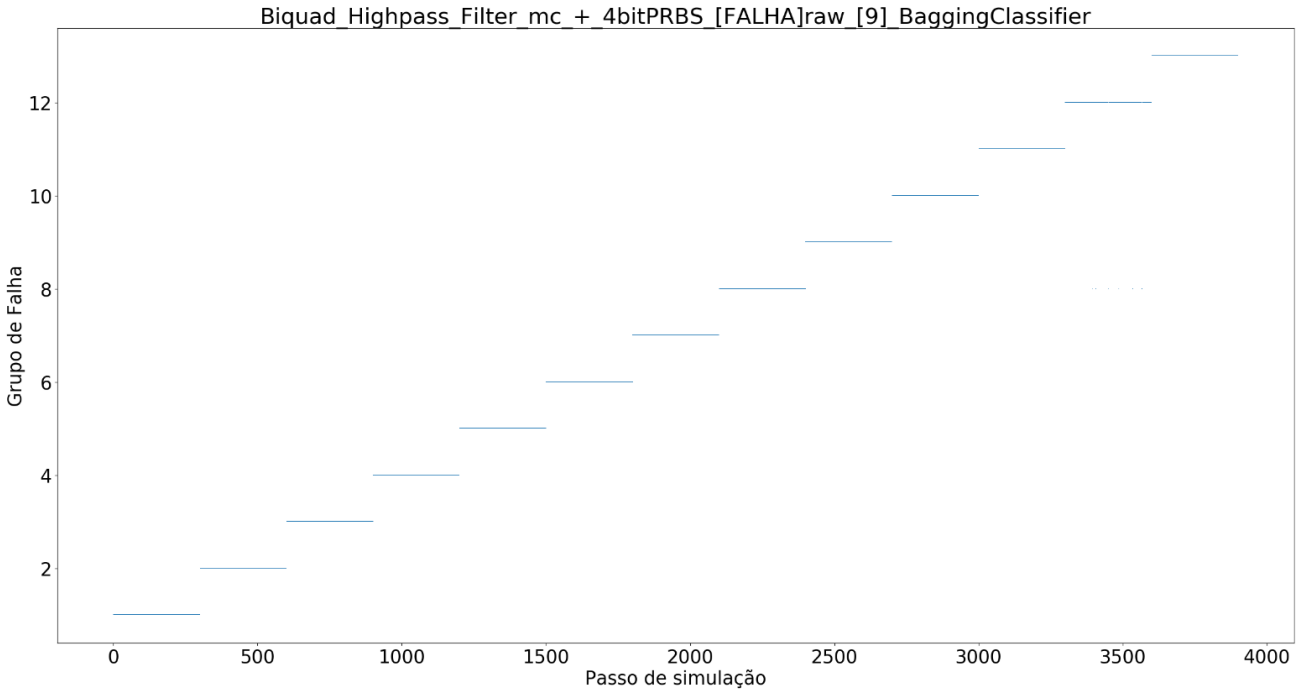


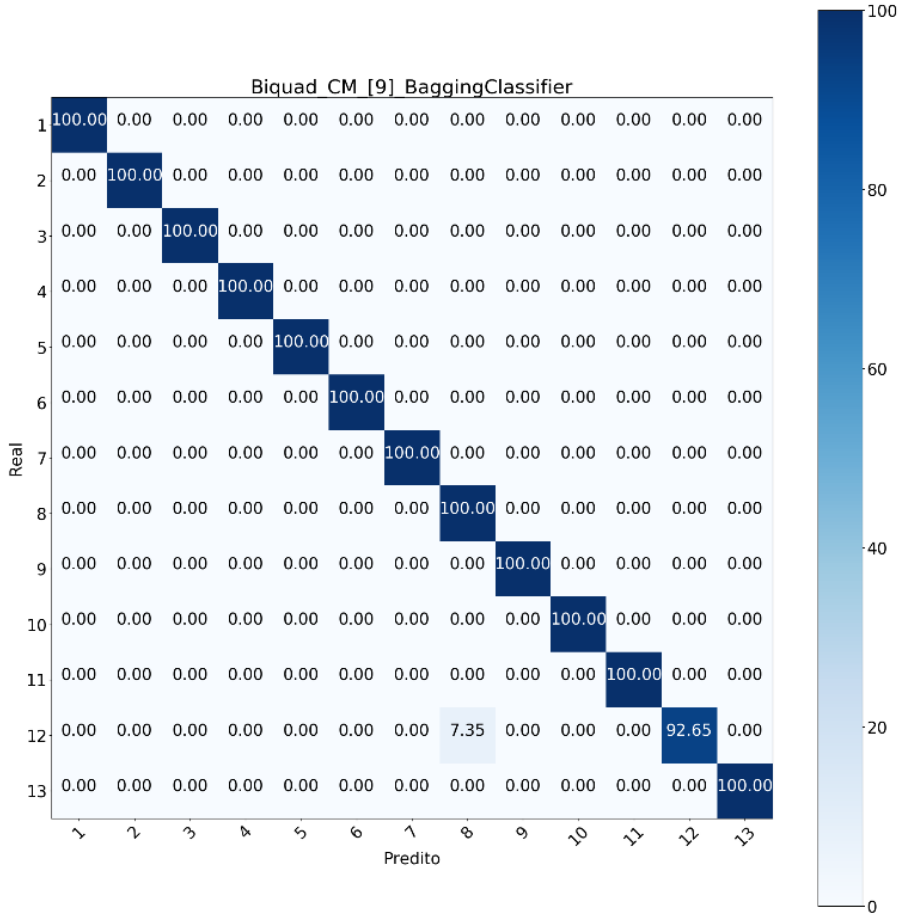
9. Resultados do Logistic Regression:
    - Score: 99,33%;
    - Tempo de processamento: 32,22 segundos;





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, este 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}
- Melhor Estimador:

  AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
  learning_rate=0.5, n_estimators=50, 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]}

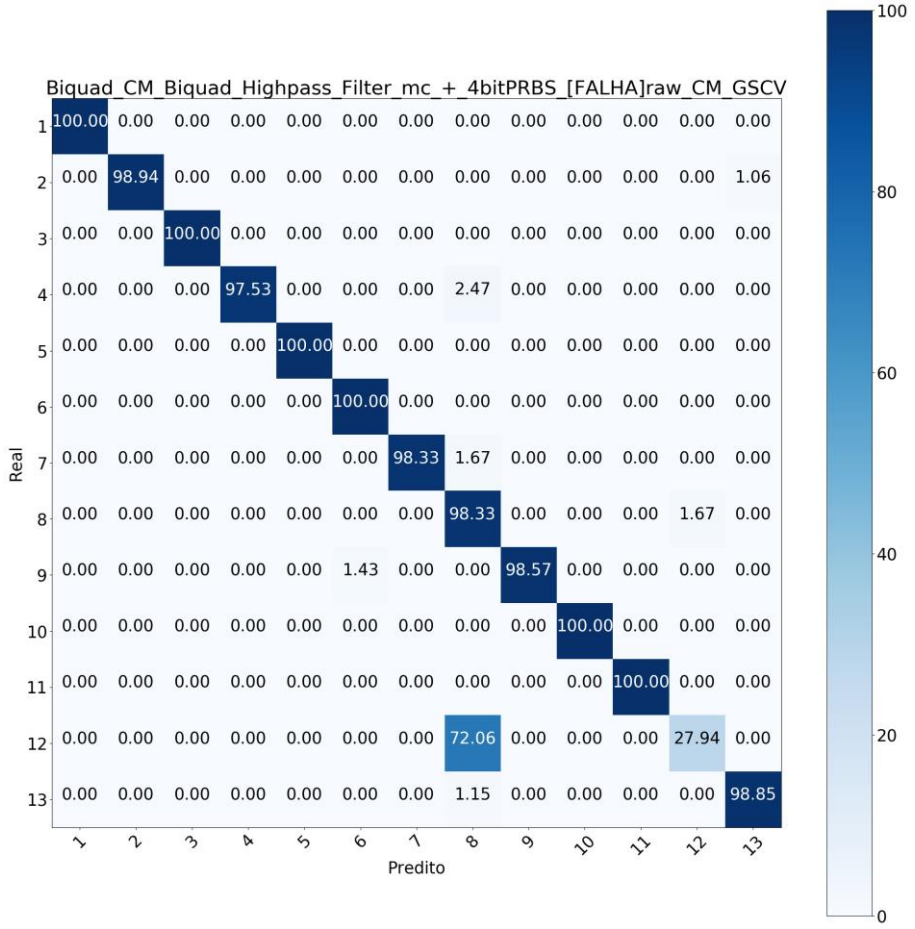


Figura 11- Confusion Matrix do método AdaBoost otimizado por GridSearchCV (GSCV)

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 diagnóstico 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 que 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 praticável. 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 uma metodologia convidativa. Ainda, neste projeto pudemos classificar as falhas com base apenas 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.

Uma possível melhoria para a aplicação em outros circuitos seria a melhor determinação dos parâmetros dos métodos aplicados, como a escolha de um *kernel* mais apropriado para o SVM, ou talvez a o *weight* no KNearestNeighbors, além, é claro, de uma determinação mais precisa dos estimadores-base para o AdaBoost. Isto aumentaria a precisão das predições se observarmos que não se pode garantir que os dados gerados pelos diversos circuitos existentes tenham as mesmas características de semelhança como visto nestes circuitos. Talvez nem mesmo estes circuitos forneçam dados com o mesmo comportamento se os expusermos a um tempo maior de simulação.

No âmbito da classificação em tempo real realço aqui o desempenho do Random Forest, que atingiu acima de 99% de precisão no menor dos tempos, sem nenhuma otimização, comprovando sua versatilidade capacidade, seguido pelo AdaBoost quando o estimador foi configurado exatamente como Random Forest, atingindo também acima de 99% de precisão e ficando atrás apenas por frações de segundos. A velocidade de treino é uma característica crucial para aplicações em tempo real.

Referências

1. https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
2. https://lamfo-unb.github.io
3. https://www.maxwell.vrac.puc-rio.br/21205/21205.PDF
4. https://medium.com/machina-sapiens/o-algoritmo-da-floresta-aleat%C3%B3ria-3545f6babdf8
5. https://www.saedsayad.com/logistic_regression.htm
6. H. Zhang (2004). The optimality of Naive Bayes. Proc. FLAIRS
7. Efron, B. (1979). Bootstrap methods: another look at the jackknife. The annals of Statistics, 1-26.
8. Efron, B, and Tibshirani, R.J. (1993). An introduction to the bootstrap. Chapman and Hall, London.