UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
MESTRADO EM ENGENHARIA ELÉTRICA

DANIEL SBORGI RIBEIRO

OTIMIZAÇÃO DO CONSUMO DE ENERGIA EM MOINHOS DE ROLOS DO TIPO *RAYMOND* ATRAVÉS DO ALGORITMO *PARTICLE SWARM OPTIMIZATION* EM UM MODELO *SURROGATE* BASEADO EM *RANDOM FOREST*

DISSERTAÇÃO

PONTA GROSSA

2021

DANIEL SBORGI RIBEIRO

OTIMIZACÃO DO CONSUMO DE ENERGIA EM MOINHOS DE ROLOS DO TIPO *RAYMOND* ATRAVÉS DO ALGORITMO *PARTICLE SWARM OPTIMIZATION* EM UM MODELO *SURROGATE* BASEADO EM *RANDOM FOREST*

RAYMOND ROLLER MILL ENERGY CONSUMPTION OPTIMIZATION THROUGH PARTICLE SWARM OPTIMIZATION ALGORITHM WITH A SURROGATE MODEL BASED ON RANDOM FOREST

Dissertação apresentada como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica, do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná.

Orientador:: Adriano Doff Sotta Gomes.

Co-orientadora: Marcella Scoczynski Ribeiro.

PONTA GROSSA

2021

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa

DANIEL SBORGI RIBEIRO

OTIMIZACÃO DO CONSUMO DE ENERGIA EM MOINHOS DE ROLOS DO TIPO RAYMOND ATRAVES DO ALGORITMO PARTICLE SWARM OPTIMIZATION EM UM MODELO SURROGATE BASEADO EM RANDOM FOREST

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Controle E Processamento De Energia.

Data de aprovação: 19 de Fevereiro de 2021

Prof Adriano Doff Sotta Gomes, Doutorado - Universidade Tecnológica Federal do Paraná

Prof Emir Baude, Doutorado - Universidade Federal do Paraná (Ufpr)

Prof.a Fernanda Cristina Correa, Doutorado - Universidade Tecnológica Federal do Paraná

Prof.a Marcella Scoczynski Ribeiro Martins, Doutorado - Universidade Tecnológica Federal do Paraná

Prof.a Virginia Helena Varotto Baroncini, Doutorado - Universidade Tecnológica Federal do Paraná

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 05/04/2021.

À minha Família

AGRADECIMENTOS

Primeiramente agradeço a Deus, por todos os privilégios e oportunidades que a vida me deu.

Aos meus pais, que sempre me apoiam e me incentivam em tudo que eu me proponho a fazer e que são um porto seguro em todos os momentos.

Agradeço a minha esposa Susi por me ajudar a seguir em frente e me apoiar sempre, principalmente nesses 4 últimos anos, onde tive que me ausentar além do normal da nossa vida familiar, seja pelas vezes em que frequentei as aulas durante o dia e trabalhei durante a noite ou, durante os fins de semana, quando fiquei muitas horas trabalhando nessa dissertação. Essa vitória é nossa e sem você ela não seria possível. Agradeço também ao meu filho Luca, que mesmo sem entender os motivos pelos quais o pai esteve ausente, sempre batia na porta do quarto dizendo que queria ”estudar com o papai”e reestabelecia a energia e a motivação necessárias para eu continuar.

Muito obrigado aos Professores do PPGEE-PG, ao meu orientador, Professor Adriano Doff Sotta Gomes, pela generosidade em aceitar orientar um trabalho fora das suas linhas de pesquisa e por contribuir com sua experiência, com tão boas ideias e sugestões. A minha co-orientadora, Professora Marcella Scoczynski Ribeiro, primeiro por dividir comigo os bancos acadêmicos durante a faculdade de Engenharia, segundo por me incentivar a começar, ao escrever uma das cartas me recomendando ao programa e, por último, por ter abraçado esse trabalho quando ele ainda era uma simples ideia.

Aos meus gestores, Aurélio Zoelner, Leopoldo Flores , Flávio Brizolla e Luciana Lima, que me incentivaram desde o início, permitindo que eu mudasse meus horários de trabalho em alguns períodos, para que eu pudesse frequentar as aulas nas manhãs e tardes na UTFPR.

Obrigado ao meu amigo Jeferson Almeida pela ajuda com o ambiente em Linux para o Python e ao meu amigo de infância Bruno Arsioli, pelas muitas horas de conversa que tivemos nos últimos meses sobre aprendizagem de máquina.

Aos colegas de profissão pela convivência diária e a todos que, direta ou indiretamente, contribuíram com esse trabalho.

A persistência é o caminho do êxito - Charles Chaplin

RESUMO

RIBEIRO, Daniel. OTIMIZAÇÃO DO CONSUMO DE ENERGIA EM MOINHOS DE ROLOS DO TIPO *RAYMOND* ATRAVÉS DO ALGORITMO *PARTICLE SWARM OPTIMIZATION* EM UM MODELO *SURROGATE* BASEADO EM *RANDOM FOREST*. 113 f. Dissertação – Programa de pós-graduação em engenharia elétrica, Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2021.

Esse trabalho propõe a utilização do *Random Forest* e do *Particle Swarm Optimization* para otimizar o consumo de energia de um moinho do tipo *Raymond*, utilizado para moagem de carbonato de cálcio natural em para baixas granulometrias. Inicialmente, os dados são filtrados, limpos e organizados para então passarem por uma etapa de pré-analise, utilizando conceitos estatísticos que incluem estrutura e dispersão, distribuição e finalmente correlação dos dados. Após isso, é aplicada a técnica *Random Forest* em um processo de aprendizagem supervisionada utilizando seleção de *features* e *tuning* de hiperparâmetros de execução, e finaliza com o processo de otimização do consumo de energia aplicando *Particle Swarm Optimization*. Os resultados demonstram um potencial de redução de aproximadamente 9% quando comparado com o consumo atual do mesmo equipamento, além da possibilidade de um ganho em escala, quando aplicada a metodologia a outros equipamentos dentro de uma mesma planta industria.

Palavras-chave: Otimização, Eficiência Energética, *Random Forest*, *Particle Swarm Optimization*

ABSTRACT

RIBEIRO, Daniel. RAYMOND ROLLER MILL ENERGY CONSUMPTION OPTIMIZATION THROUGH PARTICLE SWARM OPTIMIZATION ALGORITHM WITH A SURROGATE MODEL BASED ON RANDOM FOREST. 113 f. Dissertação – Programa de pós-graduação em engenharia elétrica, Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2021.

This work proposes the use of Random Forest and Particle Swarm Optimization to optimize the energy consumption of a Raymond mill, used for grinding natural calcium carbonate at low particle sizes . Initially, the data is filtered, cleaned and organized and then goes through a pre-analysis step, using statistical concepts that include structure and dispersion, distribution and finally correlation of the data. After that, Random Forest technique is applied in a supervised learning process with feature selection and tuning of execution hyperparameters, and ends with the energy consumption optimization process by applying Particle Swarm Optimization in two different implementations, for comparison. The results show a reduction potential of approximately 9 % when compared to the current consumption of the same equipment, also demonstrating the possibility of a gain in scale, when the same methodology is applied to other equipment within the same industrial plant.

Keywords: Optimization, Energy Efficiency, Random Forest, Particle Swarm Optimization

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CART	*Classification and Regression Trees*
OOB	*Out-of-Bag*
EAM	Erro Absoluto Médio
MIFS	*Mutual Information-Based Selection*
RF-RFE	*Recursive Feature Elimination*
CFS	*Correlation-Based Feature Selection*
RPM	Rotações por Minuto
PSO	*Particle Swarm Optimization*
API	*Application Programming Interface*
GBEST	*Global Best*
RBF	*Radial Basis Function*

SUMÁRIO

1 INTRODUÇÃO

Dados recentes mostram que as restrições nas atividades econômicas, causadas pela pandemia do coronavirus em 2020, derrubaram as demandas globais de energia em 3,8% quando comparado com os três primeiros meses de 2019. Estimativas pessimistas, que consideram os novos *lockdowns* e a manutenção da dificuldade em conter o avanço do vírus, estimam uma queda total para 2020 em torno de 6,0%, o equivalmente a sete vezes o impacto na demanda causada pela crise econômica global de 2008 e sem nenhum precedente de comparação nos últimos 70 anos (IEA, 2020a).

Apesar da demanda residencial por energia elétrica ter aumentado, devido a redução das atividades comerciais e industriais, a redução na demanda por energia elétrica pode chegar a 20% a cada mês de *lockdown* ou 1,5%, se comparado anualmente. A expectativa é que a queda global chegue a 5,0% em 2020, podendo atingir a valores acima do esperado, com a segunda onda de infecções em países desenvolvidos, principalmente no hemisfério norte (IEA, 2020a).

No Brasil, dados publicados em Agosto de 2020 pelo Ministério de Minas e Energia, através da Secretaria de Energia Elétrica, demonstram aumento de 6,0% no consumo de energia quando comparado ao mês anterior e apenas 0,2% quando comparado ao mesmo mês em 2019. Nesse cenário, destaca-se o aumento de 6,6% no consumo residencial contra quedas de até 14,2% no setor comercial e 1,6% no setor industrial. Na comparação anual até Julho de 2020, apesar de um aumento de 2,5% no consumo residencial, os setores comercial e industrial acumulam quedas de 5,5% e 4,0% respectivamente, demonstrando que o Brasil segue no mesmo cenário global citado anteriormente (ENERGIA, 2020).

A tabela 1 sintetiza o que foi citado no parágrafo anterior e apresenta um comparativo do consumo de energia elétrica no Brasil, com dados registrados até Julho de 2020 (ENERGIA, 2020).

Tabela 1: Consumo de energia elétrica no Brasil, estratificado por classes de consumo

Classe de Consumo	Valor Mensal			Acumulado 12 meses		
	Jul/2020 GWh	Evolução Mensal (Jun a Jul/2020)	Evolução Anual (Jul-19 a Jul-20)	Ago-18 a Jul-19 (GWh)	Ago-19 a Jul-20 (GWh)	Evolução
Residencial	11.703	2,7%	6,6%	140.554	144.036	2,5%
Industrial	13.864	10,7%	-1,6%	169.037	162.275	-4,0%
Comercial	5.936	5,8%	-14,2%	90.681	85.720	-5,5%
Rural	2.451	4,7%	5,5%	28.746	29.210	1,6%
Demais Classes	3.761	0,0%	-5,9%	50.407	49.404	-2,0%
Perdas e Diferenças	9.642	6,5%	7,4%	115.631	114.850	-0,7%
Total	47.357	6,0%	0,2%	595.057	585.495	-1,6%

Fonte: Adaptado de (ENERGIA, 2020)

No que tange os investimentos no mercado global de energia, no cenário pós-pandemia, a Agência Internacional de Energia publicou em Outubro de 2020 uma atualização de seu relatório anual, que havia sido publicado anteriormente em Maio deste ano. Nessa atualização, a previsão é de que os investimentos tenham uma queda de aproximadamente 18%, o que corresponde a 1,5 trilhões de dólares. Essa queda acentuada se explica pelo enfraquecimento dos balanços comerciais das empresas e também pelo cenário de incertezas sobre demandas futuras (IEA, 2020b).

Esse cenário sugere uma alteração nos padrões de investimento em demanda de energia. Enquanto o investimento em geração deve cair 7,0% em 2020, o investimento em eficiência energética deve cair em média 9,0%. Porém, as medidas para aumentar a oferta de eficiência energética podem alavancar a geração de empregos, a diminuição dos custos com energia e também a emissão de gases prejudiciais ao meio ambiente e isso a torna parte importante do processo de recuperação no período pós-pandemia (IEA, 2020c, 2020b).

No mundo, muitos programas de incentivo a eficiência energética podem ser encontrados e em muitos países, esses programas são obrigatórios e definem inclusive metas de redução de consumo de energia em determinados períodos de tempo, através de ações voltadas a eficiência. Alguns países, como Austrália, França e Itália, possuem ainda um certificado para

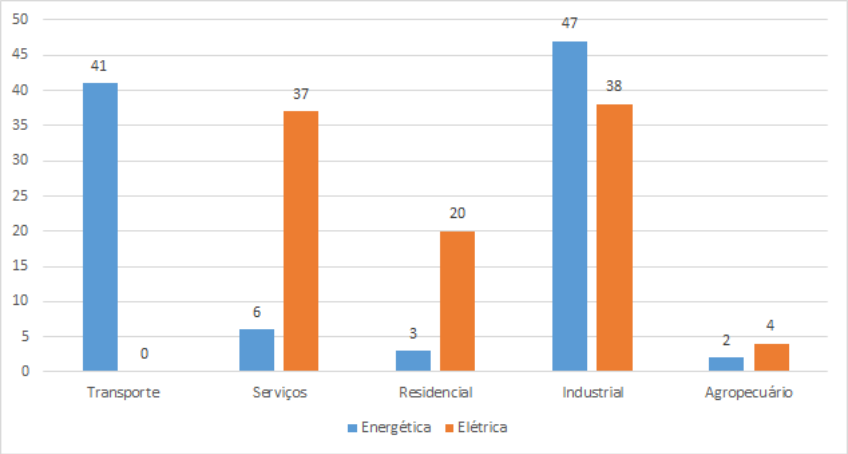
os programas, que atestam o alcance das metas estabelecidas enquanto outros países adotam inclusive leilões e licitações voltadas a programas de eficiência energética (IEA, 2020c).

Outro fator importante sobre os programas de eficiência energética é que os mesmos, quando comparados com programas de geração de energia, possuem um custo muito mais baixo de implantação. Mundialmente, o custo total do kWh em redução de consumo por eficiência fica entre 0,034 e 0,051 dólares, o que é muito menor quando comparado ao custo do kWh com eletricidade ou gás, na maioria dos lugares. No cenário nacional, reforçando essa diferença, foi registrado um aumento de 5,0% do custo de energia em 2019, quando comparado com 2018 (IEA, 2020c; CNI, 2020).

Isso indica novamente o potencial que programas de eficiência possuem nesse momento de recuperação das economias em todo o mundo, tanto para fornecedores quanto para consumidores. Os consumidores podem ter impactos positivos em suas contas de energia enquanto os fornecedores, podem evitar grandes investimentos em capacidade e/ou infraestrutura nesse momento de maior dificuldade (IEA, 2020c).

Nenhum estudo pós-covid foi publicado pelo Governo do Brasil no que se refere a eficiência energética, porém, dados do último Plano Decenal de Expansão de Energia 2029, publicado em Fevereiro de 2020, apontam que os ganhos de eficiência no Brasil podem chegar a 8,0% do consumo total do país. Nesse relatório, o setor industrial é o que apresenta o maior impacto, tanto em eficiência energética como em eficiência elétrica, como pode ser visto no gráfico 1 (ENERGIA et al., 2020).

Figura 1: Contribuição setorial para os ganhos de eficiência no ano de 2029, em porcentagem



Fonte: Adaptado de (ENERGIA et al., 2020)

Especificamente na indústria, o estudo do Governo brasileiro aponta um potencial de ganhos de 6,0%, somente em redução no consumo de energia até 2029, o que corresponde a

todo o consumo de gás natural da indústria nacional no ano de 2018. Citando apenas os ganhos de consumo com eletricidade, esse montante chega a 4,2% e correspondem a 15 TWh. A combinação de políticas nacionais e também ações autônomas das indústrias, como reforma do parque industrial, modernização da produção e ações voltadas a gestão e uso da energia elétrica são os pilares que contribuem para atingir esses resultados (ENERGIA et al., 2020).

Conforme cita Santana e Bajay (2016), existem algumas estratégias que podem ser adotadas pelas indústrias e pelos governos para promover a eficiência energética em seus processos produtivos, entre eles destacam-se:

- A difusão dos conceitos de eficiência energética para os empregados, promovendo inclusive treinamentos sobre gestão e conservação de energia;
- Facilidade em obtenção de crédito para financiamento de equipamentos industriais mais eficientes;
- Definição de parâmetros mínimos de performance de consumo de energia para equipamentos de uso geral na indústria;
- Programas obrigatórios de redução de consumo de energia, entre concessionárias e consumidores industrias, com metas a serem alcançadas;
- Incentivo de impostos para equipamentos industriais eficientes;
- Implantação de acordos voluntários entre governo e industrias, incentivando programas de eficiência energética principalmente naqueles setores onde historicamente o consumo é maior;

No Brasil, o PROCEL, que é o Programa Nacional de Conservação de Energia Elétrica, foi o primeiro programa de âmbito nacional, promovido pelo Governo Federal, para incentivar o uso eficiente de energia elétrica. O programa possui três premissas, a primeira busca suportar as indústrias nas suas estratégias de melhoria de performance energética, a segunda seleciona algumas plantas industriais para receberem novos projetos de uso eficiente de energia e a terceira, trata da disseminação de informações sobre projetos eficientes, com a intenção dos mesmos serem multiplicados. Apesar dessa iniciativa do governo brasileiro, muito pouco se obteve em termos de resultados (SANTANA; BAJAY, 2016).

O setor de mineração/processamento de minerais não metálicos destaca-se dentro do cenário brasileiro de consumo de energia, respondendo por 6,9% de tudo o que foi consumido

no país em 2018, subindo para 8,7% quando somados os gastos de processamento e extração (ENERGÉTICA, 2019).

A maior parte do uso de energia elétrica desse setor se justifica pela alto consumo de energia associado ao processo de fragmentação do mineral, desde a mina até o processamento final. Portanto, estudos que envolvam o processo de fragmentação são de grande interesse para a indústria de mineração e/ou processamento mineral, pois melhorias nesse processo impactam diretamente em redução dos custos operacionais através principalmente da redução do consumo de energia elétrica (FIGUEIRA et al., 2010).

Para efeito de comparação e usando a divisão clássica da fragmentação em fases de desmonte, britagem e moagem, o consumo de energia por tonelada de material processado aumenta em grandeza de 10x a cada uma delas. Enquanto o desmonte possui uma energia específica de aproximadamente 0,1 kWh/t, o processo de britagem opera na ordem 1 kWh/t chegando então a 10 kWh/t em circuitos de moagem. Podemos ainda encontrar, em processos de micronização/moagem fina, consumos na ordem de 100 kWh/t (JUNIOR, 2013).

A moagem, que corresponde a última etapa do processo de fragmentação, é responsável pela diminuição do tamanho de partícula a uma granulometria muitas vezes inferiores a 10micras, de acordo com mineral usado e a aplicação dos mesmos em processos subsequentes. É na moagem que estão os maiores gastos com energia e onde se concentram também os maiores investimentos do setor, o bom desempenho de uma instalação industrial de processamento de minerais depende em muito da operação de moagem e sua otimização é um desafio constante às empresas. Dessa forma, o uso mais eficiente de energia apresenta impactos não apenas financeiros, como também na demanda global por energia dessas plantas industriais (FIGUEIRA et al., 2010; JUNIOR, 2013)

Vários fatores influenciam no consumo de energia na etapa de moagem e nem todo o consumo está associado a quebra e diminuição do tamanho da partícula, o próprio movimento dos equipamentos consome boa parte da energia fornecida aos moinhos e portanto, fatores como velocidade de operação, enchimento do moinho, tamanho e carga total circulante de material, influenciam bastante no consumo final (FIGUEIRA et al., 2010).

Mais recentemente, as pesquisas em torno do processo de moagem tem aumentado bastante. Os esforços se concentram no desenvolvimento de simuladores e na aplicação de técnicas modernas de controle, inclusive com o uso de inteligência artificial, e elas tem sido utilizadas com sucesso pela indústria (FIGUEIRA et al., 2010)

Dentre os inúmeros minerais não metálicos que são processados, o carbonato de cálcio

possui destaque no cenário mundial e ele está sempre presente, desempenhando um papel muitas vezes invisível, em vários processos produtivos modernos. De forma simplificada, podemos citar aplicação de carbonato de cálcio em industrias de cimento, papel, plástico, tinta, vidro, cerâmica, agricultura, alimentação de animais, metalurgia, tratamento de água e higiene pessoal, entre outros (SAMPAIO; ALMEIDA, 2005).

Algumas dessas industrias, como as de plásticos, papel e tintas a aplicação do carbonato de cálcio é feito em granulometrias conhecidos como ultrafinas, com tamanho médio de partícula abaixo de 10 micras e nesse cenário, o consumo de energia aumenta significativamente. Paralelamente, muito devido a raridade de esforços dirigidos a modernização desse setor produtivo, muitas empresas ainda buscam equipamentos de custo mais baixo ou já consolidados no mercado há vários anos e sem muita inovação, quando a economia de energia não era uma exigência de mercado. Nesse cenário, pode-se incluir os moinhos de rolos do tipo *Raymond* ou seus equivalentes mais recentes e otimizados, amplamente utilizados em moagem de carbonato de cálcio do tipo seca (SAMPAIO; ALMEIDA, 2005)

Sendo assim, torna-se relevante estudos que busquem otimizar o consumo de energia em equipamentos de mineração/moagem, primeiro pelo fato deles representarem grande parte do consumo de energia nesse tipo de indústria e segundo, pelo momento atual, onde os temas que relacionam eficiência energética devem ser colocados em evidência visto a baixa disponibilidade de recursos para novos investimentos.

1.1 OBJETIVO GERAL

Otimizar o consumo de energia elétrica em um moinho do tipo *Raymond* usado para moagem de carbonato de cálcio natural, utilizando o algoritmo *Particle Swarm Optimization* em um modelo *surrogate* obtido através de *Random Forest*.

1.2 OBJETIVOS ESPECÍFICOS

Objetivos específicos desse trabalho:

- Desenvolver a modelagem de um moinho *Raymond* através da estratégia de modelo *surrogate* implementado com a técnica de *random forest*;
- Aplicar técnicas de seleção de *features* com o objetivo de otimizar a execução da modelagem, reduzindo dimensionalidade do conjunto de dados, sem penalizar o

desempenho;

- Utilizar a técnica de *Grid Search* para obtenção de parâmetros otimizados para execução do *Random Forest*;
- Demonstrar como modelos computacionais obtidos com técnicas de *machine learning* podem ser utilizados na otimização do consumo de energia em processos industriais;
- Otimizar o consumo de energia do moinho em estudo, demonstrando os parâmetros ideais de operação encontrados através do algoritmo *Particle Swarm Optimization*.

1.3 JUSTIFICATIVA

Os gastos com energia elétrica correspondem a grande parte das despesas de operação de uma indústria de processamento mineral. Seja em processos altamente automatizados ou em simples processos manuais de produção, equipamentos elétricos sempre estarão presentes e portanto, iniciativas que busquem otimizar o consumo de energia passam a ser vitais a qualquer negócio.

O processo de otimizar o consumo nem sempre é viável, muitas vezes a complexidade ou a falta de conhecimento profundo sobre os equipamentos inviabiliza a obtenção de modelos que possam representar com fidelidade o processo produtivo em questão, o que torna a otimização de operação dos equipamentos, com consequente redução do consumo de energia, cada vez mais distante da realidade.

Além disso, o baixo custo de implementação, o tempo relativamente curto para obtenção dos resultados e a possibilidade de replicar a solução a outros equipamentos e aumentar o ganho com economia de energia em escala, justificam esse trabalho.

1.4 ORGANIZAÇÃO

Esse trabalho foi dividido em outros cinco capítulos. No primeiro deles, será apresentada uma revisão bibliográfica dos principais temas abordados e que sustentará toda a prática aplicada nos capítulos subsequentes. No capítulo 3, será tratada de parte da metodologia aplicada ao trabalho, concentrando na preparação dos dados e análises preliminares dos mesmos. O capítulo 4 é dedicado a modelagem utilizando o *Random Forest* enquanto no capítulo 5, a execução do processo de otimização usando o *Particle Swarm Optimization*. No capítulo 6 é apresentada a conclusão e as considerações finais.

2 REFERENCIAL TEÓRICO

2.1 MOINHOS DO TIPO *RAYMOND*

Os primeiros moinhos desse tipo surgiram em 1906 na Alemanha e com o passar das décadas o seu projeto foi sendo aprimorado. São conhecidos como um dos mais antigos moinhos de rolos usados em plantas de moagem e foram originalmente projetados para transformar carvão comum em pó de carvão, que era então destinado a obtenção de vapor em plantas geradoras de energia, tendo facilmente substituído, na época, outros modelos de moinhos devido a seu baixo consumo de energia e pouco espaço físico necessário para instalação. Foram gradativamente sendo usados também para o processamento mineral, quando outros moinhos mais eficientes surgiram para o processamento de carvão (WHEELDON et al., 2015; SBMGROUP, 2020)

Em geral, esses moinhos possuem a capacidade de processar minerais com tamanhos máximo de partícula em D97,3 entre 20 e 200micras (97,3% de todo o material abaixo do tamanho máximo informado, que nesse caso pode variar entre 20 e 200micras conforme a especificação do equipamento) porém, com algumas aplicações já exigindo D97,3 $<$ 10micras (WHEELDON et al., 2015).

Exemplo de alguns moinhos *Raymond* pode ser encontrado na figura 2 abaixo.

Figura 2: Moinhos de rolos do tipo *Raymond* de diferentes fabricantes

Fonte: (ZENIT, 2020; GYPMAK, 2020; CHAOYANG, 2020)

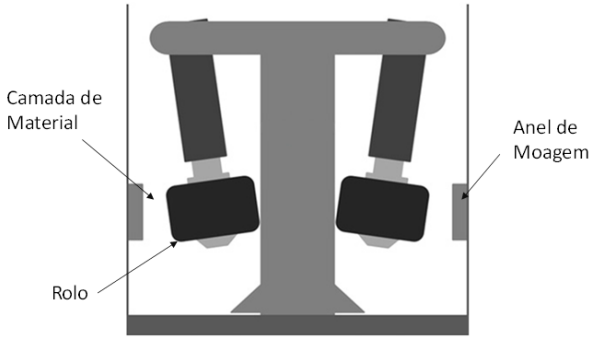
Conforme cita o grupo Chinês Shibang Machinery, um dos muitos fabricantes desses moinhos atualmente, a unidade principal da planta de moagem, que pode ser representada pelo moinho e seu elemento central interno, permite três tipos distintos de movimento girante, sendo o primeiro deles o movimento dos rolos ao redor do eixo central, o segundo, um movimento de revolução em torno do anel de moagem e por fim, o movimento causado pelo atrito entre o rolo, o material processado e o anel de moagem, sendo esse último o responsável pela moagem uniforme e também pelo desgaste uniforme das partes envolvidas, trazendo assim uma vantagem em relação a vida útil do equipamento (SBMGROUP, 2020).

Cada um dos rolos de moagem é parte de um pêndulo, que fica fixado na parte superior a uma peça chamada de cabeça cruzada, que por sua vez está acoplada ao eixo central tracionado. Quando o motor do moinho é acionado, o eixo se movimenta rotacionando também a cabeça cruzada e, em um movimento centrífugo, os rolos de moagem são forçados contra o anel de moagem. Por essa característica, as forças que determinam a moagem do material processado aumentam conforme o aumento na rotação do motor do moinho (WHEELDON et al., 2015).

O processo de moagem em moinhos do tipo Raymond é descrito pelas etapas a seguir (WHEELDON et al., 2015; SBMGROUP, 2020; ZENIT, 2020; HENAN, 2020; GYPMAK, 2020). A figura 5 representa um fluxograma do mesmo processo.

1. Em plantas com esse tipo de moinho, o material a ser processado fica geralmente armazenado em silos de alimentação e é então enviado ao equipamento por algum tipo de alimentador automático. Ao ser introduzido na câmara de moagem, o material vai ao fundo e é impulsionado por facas inclinadas que são instaladas logo abaixo dos rolos e que se movimentam junto aos mesmos. Essas facas movimentam o material do fundo do moinho, projetando-os no espaço existente entre os rolos e os anéis de moagem, formando assim uma camada de material ao redor do anel (o ar soprado pelo ventilador de recirculação tem função importante nesse processo). Esse material é então pulverizado para fora através da força centrífuga gerada com o movimento dos rolos de moagem. Na figura 3 estão representados os rolos, o anel de moagem e a camada de material;

Figura 3: Representação da câmara de moagem de um moinho *Raymond*



Fonte: Adaptado de: (WHEELDON et al., 2015)

2. O movimento ascendente do material dentro do corpo do moinho leva o mesmo de encontro ao classificador, que é o equipamento responsável por classificar o material que está dentro da especificação de moagem, reintroduzido o que for rejeitado para que seja novamente processado. O classificador é um equipamento girante, que possui inúmeras pás instaladas ao redor do seu eixo principal, com o espaçamento entre elas definido de acordo com a especificação do produto final que se deseja obter com o processo de moagem. A distribuição do tamanho da partícula do material que está sendo processado pode ser ajustada por uma combinação do fluxo de ar interno e da velocidade do classificador;

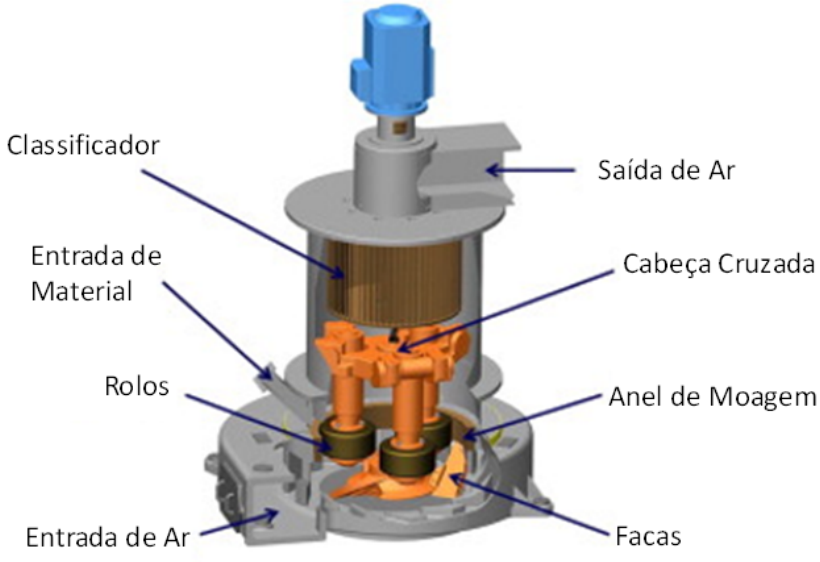
   A figura 4 representa o moinho em corte, onde pode-se notar a existência dos elementos citados nos itens anteriores.

3. Para que o material possua força de arraste suficiente para passar pelo classificador, emprega-se um ventilador centrífugo de recirculação, que succiona ar + material no topo de um segundo equipamento chamado ciclone e o empurra de volta na base, criando assim um movimento ascendente dentro do corpo do moinho;

4. O ciclone é responsável por desacelerar o material que vem do classificador e, em sua base, é onde geralmente é feita a coleta para destinação final do produto já processado e dentro das especificações. O ciclone é um equipamento estático, com formato cônico, e desacelera o material que adentra o seu topo, justamente por ter esse formato;

5. O último equipamento empregado em plantas com moinhos do tipo *Raymond* é o filtro de despoeiramento. Esse filtro coleta o material mais fino de todo o processo, material que geralmente fica em circulação e precisa ser removido por questões de produtividade, estabilidade do moinho e também consumo de energia. Esse filtro, usualmente montado

Figura 4: Moinho *Raymond* representado em corte

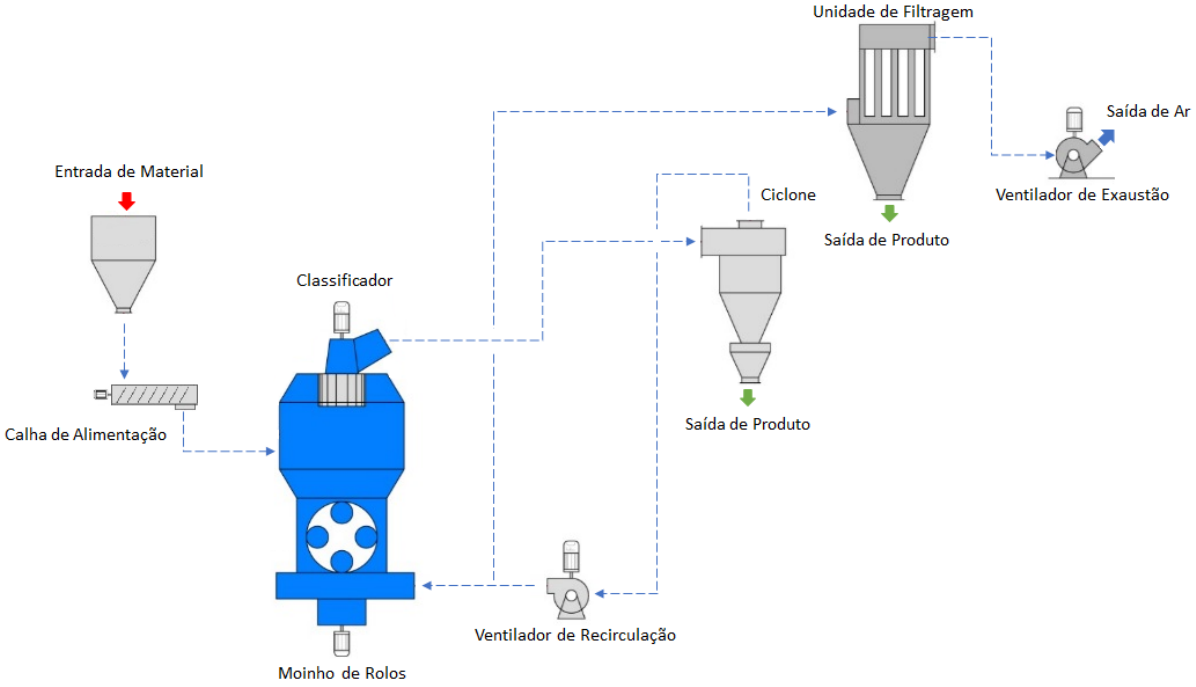


Fonte: Adaptado de: (WHEELDON et al., 2015)

internamente com mangas ou bolsas de filtragem, possui um ventilador exaustor que traz o material fino de encontro ao elemento filtrante, que retém o material e permite a passagem do ar, ficando impregnado até que algum sistema de limpeza (ar comprimido é o mais empregado) faça a purga do material para fora do filtro. Esse material, apesar de em geral ser mais fino que o material retirado pelo ciclone, está em menor quantidade e dependendo da especificação do produto, pode também ser usado e incorporado ao volume final processado.

A figura 5 representa o fluxograma de processo de uma planta com moinho *Raymond* enquanto a figura 6, a projeção de uma planta de processamento mineral com o mesmo tipo de moinho.

Figura 5: Fluxograma de processo



Fonte: Adaptado de: (GYPMAK, 2020)

Figura 6: Planta de operação com moinho *Raymond*

Fonte: Adaptado de: (XINGYANG, 2020)

Esse trabalho, como já citado em capítulos anteriores, trata da otimização do consumo de energia em um moinho de rolos do tipo *Raymond* através do ajuste de seus parâmetros de

operação, que permitam o mesmo a operar em condições otimizadas e sem comprometer a sua capacidade de produção e/ou a qualidade do produto final produzido.

O estudo foi baseado em um moinho fabricado pela Alstom, modelo 6669, com potência máxima no motor dos rolos de 220kW, processando carbonato de cálcio calcítico e dolomítico em diferentes granulometrias e capacidades de produção.

O cálculo do consumo total de energia do equipamento engloba a soma das potências instantâneas do motor dos rolos, do ventilador de recirculação, do classificador e do ventilador exaustor. Por representarem potências muito baixas individualmente e coletivamente, os motores de menor capacidade foram desprezados no estudo.

No que se referem aos parâmetros de operação, serão considerados no total 17 deles e os mesmos incluem pressões, temperaturas, vibrações e velocidades de rotação de motores. Os parâmetros não serão expostos por questões de segredo industrial e propriedade intelectual. Dos 17 parâmetros amostrados, 5 deles representam parâmetros de *setpoint* de operação enquanto os restantes são parâmetros de visualização e acompanhamento de processo apenas.

Uma das grandes dificuldades de utilizar uma abordagem mais convencional para resolver esse tipo de problema se dá pela multidimensionalidade da equação matemática envolvida em uma modelagem tradicional. Nesse caso, optou-se em aproveitar a funcionalidade dos modelos de *Machine Learning*.

A proposta, além de ser inovativa, visto que não existem muitos trabalhos publicados que englobem esse tipo de equipamento e nem mesmo a mesma metodologia, traz à tona a possibilidade de desenvolver aplicações que busquem um maior grau de eficiência energética para a indústria e que apresentam baixíssimo custo de implantação.

2.2 MODELOS DE PREDIÇÃO

Em muitas áreas da ciência, das finanças, das engenharias, entre outras, a aprendizagem estatística ocupa um lugar fundamental. Usando os dados disponíveis para um determinado problema, modelos de predição ou de aprendizagem podem ser construídos e nos auxiliar a prever os resultados ainda não vistos (HASTIE et al., 2009).

Em outro aspecto, mais particular, pessoas se encontram todos os dias com situações como: Qual o melhor caminho para o trabalho?, Onde eu deveria investir o meu dinheiro?, Será que eu terei alguma doença grave no futuro ou, Qual deveria ser o meu salário baseado nos meus conhecimentos? Todas essas situações indicam que, a todo tempo, pessoas estão

tentando prever algum evento futuro para que possam, o quanto antes, tomar a melhor decisão em relação a ele. Em geral, essas decisões são tomadas baseadas em experiências pelas quais elas já passaram e que as ajudam a tomar as decisões que farão no futuro (KUHN; JOHNSON, 2013).

Alguns exemplos de predição são citados por (HASTIE et al., 2009; KANT; SANGWAN, 2015; QIN et al., 2018):

- Na medicina, prever se um paciente terá um ataque do coração baseado em sua dieta, na demografia e nas analises clínicas do mesmo ou, estimar a quantidade de glicose no sangue de um paciente diabético utilizando absorção do espectro infravermelho em seu sangue;
- Nas finanças, prever o valor futuro de uma ação do mercado de renda variável, baseado na performance da empresa em questão e de outros dados econômicos;
- Na engenharia, modelar o consumo de energia de equipamentos, buscando ganhos econômicos e paralelamente, sustentáveis;
- Na agronomia, prever a taxa ótima de nitrogênio do milho, utilizando imagens aéreas;

Kuhn e Johnson (2013) traz alguns exemplos modernos de aplicação de modelos de predição: O *google* usa modelos de predição para interpretar as consultas humanas no seu mecanismo de busca, companhias de cartão de crédito aplicam modelos de predição para rastrear fraudes em seus sistemas e o *netflix* usa modelos de predição para recomendar alguns de seus filmes a seus usuários. O autor ainda cita, que as companhias de seguro gostariam, por exemplo, de prever os riscos de acidentes de seus segurados para então oferecer o melhor plano de seguro para cada perfil, que os governos buscam prever riscos que envolvam os seus cidadãos como a identificação de terroristas através de sistemas de biometria ou modelos que façam a previsão de fraudes, entre outros, demonstrando que atualmente, os modelos de predição permeiam a nossa existência.

Os exemplos citados nos parágrafos anteriores mostram que uma grande quantidade de dados está disponível para criação dos modelos de predição e que, com o avanço da tecnologia em computadores pessoais e da grande facilidade em encontrar ferramentas disponíveis gratuitamente, torna relativamente fácil o desenvolvimento de modelos preditivos porém, com o contraponto de que, conforme os dados e ferramentas foram ficando cada vez mais disponíveis, a qualidade dos modelos criados começou a cair (KUHN; JOHNSON, 2013).

Um conceito de modelo de predição ou modelo preditivo é encontrado em Kuhn e Johnson (2013): Modelo de predição é o modelo criado ou escolhido para tentar prever a probabilidade de ocorrência de uma determinada saída ou resultado.

Nos modelos de predição, então, o objetivo do algoritmo é prever os valores de saída e para isso, ele utilizada os dados de entrada. Assim, os dados de entrada são dados medidos ou amostrados, enquanto os dados de saída são dados de predição. Estatisticamente falando, os dados de entrada são chamados de preditores ou mais classicamente, de variáveis independentes porém, é comum encontrar em alguns autores o conceito de *feature* (característica, do inglês). Devido aos tipos de saídas existentes, as mesmas são ditas como regressão, quando o valor de predição representa uma variável quantitativa ou, classificação, quando a mesma representa uma variável qualitativa (HASTIE et al., 2009; BRAMER, 2007). Mais especificamente sobre modelos de regressão, o conceito está associado a fazer predições e suas respostas são expressas através de uma combinação dos dados de entrada. Esse processo está associado a uma transição de dados para modelos, onde esses modelos podem ser úteis em diferentes tarefas, como por exemplo (IGUAL; SEGUÍ, 2017):

- Analisar o comportamento de algum dado ou da relação entre os dados;
- Prever um acontecimento futuro ou predizer um valor, baseado em dados de entrada;
- Qualificar variáveis para o modelo.

2.3 MODELOS *SURROGATES*

A solução de problemas complexos muitas vezes exige técnicas adicionais para melhorar a eficiência nos seus principais gargalos que são a avaliação da função de atribuição de *fitness* e a construção do modelo (HAUSCHILD; PELIKAN, 2011).

Um dos métodos utilizados para melhorar o desempenho dos algoritmos com alto custo computacional na avaliação do *fitness* é a possível eliminação ou relaxamento de algumas das avaliações por meio de modelos aproximados da função de *fitness*. Estes modelos podem ser avaliados mais rápido do que a função de *fitness* efetiva. Técnicas de aumento de eficiência com base neste princípio são chamadas técnicas de relaxamento de avaliação ou modelos *Surrogate*.

Modelos *surrogate* buscam determinar uma função contínua (f) baseados num conjunto limitado de dados referentes a essa função. Nesse caso, os modelos podem ser classificados como um problema de inversão não-linear. Os dados disponíveis referentes a função, apesar de serem determinísticos por natureza ao representar exatamente valores de (f), em geral não

possuem informação suficiente para que a função (f) seja identificada a partir deles (QUEIPO et al., 2005).

Segundo Ong et al. (2003), modelos *surrogate* são modelos construídos com o objetivo de aproximar computacionalmente os modelos de simulação usados em alguns problemas, permitindo as mesmas análises que as simulações permitiriam, porém de forma muito mais barata.

Queipo et al. (2005) cita que a construção de modelos *surrogate* passa por duas alternativas, a primeira alternativa é a paramétrica onde a relação entre as variáveis resposta e as variáveis de projeto do modelo são conhecidas, e a não-paramétrica, que está relacionada a construção de diferentes tipos de modelos simples ou locais, em diferentes regiões do espectro de dados, para que seja então criado um modelo geral.

Como exemplo de modelos paramétricos, pode-se citar a regressão polinomial e o método de *kriging*, por exemplo, enquanto para modelos não-paramétricos são citadas as redes neurais de base radial e modelos de regressão baseados em *kernel* (*suppor vector machines*, por exemplo) (ONG et al., 2003; QUEIPO et al., 2005; SUN et al., 2017; DASARI et al., 2019).

De acordo com Jin (2011), os modelos *Surrogates* podem ser aplicados a quase todas as operações dos algoritmos evolutivos, como inicialização da população, busca local e avaliação da função de *fitness*. Uma variedade de modelos computacionais, incluindo polinomiais e redes neurais, juntamente com técnicas de amostragem de dados estão presentes. Para a avaliação da função de *fitness*, o gerenciamento dos *Surrogates* pode ser baseado em indivíduos, nas gerações e na população. Os modelos baseados na geração utilizam os *Surrogates* para avaliações de *fitness* em algumas gerações, enquanto no restante a função real de *fitness* é utilizada. Nos modelos baseados em indivíduos, a função real de *fitness* é utilizada na avaliação de alguns indivíduos na mesma geração. E os modelos baseados em população mais de uma população co-evolui, cada uma usando seu próprio modelo *Surrogate* para a avaliação de *fitness*.

Ainda, segundo afirma Jin (2011), na maioria dos problemas do mundo real, não existem funções *fitness* analíticas para avaliar o resultado de uma solução candidata. Em geral, existem métodos que estimam o resultado, com menos ou mais acuracidade e esses métodos estão relacionados a mais ou menos recursos computacionais disponíveis e/ou empregados.

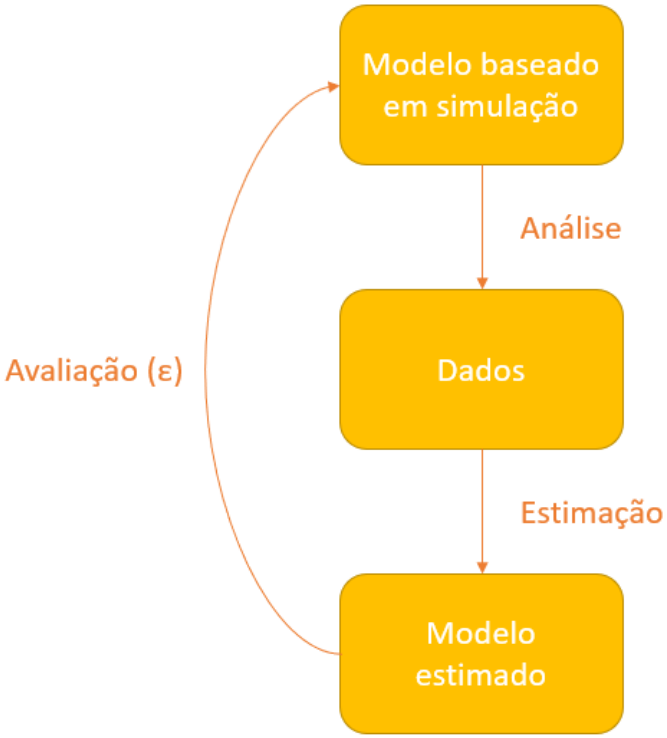
De acordo com Hauschild e Pelikan (2011), existem duas abordagens básicas para as técnicas de relaxamento de avaliação: modelos endógenos (SMITH et al., 1995; SASTRY et al., 2004; SASTRY, 2001; PELIKAN; SASTRY, 2004) e modelos exógenos (ALBERT, 2001; SASTRY et al., 2001). Com modelos endógenos, os valores de *fitness* para algumas soluções

candidatas são estimados no modelo de distribuição de probabilidade, com base no *fitness* das soluções previamente geradas e avaliadas (herança de *fitness*). Com modelos exógenos, um modelo substituto mais rápido é usado para algumas das avaliações através da substituição da função de *fitness* complexa e de alto custo computacional por um procedimento de atribuição de *fitness* aproximado de baixo custo, o que reduz de forma eficiente a carga computacional. As duas abordagens podem ser combinadas para maximizar os benefícios.

No trabalho de Sastry et al. (2004) são apresentadas técnicas endógenas utilizando a herança de *fitness* para o aumento da eficiência em algoritmos. Modelos probabilísticos das soluções filhas são desenvolvidos para estimar o *fitness* de uma proporção de indivíduos na população, evitando, assim, avaliações computacionalmente caras.

Já no trabalho de Sastry et al. (2001) é apresentado um estudo de três esquemas exógenos. Cada um desses esquemas é modelado usando modelos dedicados e são usados para desenvolver modelos de dimensionamento de população e de tempo de convergência. O trabalho de Albert (2001) justifica o fato de que em muitos problemas, os parâmetros de granulação grossa *(coarse-grained)* podem ser utilizados para diminuir o tempo de computação através da introdução de erros nas avaliações de *fitness*. Neste trabalho, a autora se concentra em como esses tipos de problemas podem ser executados de forma eficaz e com precisão ao lidar com quantidades consideráveis de erro na avaliação. O objetivo final do trabalho de Albert (2001) é estabelecer diretrizes eficazes para os algoritmos genéticos que usam uma discretização finita para aproximar um sistema contínuo. Para atingir este objetivo, a compreensão dos efeitos do erro de avaliação sobre o desempenho do algoritmo deve ser obtida, e um modelo deve ser criado que possa prever tanto a exatidão quanto os requisitos computacionais em um ambiente com avaliação de erro presente.

Figura 7: Anatomia básica de um modelo *surrogate*



Fonte: Adaptado de (QUEIPO et al., 2005)

A figura 7 representa basicamente a anatomia de um modelo *surrogate*. A partir da análise de um problema e da obtenção dos seus dados, um modelo é estimado e avaliado e permite prever o erro associado ao mesmo ao fazer estimativas referentes a função modelada. Assim, modelos *surrogate* permitem duas abordagens distintas em relação ao problema, primeiro estimando uma função baseado nos dados disponíveis relacionados a essa função e segundo, mensurando o erro relacionado a essa estimativa (QUEIPO et al., 2005).

Matematicamente, o modelo *surrogate* pode ser representado pela equação 1 onde $\hat{f}(x)$ representa a saída predita pelo modelo e $\varepsilon(x)$ representa o erro de predição. A função contínua $\hat{f}(x)$ será determinada por um conjunto de dados $x = x1, ...x_n$ e suas respectivas saídas (como em um processo de aprendizagem supervisionado em *Machine Learning*) (QUEIPO et al., 2005; DASARI et al., 2019):

$$f_p(x) = \hat{f}(x) + \varepsilon(x) \tag{1}$$

A construção do modelo passa por quatro estágios, conforme cita (DASARI et al.,

2019):

1. Análise experimental: Selecionar as variáveis que serão usadas na construção do modelo, $x = (x_1, ...x_d)$;
2. Simulações numéricas: Avaliar a função f nos pontos $y_i = f(x_i)$ onde $x_i \in \mathbb{R}^d$ e $y_i \in \mathbb{R}$;
3. Construção do modelo: Baseado em um conjunto de dados com entradas e saídas, realizar o processo de definição da função $\hat{f}$ para avaliar novos pontos $\hat{y} = f(\hat{x}_i)$;
4. Validação do Modelo: Avaliar a performance de predição do modelo baseado em dados de teste disponíveis.

A figura 8 representa graficamente a as etapas de construção de um modelo *surrogate*:

Figura 8: Etapas de construção de um modelo *surrogate*

Fonte: Adaptado de (DASARI et al., 2019)

A medida mais comum para avaliar a qualidade de um modelo *surrogate* é o erro quadrático médio ou *mean-squared error*-MSE que mede a diferença entre um resultado real e o resultado predito pelo modelo (JIN, 2011).

2.4 PREPARAÇÃO DOS DADOS E PRÉ-PROCESSAMENTO

Para que os dados coletados possam ser usados em um processo de aprendizagem e/ou predição, os mesmos precisam estar organizados. Em geral, os dados não são encontrados prontos para serem utilizados, eles podem estar, por exemplo, registrados em *logs* bastante complexos e extensos, podem estar misturados com outros dados não relevantes, podem estar

codificados ou mesmo pertencerem a fontes diferentes e portanto, armazenados em locais ou formas distintas (AGGARWAL, 2015).

Para que os dados possam ser usados, eles devem ser pré-processados e preparados de forma que os algoritmos de aprendizagem posam reconhecê-los e o formato multidimensional, onde os dados são organizados como *features* ou atributos, é o mais comum deles. Assim, passa a ser muito relevante organizar esses atributos ao mesmo tempo que o processo de limpeza dos dados estiver ocorrendo, quando dados faltantes ou equivocados serão encontrados e tratados de forma adequada. Após essa etapa, o resultado deve ser um conjunto de dados bastante estruturado e coeso, que possa então ser aplicado na fase seguinte do processo de aprendizagem pelo algoritmo (AGGARWAL, 2015).

Essa fase é citada como fase de pré-processamento e deve ser iniciada tão logo a etapa de coleta de dados tenha finalizado. Essa etapa pode ser subdividida em outras sub-etapas e tem o poder de influenciar significativamente o desempenho de algoritmos de aprendizagem supervisionada em aprendizagem de máquina - *Machine Learning* (AGGARWAL, 2015; KOTSIANTIS et al., 2006).

Na sub-etapa de extração dos atributos, partindo-se do principio que uma grande quantidade de dados pode estar disponível, a experiência do analista de dados deve ser capaz de abstrair os atributos que realmente são importantes para a solução do problema e para que esses dados sejam transformados em um conjunto de dados que faça sentido para ser processado (AGGARWAL, 2015).

A representação de dados, em geral, utiliza uma quantidade grande de atributos, porém somente alguns deles realmente estão relacionados com o problema (KOTSIANTIS et al., 2006).

Segundo Aggarwal (2015), na etapa de limpeza dos dados, assim como uma grande quantidade de dados pode ser encontrada, os mesmos, frequentemente, podem conter amostras errôneas ou mesmo entradas faltantes. Lidar com dados faltantes, como afirma Kotsiantis et al. (2006), é algo que frequentemente é necessário nas etapas de preparação de dados, principalmente quando dados do mundo real estão sendo utilizados. O autor afirma ainda que é importante entender o motivo da ausência desses dados para então aplicar a técnica correta de tratamento. Sendo assim, deve ser levado em consideração os fatores abaixo:

- O dado foi perdido ou esquecido no processo de amostragem;
- Não existe para algumas instâncias durante a amostragem;

- No processo de construção do conjunto de dados, o desenvolvedor não se importou com as amostras de um certo atributo.

Baseado nos fatores descritos, Kotsiantis et al. (2006) referencia várias técnicas que podem ser usadas para tratamento:

- Método de ignorar as entradas que não possuam um valor para o atributo, onde a entrada é simplesmente ignorada;
- Valor mais comum do atributo, método no qual o valor encontrado com mais frequência é usado para substituir um valor faltante;
- Substituição por média, quando a média de todos os valores é usada;
- Regressão, quando um método estatístico é usado para definir o valor a ser atribuído;
- Atribuição a quente, onde é identificado um caso similar que possua um valor conhecido e então esse valor é atribuído ao dado faltante;
- Valor especial, considerando que o dado faltante é parte do atributo então o mesmo permanece como tal.

Apesar de existirem várias técnicas para tratar os dados ausentes no conjunto de dados, frequentemente os mesmos são simplesmente eliminados, mesmo existindo estudos que afirmam que excluir dados faltantes do conjunto de dados pode levar a uma grande quantidade de informação perdida e além disso, atribuir algum viés aos que sobrarem (HUANG et al., 2015).

No caso de se encontrar dados de muitas dimensões, algum método de seleção desses atributos deve ser aplicado para evitar erros no processamento e também de forma a eliminar o que for ruído e que possa causar redução no desempenho ou na qualidade dos resultados. O resultado dessa sub-etapa deve ser transformar os dados crus em dados relevantes para análise. Essa etapa não é considerada pré-processamento pois nela está embutida a necessidade prévia de conhecimento do problema em questão e, muitas vezes, esse conhecimento só está disponível numa etapa posterior ao pré-processamento dos dados. De todo modo, essa etapa de seleção dos atributos deve ser realizada antes de aplicar os dados ao algoritmo/modelo do processo de aprendizagem (AGGARWAL, 2015).

Eliminar atributos que sejam irrelevantes ou redundantes causa redução da dimensionalidade dos dados e pode trazer benefícios aos algoritmos de aprendizagem, que

podem executar mais rapidamente e apresentarem desempenho melhor (KOTSIANTIS et al., 2006; HUANG et al., 2015).

2.5 *MACHINE LEARNING*

O sonho de que máquinas poderiam ser capazes de aprender é tão antigo quanto a própria ciência da computação e quem sabe até mais antiga que ela. Por muitos anos, esse conceito de que máquinas poderiam ser treinadas para aprender, foi muitas vezes ignorado, não recebeu atenção da comunidade acadêmica, não recebeu aportes financeiros de agências de financiamento à pesquisa e também de grandes companhias de desenvolvimento de software. A fraqueza dos sistemas baseados em conhecimento, desenvolvimentos principalmente na década de 1970, era justamente saber de onde viria o conhecimento que permitiria às máquinas terem a capacidade de aprender. Muitos pesquisadores imaginavam que esse conhecimento deveria vir de pessoas, que através de suas experiências, deveriam instruir as máquinas a realizar tarefas baseadas no seu próprio conhecimento sobre determinado assunto, porém, essa técnica se provara muito complicada pois muitas vezes era difícil traduzir o conhecimento de pessoa para pessoa e mais ainda, criar uma base de dados que fosse grande e coeza o suficiente para tal. Assim, o conhecimento de aprendizagem surgiu como a mudança necessária para fazer da aprendizagem de máquina um dos temas mais atuais nas áreas de computação (KUBAT, 2017).

O desejo de utilizar a vasta quantidade de dados disponível atualmente aumenta conforme o acesso a essas informações torna-se mais fácil e simples, principalmente através da internet e de outras mídias existentes. Ao mesmo tempo que o cérebro humano é capaz de processar paralelamente uma grande quantidade de informações, ele é limitado ao processar essa grande quantidade de dados que facilmente pode ser obtido e que muitas vezes está relacionada com problemas que precisam ser resolvidos por humanos (KUHN; JOHNSON, 2013).

Skiena (2017) afirma que atualmente, dentro da ciência da computação, *Machine Learning* é o tópico de estudo mais interessante, tanto pela descoberta de novos e poderosos algoritmos quanto pela grande possibilidade de aplicações dessas novas técnicas. Segundo o mesmo autor, esse interesse está associado principalmente a grande quantidade de dados disponíveis e ao aumento do poder computacional que permite, mesmo utilizando abordagens mais antigas, inspirar o desenvolvimento de métodos que se adéquam melhor e que são capazes de atrair investimentos em recursos de dados e desenvolvimento de novos sistemas. Além disso, ele cita que a facilidade com que novas ideias se transformam em ferramentas disponíveis é incrivelmente rápida hoje em dia, principalmente pela cultura do software de código aberto.

Ao utilizar ferramentas computacionais de aprendizagem para resolver problemas, estamos levando em consideração não somente o aspecto técnico-computacional diretamente envolvido mas, também, toda a teoria relacionada ao problema em si, que pode ser um problema relacionada a química, a física, a medicina, entre outros. Essa técnica, capaz de associar ferramentas computacionais com temas do cotidiano, das ciências e da estatística, é chamado de aprendizagem de máquina ou, como será tratado até o final desse trabalho, *Machine Learning*. Outros nomes podem estar associados ao mesmo conceito e é comum encontrar referencias a *data mining*, *predictive analysis*, *knowledge discovery*, entre outros (KUHN; JOHNSON, 2013).

O conceito de *Machine Learning* envolve softwares capazes de automaticamente ajustar a performance do seus resultados baseado na quantidade e qualidade dos dados aos quais ele está exposto. A melhoria na performance ou a melhoria do aprendizado, está relacionado diretamente com a otimização da parametrização dos seus parâmetros ajustáveis e, essa otimização, deve ser realizada conforme diferentes critérios de desempenho relacionados a técnica ou ao problema que se pretende resolver (IGUAL; SEGUÍ, 2017).

Igual e Seguí (2017) classifica *Machine Learning* como um sub-campo da inteligência artificial, que pode ser dividido em três classes principais:

- Aprendizagem por reforço;
- Aprendizagem não supervisionada;
- Aprendizagem supervisionada.

A primeira das classes diz respeito a aprendizagem por reforço. Nessa modalidade de aprendizagem, os algoritmos se baseiam na qualidade da solução e não em como melhorá-las (IGUAL; SEGUÍ, 2017). Skiena (2017) afirma que, em alguns problemas, a amostragem dos dados está relacionada com observações em interações com o mundo ou, algumas vezes, resultado de simulações do mesmo e a esse conceito de aprendizagem a partir do ambiente, ele classifica como aprendizagem por reforço.

Quando se fala em aprendizagem não supervisionada, basicamente se relacionam algoritmos que possuem capacidade de aprender baseado em dados não rotulados. O processo de aprendizagem consiste em explorar os dados de acordo com algum critério de similaridade, estatística ou mesmo geométrico (IGUAL; SEGUÍ, 2017). O autor cita como técnicas desse tipo de aprendizagem a *k-means clustering* e a *kernel density estimation*, por exemplo.

Igual e Seguí (2017) classifica aprendizagem supervisionada como a capacidade de

aprender baseado em dados rotulados, isso significa dizer que nesse tipo de aprendizagem, são conhecidas tantos as entradas como as saídas, permitindo ao algoritmo se aproximar cada vez mais da saída desejada, através da otimização dos seus pesos conforme a rotina de treinamento vai evoluindo. Outra abordagem é dada por Skiena (2017) onde o autor cita que a aprendizagem supervisionada consiste em apresentar ao algoritmo, vetores de entrada X_i, onde cada entrada está relacionada com uma saída Y_i, que representa a supervisão do treinamento, tipicamente derivada de um processo de amostragem ou registro de dados. Algumas técnicas conhecidas, que abordam o aprendizado supervisionado, são as *Neural Networks*, *Support or Regression Vector Machines*, *Decision Trees*, *Boosting*, *Random Forests*, entre outras menos conhecidas. Por fim, Hastie et al. (2009) cita que a aprendizagem é supervisionada devido a presença de uma variável de saída, que guia todo o processo de aprendizagem.

Uma definição matemática é dada por Hastie et al. (2009) onde o autor simplifica o conceito supondo que o erro é aditivo e que um modelo representado por $Y = f(X) + \varepsilon$ é uma definição razoável para exemplificar que algoritmos de aprendizagem supervisionada trabalham no sentido de aprender a função f, observando tanto X quanto Y, definindo um conjunto de treinamento $T = (x_i, y_i), i = 1, ..., N$. Assim, o algoritmo irá produzir saídas $f(x_i)$ em resposta as entradas de 1 a N e modificar a relação entre as entradas e saídas em resposta as diferentes $y_i - f(x_i)$ entre as saídas originais e as saídas geradas pela execução. Até o completo processo de aprendizagem, as saídas reais e as saídas geradas pela execução do algoritmo serão próximas o suficiente para garantir generalização a outras entradas encontradas na prática.

Os dados utilizados pelos algoritmos de aprendizagem supervisionada são divididos em três tipos: o primeiro deles são os dados de treinamento, que correspondem aos dados que fornecem as informações com as quais o algoritmo aprende. O segundo, chamados de dados de avaliação, correspondem aos dados usados para ajustar o processo de treinamento, visto que essa etapa do processo tem seu desempenho medido através da avaliação de quão bem os valores previstos correspondem aos valores dos dados reais e, o terceiro, são chamados de dados de teste, que são usados para obter uma avaliação final do desempenho do procedimento de aprendizagem (BERK, 2008).

Ertel (2017) traz um conceito semelhante. Para o autor, dados de treinamento são os dados que contém o conhecimento o qual o algoritmo de aprendizagem deve extrair e então aprender. Sobre essa classe de dados, ele afirma que a escolha dos mesmos deve garantir uma amostra representativa sobre a tarefa a ser aprendida. Dados de testes são classificados como sendo aqueles usados para verificar se o modelo treinado é capaz de generalizar o resultado para novos dados apresentados e por último, a medição de desempenho, os dados usados para medir o

resultado final do modelo de aprendizagem, considerando ainda que o resultado do desempenho é muito mais fácil de entender se comparado a função a qual o modelo está tentando representar através do processo de aprendizagem, funcionando como uma medida de qualidade.

Ainda sobre a classificação dos dados no processo de aprendizagem supervisionada, Igual e Seguí (2017) introduz o conceito de hiper parametrização que, segundo o autor, é chamado de validação. Dentro desse processo, torna-se necessário dividir os dados nas três classes citadas nos dois últimos parágrafos e, além disso, garantir que os dados de teste sejam usados exclusivamente para o teste de desempenho e nunca no processo de aprendizagem, como dado de treinamento. Ainda segundo o autor, a validação deve ser usada para selecionar os parâmetros e/ou modelos com o melhor desempenho, geralmente baseados na generalização do erro entre o resultado real e o resultado encontrado.

2.6 DIVISÃO DO CONJUNTO DE DADOS

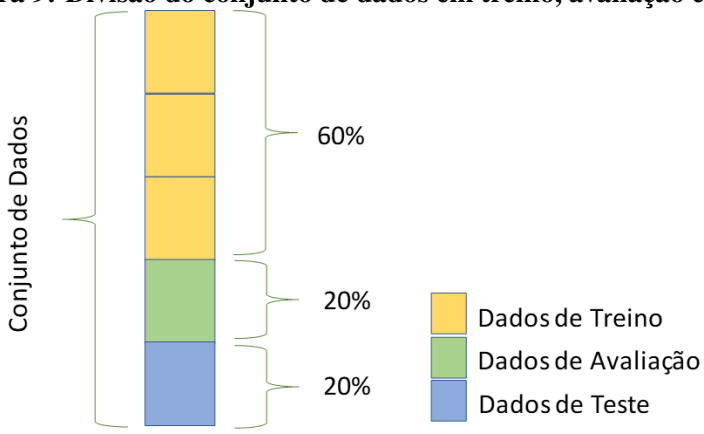
Como citado anteriormente, no que tange a divisão do conjunto de dados, é comum a opinião de que três subconjuntos devem ser criados, representando os dados de treino, avaliação e teste porém, são limitados os autores que afirmam existir um tamanho mínimo para esses subconjuntos.

Segundo Berk (2008) o tamanho desses subconjuntos depende da configuração, porém um mínimo de 500 linhas em cada um deles deve ser suficiente para um processo de aprendizagem efetivo. Nessa linha de raciocínio, um conjunto de dados de 1500 linhas pode ser aleatoriamente dividido entre os três subconjuntos, ficando cada um deles com 500 amostras.

Não está exatamente claro o quão grande o tamanho de um conjunto de dados deve ser, até que ele esteja próximo o suficiente desse tamanho e portanto, essa abordagem passa a ser justificada assintoticamente (uma proximidade cada vez maior, porém sem realmente chegar a esse limite). Por exemplo, se algumas variáveis estiverem muito concentradas dentro de um determinado limite, eventualmente alguns dados das extremidades podem ter sido deixados de fora do conjunto de dados e consequentemente, dos subconjuntos divididos a partir dela (BERK, 2008).

Na figura 9 é representado o processo de divisão do conjunto de dados em 3 subconjuntos. A escolha da quantidade de amostras em cada subconjunto é arbitrária, porém está relacionada diretamente com o resultado do processo de aprendizagem.

Figura 9: Divisão do conjunto de dados em treino, avaliação e teste



Fonte: Autoria Própria

No processo de aprendizagem supervisionada de um algoritmo utilizando um conjunto de dados, é importante se preocupar com a melhor forma de executar essa etapa do treinamento. A fase de treino deve ser preparada levando em conta dois aspectos, o primeiro relacionado à necessidade de conseguir bons resultados sem super-treinar o algoritmo, quando se tem disponível uma grande quantidade de dados para esse fim. O Segundo, ao contrário, quando se possui uma pequena quantidade de dados, onde o resultado pode eventualmente estar abaixo da expectativa, já que o algoritmo não será treinado corretamente (YADAV; SHUKLA, 2016).

Em aplicações reais, em geral, somente uma quantidade pequena de dados está disponível o que leva a ideia de dividir esses dados. Data dos anos 30 o conceito de que treinar um algoritmo e usar os mesmos dados para validar estatisticamente seu desempenho, leva a um resultado superestimado. A validação cruzada surge exatamente nesse contexto, onde validar um algoritmo com novos dados torna-se uma boa estimativa de seu desempenho e a amostra de avaliação, derivada do conjunto de dados principal, faz o papel dos novos dados sendo apresentados ao algoritmo e por consequência, traduzem melhor os resultados encontrados (ARLOT et al., 2010).

Berk (2008) afirma que quando o conjunto de dados disponível possui muito poucas amostras e somente os dados de treinamento estão disponíveis, existem algumas técnicas que podem ser usadas para aproximar os resultados do processo de aprendizagem aos resultados que poderiam ser encontrados caso o conjunto de dados fosse maior o suficiente para a divisão. Dentre as técnicas disponíveis, a validação cruzada é talvez a mais comum.

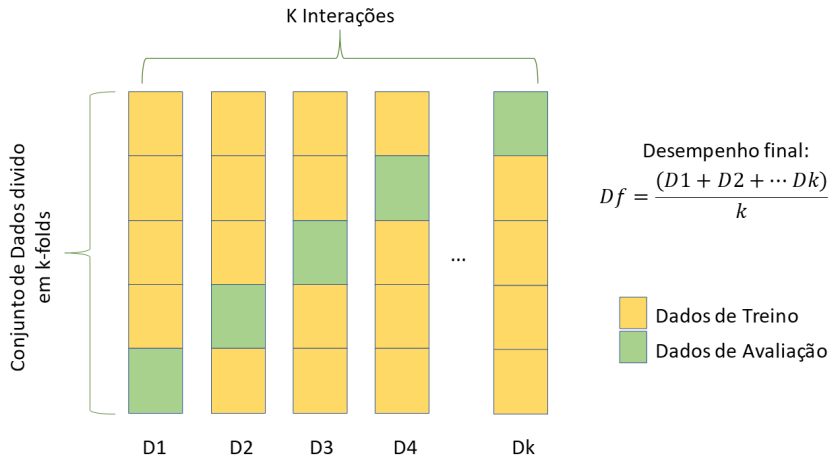
Já Reitermanova (2010) afirma que uma das principais premissas em *Machine Learning* é construir modelos computacionais com grande habilidade de generalização e que,

frequentemente, uma generalização pobre está relacionada com super-treinamento. A autora afirma que a validação cruzada é comumente usada para evitar esse problema.

Em problemas com um conjunto de dados pequeno, a escolha ideal passa a ser o método *K-fold* de validação cruzada, com valores grandes de K (desde que K seja menor que o numero de amostras no conjunto de dados) (YADAV; SHUKLA, 2016).

Considerando um conjunto de dados com N amostras, por exemplo, somente os dados de treino estariam disponíveis, não existindo amostras suficientes para os subconjuntos de dados de avaliação e teste. Nesse caso, utilizando a técnica *K-fold* de validação cruzada, o conjunto de dados com N amostras pode ser dividido em K subconjuntos com N/K amostras aleatoriamente escolhidas e então, alternadamente, K-1 desses subconjuntos seriam usados como dados de treino e o subconjunto que resta como dado de avaliação, a cada interação. Sendo assim, após K interações (já que o conjunto de dados foi dividido em K subconjuntos), todas os subconjuntos serão, em alguma interação, usados como dados de avaliação e nas outras interações, uma combinação de dados de treino. Ao final do processo, uma média dos resultados da avaliação de cada interação pode ser calculada e o valor encontrado passa a ser considerado como a pontuação de desempenho nessa etapa do processo de aprendizagem. A figura 10 traz a representação do processo de validação cruzada usando o *k-fold* (BERK, 2008; BLUM et al., 1999; YADAV; SHUKLA, 2016).

Figura 10: Representação da validação cruzada com *k-fold*



Fonte: Autoria Própria

A técnica de validação cruzada pode parecer bastante simples, mas ela depende de dois fatores importantes: como os dados foram gerados e de qual procedimento estatístico foi aplicado para se obter o conjunto de dados. Outro ponto de atenção está na parametrização do procedimento, que pode levar ou não a uma melhor performance. No exemplo acima, o

uso de K subconjuntos pode ter sido definido arbitrariamente. Um grande numero de divisões significa que o subconjunto com os dados de treino se tornará cada vez maior em relação ao subconjunto com os dados de avaliação e isso, em geral, tem consequências positivas no resultado do processo de aprendizagem porém pode fazer com que a performance da execução se torne um problema, computacionalmente (BERK, 2008).

Um complemento a essa abordagem, encontrado no trabalho de (SHAO, 1993), é de que no processo de validação cruzada, a avaliação do modelo é usada não somente com o os K-1 dados e sim com todos os N dados disponíveis no conjunto de treinamento, já que existem S diferentes formas de dividi-lo, com a consideração de que a complexidade computacional vai aumentar conforme K aumente.

Yadav e Shukla (2016) afirma que o fator mais crítico na aplicação do método K-fold de validação cruzada está justamente na escolha dos seus hiper parâmetros. Ele afirma ainda que a escolha torna-se particularmente sensível a partir do momento em que um pequeno erro de parametrização pode levar ao super ou ao sub-treinamento do modelo.

Sendo assim, não existe formalmente algo que justifique a escolha de um determinado numero de divisões porém, na prática comum, é normal trabalhar com 5 ou 10 subconjuntos, lembrando que o principal problema desse procedimento não está relacionado com a quantidade de subconjuntos e sim com o fato de que somente dados de treino estão sendo originalmente utilizados (BERK, 2008).

A divisão do conjunto de dados entre os três subconjuntos e o procedimento de validação cruzada, são assuntos diferentes e por consequência produzem, da mesma forma, desafios também diferentes. Em resumo, a divisão do conjunto de dados em três subconjuntos permite uma avaliação de desempenho mais fiel quando comparada com a validação cruzada. Outro ponto a ser considerado é que a divisão em subconjuntos fornecem uma ampla variedade de alocações de dados diferentes (já que é possível variar as amostras dentro de casa subconjunto em diferentes execuções do processo de treinamento). Por fim, algumas vezes, a escolha entre um e outro é determinada puramente pela aplicação (BERK, 2008). Dentro do mesmo contexto Arlot et al. (2010) afirma ainda que dados de teste do mesmo tamanho possuem o mesmo viés no processo de validação cruzada porém, mesmo assim, se comportam de forma diferente no processo de treinamento, o que pode ser explicado pela diferença de variança nos dados.

2.7 RANDOM FOREST

Breiman (2001) originalmente definiu *Random Forest* como sendo um classificador, formado por uma coleção de árvores estruturadas de decisão $\{h(x,\theta k), k = 1,...\}, onde\{\theta k\}$ é um vetor aleatório independente distribuído identicamente e que cada árvore representa um voto unitário para a classe mais popular referente a entrada x. De maneira mais usual, Genuer et al. (2010) pondera que *Random Forest* é um algoritmo muito eficiente e popular, baseado na ideia de *bagging* (do original (BREIMAN, 1996)), que pode ser usado para resolver problemas de classificação e regressão, pertencendo a família dos métodos *ensemble*, onde múltiplos classificadores ou regressores são associados e seus resultados são agregados utilizando a média (regressão) ou o mais votado dentre todos (classificação) (LIAW et al., 2002; BIAU; SCORNET, 2016).

Usando uma outra abordagem de definição, *Random Forest* pode ser definido como um *ensemble* desenvolvido para aumentar o desempenho do método CART *(Classification and Regression Trees*, Breiman et al. (1984) através da combinação de um grande número de *Decision Trees*, onde a construção de cada árvore é determinada por uma algoritmo determinístico e o mesmo seleciona aleatoriamente um conjunto de variáveis de entrada e também uma amostra do conjunto de dados para sua execução (MUTANGA et al., 2012; WANG et al., 2018; PRASAD et al., 2006). Nesse sentido, Couronné et al. (2018) sugere que *Random Forest* é uma técnica que tem como resultado uma redução grande de variância quando comparado a uma simples árvore de decisão, já que ele resulta de um agregado de um grande número de árvores.

Pouco viés e pouca variância são as propriedades mais desejadas de qualquer modelo de predição. Durante o processo de crescimento de suas árvores, o método do *Random Forest* é capaz de atingir os objetivos acima explorando a combinação da aleatoriedade derivada do processo de agregação (que permite preditores com pequeno viés) e da aleatoriedade na construção de suas árvores (que permite redução da variância quando os resultados das mesmas tem suas médias calculadas) (TORRES-BARRÁN et al., 2019).

Prasad et al. (2006) e Tohry et al. (2019) citam algumas vantagens do método *Random Forest*:

- Pequeno número de parâmetros a serem sintonizados;
- Baixo viés devido a seleção aleatória de preditores, que diminui a correlação entre as árvores criadas;

- Impossibilidade de *overfitting*;
- Tratativa automática para dados faltantes.

Random Forest demonstra excelente desempenho em problemas com conjuntos de dados onde a quantidade de entradas é maior que a quantidade de amostras e em problemas de larga escala, além de ser muito útil medindo a importância das variáveis de entrada na construção do modelo, o que permite reduzir o conjunto de dados e melhorar o desempenho e o tempo de execução do processo de aprendizagem. A avaliação de importância das variáveis de entrada é feita através de permutação, onde cada entrada é permutada aleatoriamente e individualmente e o desempenho do modelo é calculado e então comparado com os outros resultados encontrados através da permutação das outras variáveis de entrada (PRASAD et al., 2006; BIAU; SCORNET, 2016).

A técnica por trás do *Random Forest* está associada ao conceito de dividir para conquistar, onde frações do conjunto de dados desencadeiam o crescimento aleatório de inúmeras arvores de decisão e ao final os resultados de cada árvore são agregados. O algoritmo é ainda conhecido como um dos métodos mais bem-sucedidos disponíveis atualmente, para lidar com dados dessa forma (BIAU; SCORNET, 2016).

Por ser aplicável a uma grande variedade de problemas relacionados a predição e ao fato de ter poucos parâmetros para sintonizar, além de ser simples de usar, tornaram essa técnica bastante popular atualmente (BIAU; SCORNET, 2016; TORRES-BARRÁN et al., 2019), apesar de muitas vezes ser classificado como uma caixa-preta, já que muito pouco pode ser analisado individualmente de cada árvore criada (PRASAD et al., 2006; GENUER et al., 2010; COURONNÉ et al., 2018).

Alguns exemplos onde é de conhecimento o bom resultado de aplicação do *Random Forest* em problemas práticos:

- Ecologia (MUTANGA et al., 2012; PRASAD et al., 2006) ;
- Reconhecimento de face em 3-Dimensões (FANELLI et al., 2013);
- Bioinformática (BOULESTEIX et al., 2012);
- Predições para geração de energia eólica e solar (TORRES-BARRÁN et al., 2019);
- Otimização de consumo de energia (SMARRA et al., 2018);
- Predição no consumo de energia (TOHRY et al., 2019);

- Astrofísica (GAO et al., 2009).

De forma concisa, Biau e Scornet (2016) introduz matematicamente o conceito de *Random Forest* aplicado a problemas de regressão. O autor faz referência a uma estimativa de regressão não paramétrica, onde um vetor aleatório $X \in X \subset R^p$ é monitorado e o objetivo é prever o quadrado da resposta randômica integrável $Y \in R$ através da estimativa da função de regressão:

$$m(x) = E[Y|X = x] \tag{2}$$

E de um conjunto de dados de treinamento de variáveis independentes distribuídas randomicamente como o par prototipado (X,Y):

$$2D_n = ((X_1, Y_1), ..., (Xn, Yn)) \tag{3}$$

O objetivo é construir uma estimativa $m_n : X \to R$ da função m através do uso do conjunto de dados D_n, onde a função de regressão m_n, através do erro médio quadrado, consiste em $E[m_n(X)$ - $m(X)]^2 \to 0$,enquanto $n \to \infty$.

Sendo assim, (BIAU; SCORNET, 2016) cita ainda que o *Random Forest* é um preditor baseado em uma coleção de M árvores de regressão randomizadas onde, para a j-ésima árvore da floresta, o resultado da predição de uma entrada x é definida por $m_n(x; \Theta j, D_n)$ e $\Theta 1, ..., \Theta M$ são variáveis aleatórias independentes, distribuídas de forma semelhante a variável genérica aleatória Θ e independentes do conjunto de dados D_n.

Em outras palavras, previamente a fase de crescimento de cada árvore individual e também da direção das divisões em cada decisão, a variável Θ é usada para uma nova amostragem do conjunto de dados de treinamento e de forma matemática, a j-ésima árvore da floresta assume a forma:

$$m_n(x; \Theta j, D_n) = \sum_{i \in D_n^*(\Theta j)} \frac{\mathbb{1}_{X_i \in A_n(x; \Theta j, D_n)} Y_i}{N_n(x; \Theta j, D_n)}$$

onde $D_n^*(\Theta j)$ é o conjunto de dados selecionado antes da construção da árvore através do uso da variável Θ, $A_n(x; \Theta j, D_n)$ é a célula contendo a entrada x e $N_n(x; \Theta j, D_n)$ é o número de pontos que se relacionam com $A_n(x; \Theta j, D_n)$ (BIAU; SCORNET, 2016).

Ainda segundo Biau e Scornet (2016) a combinação das diversas árvores pode ser representada de forma finita:

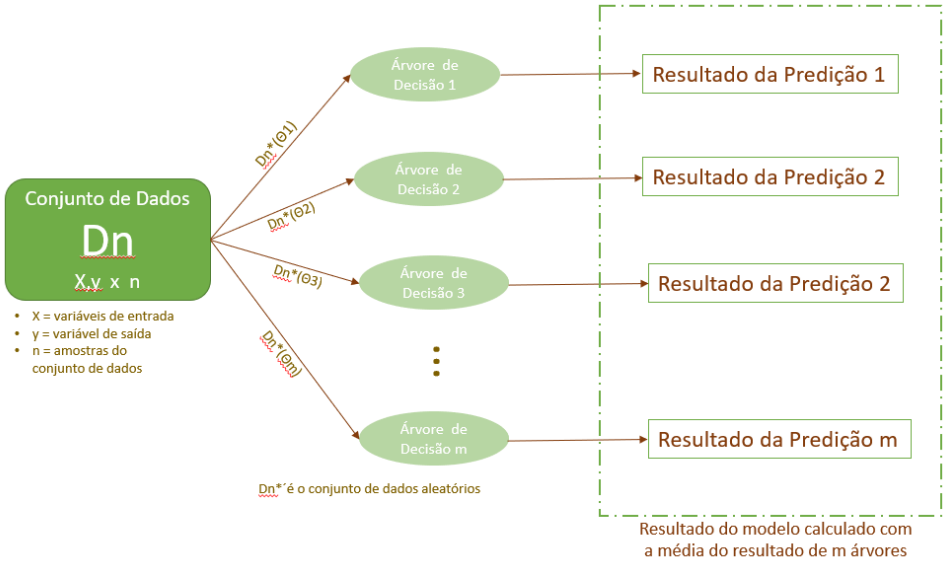
$$m_{M,n}(x;\theta 1,....,\theta M,D_n) = \frac{1}{M}\sum_{j=1}^{M} m_n(x;\theta j,D_n)$$

ou infinita, sendo a segunda forma mais usual já que a quantidade de árvores escolhidas na parametrização do método pode ser escolhida arbitrariamente e será limitada apenas pela disponibilidade computacional:

$$m_{\infty,n}(x;D_n) = \mathbb{E}_{\Theta}[m_n(x;\Theta,D_n)]$$

O algoritmo do *Random Forest* funciona criando M diferentes árvores de forma aleatória. Previamente a construção de cada árvore, uma fração do conjunto de dados de treinamento é selecionada aleatoriamente e, a partir desse ponto, somente esse conjunto é usado para a construção das árvores. Desse modo, em cada uma dos nós de cada uma das árvores uma divisão é feita maximizada pelo critério CART em diferentes direções limitadas pelo parâmetro *mtry* escolhidas também aleatoriamente entre os P originalmente disponíveis. Nesse contexto, a raiz de cada árvore é o valor da entrada X e a cada passo de construção da árvore um nó da mesma é dividido em duas partes, sendo os nós terminais (ou nós folhas), quando associados, uma parte de X (BIAU; SCORNET, 2016). A figura 11 representa graficamente o conceito de execução do algoritmo do método *Random Forest*.

Figura 11: Representação do método *Random Forest* para modelos de regressão



Fonte: Autoria Própria

Finalmente, o processo de construção de cada árvore individualmente é interrompido quando cada célula contém menos que *nodesize* pontos. Para cada variável $x \in X$, cada uma das

árvores de regressão prediz a média de Y_i para qual o correspondente X_i está relacionado com a célula de x (BIAU; SCORNET, 2016).

O algoritmo que representa o método *Random Forest* possui, segundo Biau e Scornet (2016), Mutanga et al. (2012) e Couronné et al. (2018) alguns parâmetros importantes (porém não limitados a esses) que precisam ser otimizados para aumentar o desempenho do método (o nome dos parâmetros está baseado no padrão usado na linguagem R):

- $a_n \in 1, ..., n$, que representa o número de amostras do conjunto de dados de treinamento em cada árvore;
- $mtry \in 1, ..., p$, que representa a quantidade de variáveis de entrada que serão aleatoriamente selecionadas a cada divisão;
- $nodesize \in 1, ..., a_n$, que representa o número limite de divisões de cada célula e que está relacionado com o menor tamanho dos nós terminais (nós folhas);
- *ntree*: Número total de árvores que será criado durante a execução do processo de aprendizagem.

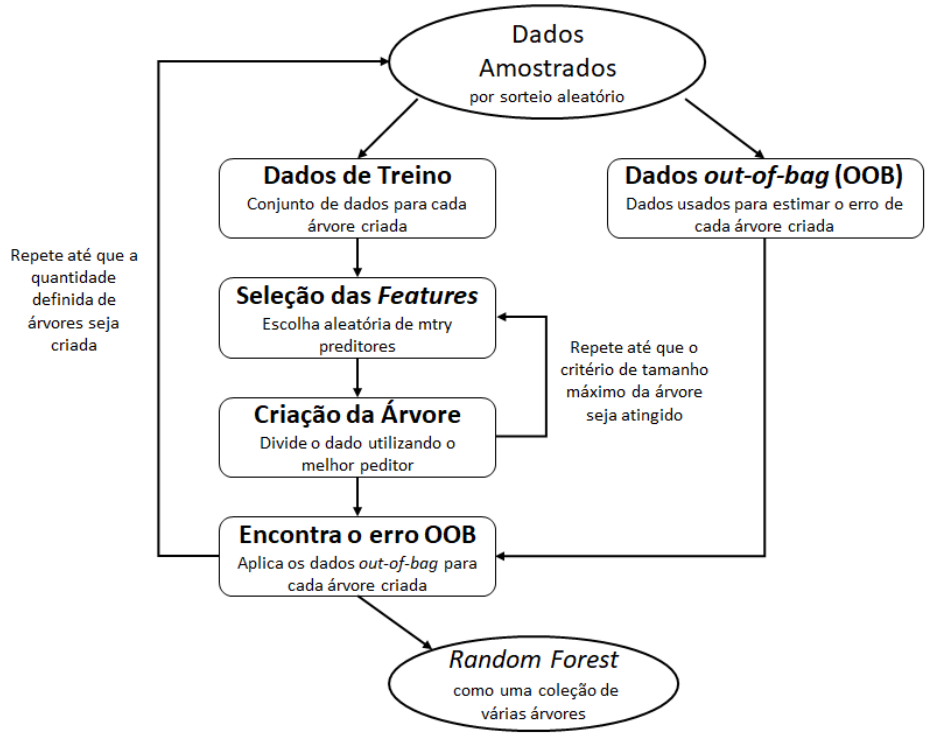
A figura 12 representa uma adaptação do algoritmo proposto por Biau e Scornet (2016) onde os parâmetros fazem referência a linguagem R, na qual o algoritmo foi originalmente escrito. Já a figura 13 representa o fluxograma executado pelo mesmo algoritmo.

Figura 12: Algoritmo *Random Forest*

Entrada: Conjunto de Treinamento D_n, numero de árvores $M > 0$, $a_n \in \{1,.....,n\}$, *mtry* $\in \{1,....., p\}$, *nodesize* $\in \{1,....., a_n\}$, and $x \in X$.
Saída: Predição do Random Forest em x.
Para j = 1, ..., M **faça**
  Selecione a_n pontos, com (ou sem) substituição, uniformemente em D_n.
    Nos passos seguintes, somente as na observações são usadas.
  Atribua P = (X) a lista contendo o nó associado com a raíz da árvore
  Atribua $P_{final} = \emptyset$ uma lista vazia.
  Enquanto $P \neq \emptyset$ **faça**
    Atribua A como o primeiro elemento de P
    Se A contém menos que *nodesize* pontos ou se todos $\mathbf{X}_i \in A$ são iguais **então**
      Remova o nó A da lista P
      P_{final} = Concatenação (P_{final}, A).
    Senão
      Selecione uniformemente, sem substituição, um subconjunto $M_{try} \subset \{1,.....,p\}$ com cardinalidade *mtry*
      Selecione a melhor divisão em A otimizando o critério CART ao longo das coordenadas em M_{try}
      Divida o nó A de acordo com a melhor divisão. Chame de A_L e A_R os dois nós resultantes
      Remova o nó A da lista P
      P = Concatenação (P, A_L, A_R).
    Fim
  Fim
  Calcule o valor predito $m_n(x; \Theta_j, D_n)$ em x igual a média de Y_i atribuido ao nó de x na partição P_{final}
Fim
Calcule a estimativa *Random Forest* ${}^{m}M_{,n}(x; \Theta_1,....., \Theta_M, D_n)$ no ponto x em questão

Fonte: Adaptado de (BIAU; SCORNET, 2016)

Figura 13: Fluxograma *Random Forest*



Fonte: Adaptado de (BOULESTEIX et al., 2012)

Um recurso importante do *Random Forest* está associado com as amostras que não são utilizadas no processo de construção das árvores, também chamadas de amostras OOB *(out-of-bag)*. Essas amostras podem ser utilizadas como um conjunto de testes, já que não foram previamente utilizadas pelo algoritmo devido ao processo de seleção aleatório de um conjunto de treino dentro do conjunto de dados, para crescimento das árvores (PRASAD et al., 2006).

Em aplicações de *Random Forest*, em média, 36% do conjunto de dados pode ser classificado como *out-of-bag* o que permite o cálculo do erro com uma amostra bastante considerável em relação ao total disponível. Nesse caso, desde que exista uma quantidade suficiente de árvores disponíveis para obtenção do modelo, essa estimativa do erro é bastante precisa e, ao contrário da validação cruzada, desprovida de viés (BREIMAN et al., 1984; LIAW et al., 2002).

Assim, essas amostras permitem o cálculo da taxa de erro sem viés durante a etapa de validação do treinamento, além de eliminarem a necessidade de um conjunto de testes separado ou então da aplicação do método de validação cruzada. Outro ponto interessante é que devido à grande quantidade de árvores que podem ser usadas no modelo, o erro de generalização torna-se limitado, o que permite afirmar que o *overfitting* do modelo é impossível (PRASAD et al., 2006).

Um outro recurso também disponível ao utilizar o *Random Forest* é a classificação de variáveis importantes. Segundo (LIAW et al., 2002) o *Random Forest* faz essa estimativa usando o resultado do cálculo do erro das amostras *out-of-bag*, comparando o quanto ele aumenta quando cada entrada do conjunto de dados X é permutada, enquanto as outras permanecem inalteradas. Quanto maior o resultado do erro, maior a influência dessa variável na obtenção do modelo e consequentemente mais importante ele é. O resultado do cálculo de importância da variável é dado pela equação abaixo:

$$MSE_{OOB} = n^{-1} \sum_{1}^{n} \{y_i - y_i^{OOB}\}^2$$

Onde Y_i^{OOB} é a média das predições das amostras que estão *out-of-box* para a i-ésima amostra.

2.8 *PARTICLE SWARM OPTIMIZATION*

Vários métodos de otimização que podem ser encontrados na literatura, são capazes de resolver problemas sob diferentes condições. Os métodos empregados são classificados quanto ao seu espaço de busca, sua função objetivo, se são contínuos ou discretos, restritos

ou irrestritos, mono ou multiobjetivo, estáticos ou dinâmicos. Dentre eles, podemos citar programação linear e não linear, programação quadrática e também programação dinâmica. Esses métodos, apesar de comprovadamente serem úteis na solução de vários problemas, possuem restrições de utilização. Em referência a programação linear, por exemplo, o problema precisa apresentar linearidade entre a função objetivo e seus elementos. Já a programação não-linear, que surge para resolver essa restrição, esbarra na dificuldade de implementação enquanto a programação dinâmica, que mesmo tendo sido matematicamente comprovada para uso em otimização, esbarra na inviabilidade de resolução de seus algoritmos. Além disso, por serem todas soluções numéricas, necessitam de grande esforço computacional, que cresce exponencialmente com o aumento do tamanho do problema, o que leva a uma solução sub-ótima muitas vezes. Técnicas de inteligência computacional, como o *Particle Swarm Optimization* (Otimização por Enxame de Partículas), podem ser a solução para esses problemas. Nesse trabalho, a técnica será referenciada apenas como PSO, acrônimo do nome original em inglês (VALLE et al., 2008; BOUSSAÏD et al., 2013).

Segundo BoussaïD et al. (2013), o PSO está numa classe de algoritmos conhecido como metaheurística. Uma metaheurística é um algoritmo capaz de resolver uma grande quantidade de problemas difíceis de otimização, sem necessariamente ser adaptado a cada problema. Por isso, é também conhecido como uma heurística de alto-nível em contraste com heurísticas mais específicas para solução de problemas. As metaheurísticas geralmente são aplicadas em soluções de problemas que algoritmos específicos não podem resolver satisfatoriamente.

Em sistemas computacionais com inteligência de enxames, cada agente computacional possui a capacidade de perceber e modificar o seu próprio ambiente de maneira local e de se comunicar com outros agentes dentro do mesmo sistema, permitindo que as mudanças no ambiente geradas por eles também sejam percebidas. Esses sistemas são auto-organizados, distribuídos, autônomos, flexíveis e dinâmicos e, embora não exista uma estrutura de controle central orientando o comportamento de cada um dos agentes existentes, as interações locais entre seus agentes levam a um comportamento global, que tende a se aproximar da solução do problema (SERAPIÃO, 2009)

As principais propriedades de um sistema de inteligência por enxame são (SERAPIÃO, 2009):

- Proximidade, pois os agentes envolvidos devem ser capazes de interagir entre eles;
- Qualidade, pois os agentes devem ter a capacidade de avaliar seu próprio comportamento;

- Diversidade, pois isso permite ao sistema reagir a qualquer situação inesperada;
- Estabilidade, pois o comportamento de um agente não pode ser afetado por qualquer variação ambiental e;
- Adaptabilidade, pois partindo do princípio que algumas variações ambientais vão afetar o comportamento de um agente, então o mesmo deve ser capaz de se adequar a essas variações.

O PSO surgiu na década de 1990 através do trabalho de (KENNEDY; EBERHART, 1995) como um algoritmo para tratar de problemas de domínio contínuo. Essa técnica derivou de observações que modelavam o comportamento social de algumas espécies de animais como peixes e pássaros, porém, sem descartar o comportamento social humano. Nessa técnica, assim como em outras técnicas que envolvam inteligência por enxame, cada indivíduo tem sua própria experiência e é também capaz de avaliar como seus vizinhos se comportam, adotando uma característica individual e social. Assim, qualquer decisão que venha a ser tomada por um indivíduo, vai estar relacionada a sua experiência individual, baseada no seu desempenho no passado, e também a experiência de alguns vizinhos, baseado em seus próprios desempenhos. Esse comportamento sugere que o compartilhamento de informações entre indivíduos da mesma espécie, atuando em comunidade, oferece uma vantagem evolucionária (SERAPIÃO, 2009; BANKS et al., 2007)

Segundo Kennedy e Eberhart (1995) o PSO foi desenvolvido como um método para otimização de funções contínuas não-lineares e possui suas origens em *artificial life* ou *A-life*, quando associada ao comportamento de bandos de pássaros, cardumes de peixes ou inteligência por enxame, e na computação evolucionária, quando associado a algoritmos genéticos e programação evolucionária.

Outra abordagem é dada por Poli et al. (2007) que caracteriza o PSO como uma quantidade de partículas (ou indivíduos) distribuídos dentro do espaço de busca onde cada um deles avalia individualmente a função objetivo *(fitness function)* comparando com sua posição atual. A partir disso, cada partícula determina seu próximo movimento baseado em sua posição atual, sua melhor posição historicamente e também a melhor posição do enxame frente a função objetivo, adicionando também uma perturbação aleatória ao mesmo. Assim, a próxima iteração de cada partícula vai ocorrer após todas as outras terem finalizando seus movimentos e esses movimentos irão convergir para uma solução ótima. Uma partícula dentro do enxame não tem poder algum para resolver qualquer problema, portanto o PSO só se torna efetivo quando existe interação entre as partículas.

Adicionalmente, esse comportamento tende a evitar soluções ruins além de auxiliar no contexto de soluções difíceis já que a solução será dividida entre vários indivíduos ao invés de ficar concentrada em um único deles. Além disso, soluções sub-ótimas prematuras podem ser evitadas adicionando um componente aleatório ao algoritmo, o que faz com que indivíduos se movam de suas posições ótimas atuais em busca de novas potenciais posições ótimas (BANKS et al., 2007).

O PSO se mostra efetivo em implementações para solucionar vários tipos de problemas além da proposta original e envolve conceitos simples e matemática básica para sua codificação, além de consumir poucos recursos computacionais. Embora sua origem seja associada a otimização de problemas não-lineares, resultados promissores também estão associados a otimização combinatória onde, na maioria dos casos, as equações originais são mantidas (MARTINS et al., 2013; KENNEDY; EBERHART, 1995; BANKS et al., 2007).

No trabalho de (POLI, 2008) existem referências de implementação do PSO em áreas como projeto de antenas, biomedicina, redes de comunicação, otimização combinatória, controle, clusterização e classificação, projeto de circuitos eletrônicos, finanças, eletrônica e eletromagnetismo, motores e turbinas, imagem e vídeo, entretenimento, analise de falhas, metalurgia, robótica, modelagem, segurança militar, entre outros.

Ao comportamento individual e coletivo de cada indivíduo, podem ser atribuídas as características seguintes (SERAPIÃO, 2009):

- Avaliar, onde os indivíduos podem estimar seu próprio comportamento baseado na sensação que os mesmos possuem em relação ao ambiente;
- Comparar, onde cada indivíduo se compara a outros indivíduos e isso torna-se referência para avaliar suas experiências individuais;
- Imitar, onde cada indivíduo pode imitar outros indivíduos.

No PSO, cada indivíduo possui as características acima para definir o seu comportamento, baseados na possibilidade de interação entre si e com o ambiente. Baseado nessas características, cada indivíduo vai influenciar o comportamento global, que será o resultado das interações entre os indivíduos (SERAPIÃO, 2009)

Segundo Serapião (2009), a maioria dos algoritmos existentes de PSO emprega o conceito global e local através de dois tipos de informação que são importantes para o processo de decisão. O primeiro é o *global best* ou *gbest*, que influencia todos os membros de uma população entre si mesmos globalmente e o segundo é o *local best* ou *pbest*, que cria uma

vizinhança para cada indivíduo e que influência localmente cada um deles e seus vizinhos mais próximos. Ambos, *global* e *local best* são avaliados por uma função objetivo (também chamada de *fitness*), que corresponde a solução ótima do problema.

Matematicamente, para simplificar a abordagem, define-se o PSO em torno de uma única partícula p_i que irá se mover em função da sua posição atual $x_i(t)$, de uma velocidade $v_i(t+1)$, da posição com seu melhor desempenho local *pbest* até o momento e, também, a da posição com o melhor desempenho global *gbest* do sistema, até o momento (BONYADI; MICHALEWICZ, 2017; SERAPIÃO, 2009).

Computacionalmente, cada indivíduo pode ser representado como três vetores $\vec{x_i}$ (que representa a posição atual) $\vec{p_i}$ (que representa a melhor posição local) e $\vec{v_i}$ (que representa a velocidade) de dimensão D, onde D representa a dimensionalidade do espaço de busca. O vetor $\vec{x_i}$ contém as coordenadas que representam uma posição dentro do espaço de busca. Essa posição é avaliada a cada nova iteração do algoritmo e se a mesma for melhor que qualquer outra posição encontrada anteriormente por aquele indivíduo, então as suas coordenadas passam a ser armazenadas no vetor $\vec{p_i}$ e essa posição é armazenada também na variável *pbest*, que será usada para as próximas comparações. As novas posições serão calculadas adicionando coordenadas $\vec{v_i}$ ao vetor $\vec{x_i}$ e o algoritmo vai trabalhar ajustando $\vec{v_i}$ estocasticamente a cada nova iteração da mesma forma (POLI et al., 2007).

A velocidade $v_i(t+1)$ da partícula é dada pela equação abaixo (VALLE et al., 2008; POLI, 2008):

$$v_i(t+1) = v_i(t) + \Phi 1R1(pbest - x_i(t)) + \Phi 2R2(gbest - x_i(t))$$

com $\Phi 1$ e $\Phi 2$ representando constantes, limitadas a um espaço finito e R1 e R2 o componente aleatório do método.

Assim, com o cálculo da velocidade para a próxima interação e o conhecimento da sua posição atual, a posição $x_i(t+1)$ também pode ser calculada e sua equação é representada por (VALLE et al., 2008):

$$x_t(t+1) = x_i(t) + v_i(t+1)$$

O algoritmo do PSO executa repetidamente até que o critério de terminação seja atingido (optimalidade da solução) ou que as mudanças de velocidade das partículas estejam próximas de zero. Afim de evitar que o sistema extrapole o espaço de busca, limites vmax (positivo e negativo) são impostos para limitar a velocidade da partícula em cada dimensão d

do espaço, seguindo o seguinte critério: Se $v_i > vmax$ então $v_i = vmax$ e se $v_i < -vmax$, então $v_i = -vmax$ (EBERHART; KENNEDY, 1995; BANKS et al., 2007).

O primeiro passo da execução do algoritmo, representado na figura 14 é gerar aleatoriamente um enxame, considerando posição e velocidade das partículas, que serão então atualizadas a cada nova iteração junto com as variáveis *pbest* e *gbest*. Quanto um número máximo de iterações T é atingido, a melhor solução será $gbest = (best1(global), ..., bestD(global))$ onde D representa a dimensionalidade do espaço de busca, como citado anteriormente. Assim, o enxame de N partículas terá N soluções para cada iteração T e o *fitness* de cada solução é obtido então pelo *fitness* médio de n simulações replicadas (MARTINS et al., 2013).

Figura 14: Algoritmo básico do PSO

Inicialize um vetor população de partículas com posições e velocidade aleatórias
em um espaço de busca com dimensão D

Enquanto condição de terminação não for atingida **faça**
(em geral, um bom *fitness* ou máximo número de iterações T)
Para cada partícula i **faça**
Avaliar o *fitness* em D variáveis
Comparar o *fitness* da partícula com seu *pbest* e atualizar *pbest* caso o valor atual seja melhor que *pbest*
Identificar qual particula da população possui o melhor *gbest* até o momento e atualizar *gbest*
Atualizar a velocidade e posição da partícula de acordo com as equações de velocidade e posição
Fim
Fim

Fonte: Adaptado de (BOUSSAÏD et al., 2013; POLI et al., 2007)

Segundo Poli et al. (2007), o PSO possui alguns poucos parâmetros que precisam ser ajustados, quando em sua implementação original. Basicamente, devem ser ajustados o tamanho da população, $\Phi 1$ e $\Phi 2$, que representam a magnitude das forças aleatórias que influenciam na direção de cada indivíduo em busca de *pbest* e *gbest* e que são comumente chamados de coeficientes de aceleração e também *-Vmax ,+Vmax* que limitam o movimento dentro do espaço de busca.

Valle et al. (2008) afirma que o PSO não é amplamente afetado pela não-linearidade e pelo tamanho dos problemas e que pode convergir a soluções de problemas onde a maioria dos métodos analíticos irão falhar. Cita ainda que o PSO possui algumas vantagens em relação a outras técnicas semelhantes de otimização:

- Fácil de implementar e com poucos parâmetros para sintonizar;
- Por ser dotado de memória, cada partícula tem a capacidade de memorizar seu melhor

valor e também o melhor valor do enxame;

- É mais eficiente na manutenção da diversidade do enxame, já que cada partícula utiliza da informação de outras partículas para melhorar a si mesmas;
- As particulas são atualizadas em paralelo;
- Novas velocidades/posições dependem somente das velocidades/posições anteriores da partícula e de seus vizinhos;
- Todas as atualizações de velocidade/posição, para todas as partículas, seguem exatamente a mesma regra.

Em contrapartida, Banks et al. (2007) cita que a estagnação do enxame é reconhecida como a maior fraqueza do método e que isso pode levar a solução sub-ótima para o problema, mas que existem técnicas capazes de auxiliar na manutenção do movimento do enxame em busca de uma melhor solução, como por exemplo, a atribuição de um coeficiente de inércia ω a velocidade

A ausência de memória da velocidade (direção) de cada partícula também faria com que a mesma ficasse restrita a um ótimo global limitado as fronteiras iniciais do enxame, levando a solução a um ótimo local por consequência, porém, a existência de memória de velocidade leva a um comportamento totalmente inverso, expandindo os limites e proporcionando uma busca global dentro do espaço de busca e nesse sentido o coeficiente de inércia trabalha justamente para equilibrar a exploração do espaço. Um coeficiente grande leva a uma exploração global, ao passo que um coeficiente pequeno, leva a uma exploração local (BANKS et al., 2007; BOUSSAÏD et al., 2013).

A equação da velocidade, que inclui o peso de inércia, é dada por (BANKS et al., 2007; VALLE et al., 2008) :

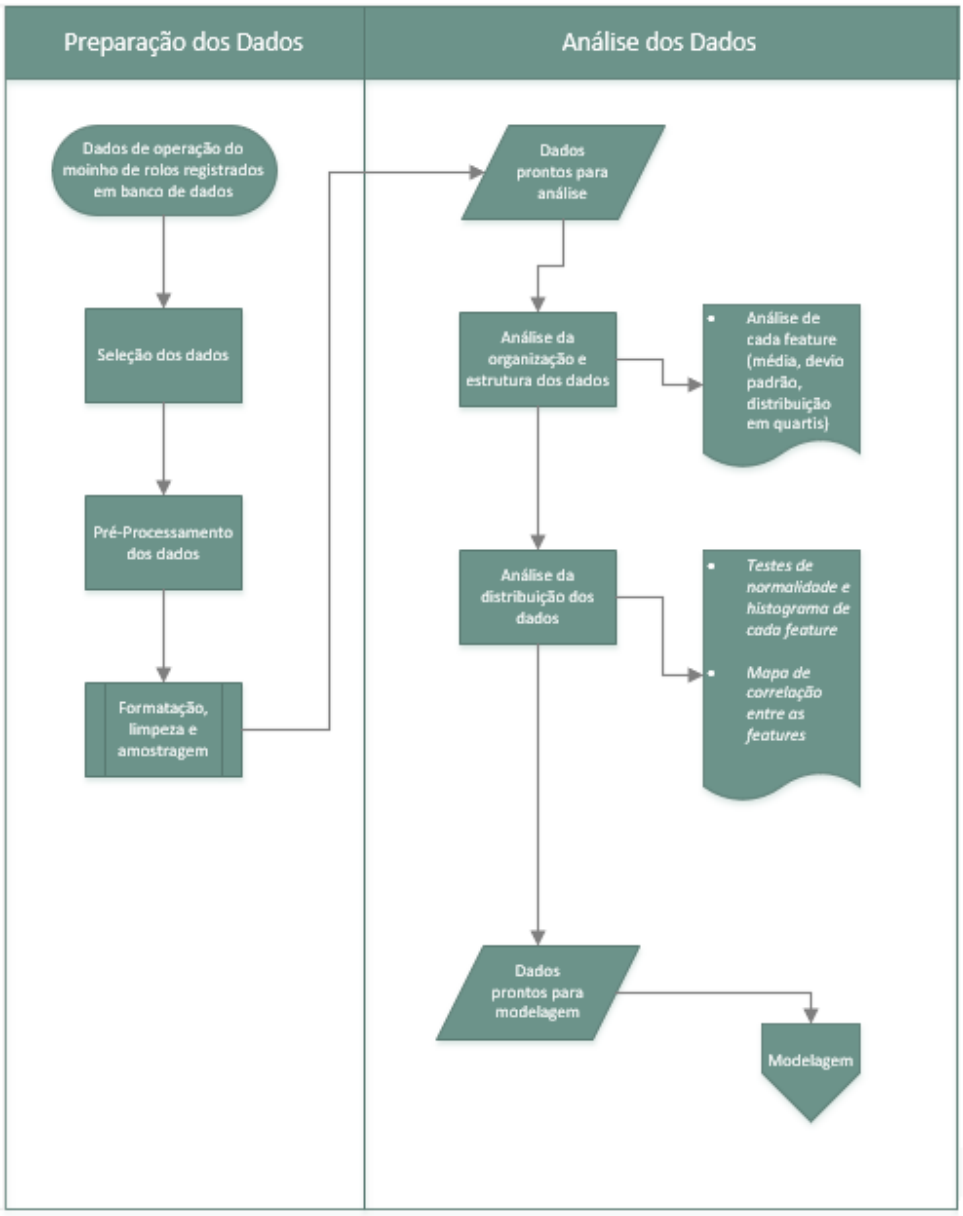
$$v_i(t+1) = \omega v_i(t) + \Phi 1 R1(pbest - x_i(t)) + \Phi 2 R2(gbest - x_i(t))$$

3 METODOLOGIA

A metodologia aplicada no desenvolvimento desse trabalho segue 4 etapas. As duas primeiras estão relacionadas a preparação e entendimento dos dados do moinho em estudo. A terceira etapa trata da modelagem utilizando a técnica de *Machine Learning* conhecida como *Random Forest* e a quarta e última etapa, trata do processo de otimização do moinho utilizando o PSO. Cada uma dessas etapas possui sub-etapas de análise e discussão, incluindo discussão de resultados nas etapas 3 e 4.

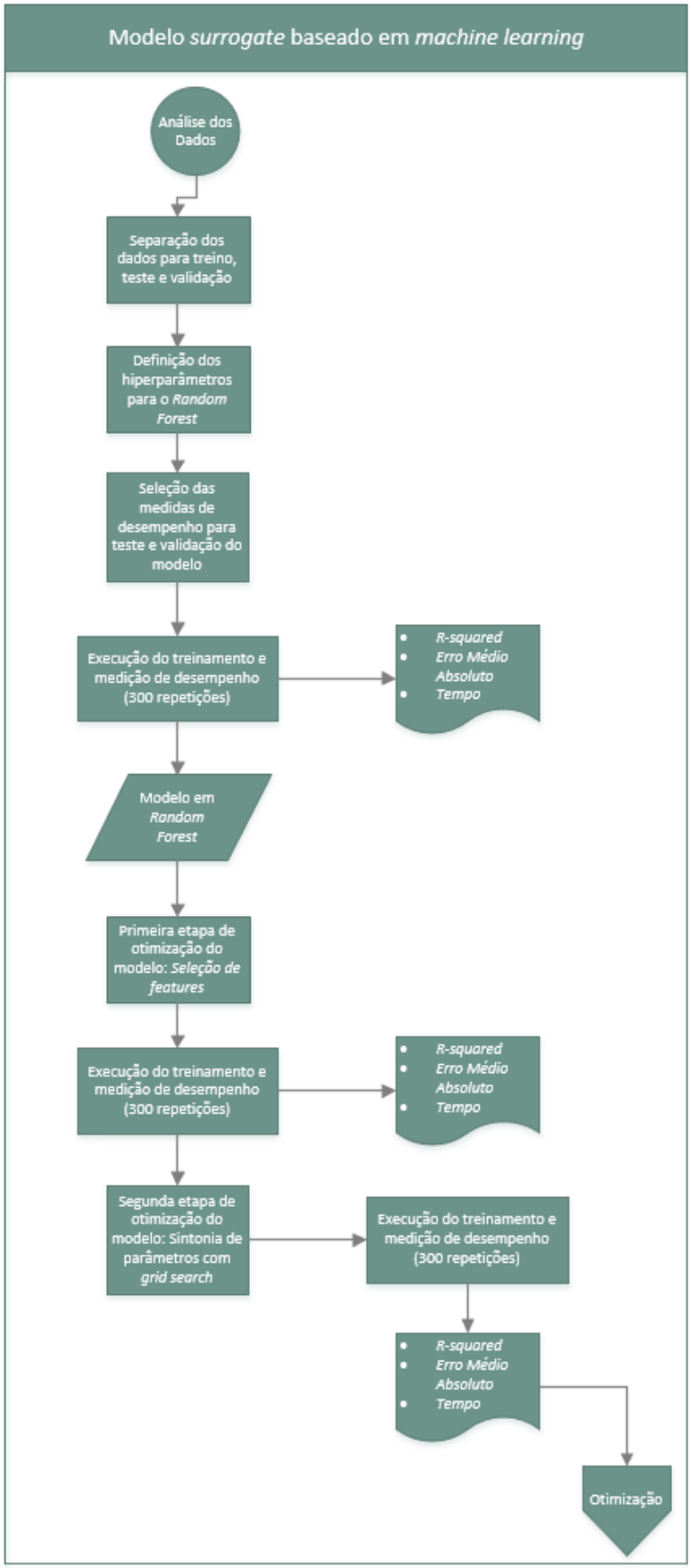
O fluxograma representado pelas figuras 15, 16 e 17 apresentam um panorama geral de tudo que será tratado nesse trabalho. O estudo se inicia com foco nos dados usados para modelagem, dentro dos capítulos 3.1 e 3.2, passa pela modelagem em *Random Forest* no capítulo 4, pela otimização usando o PSO no capítulo 5. Optou-se por separar os capítulos 4 e 5 da metodologia pois os mesmos possuem discussões de resultados parciais.

Figura 15: Fluxograma da Metodologia - Preparação e Análise dos dados



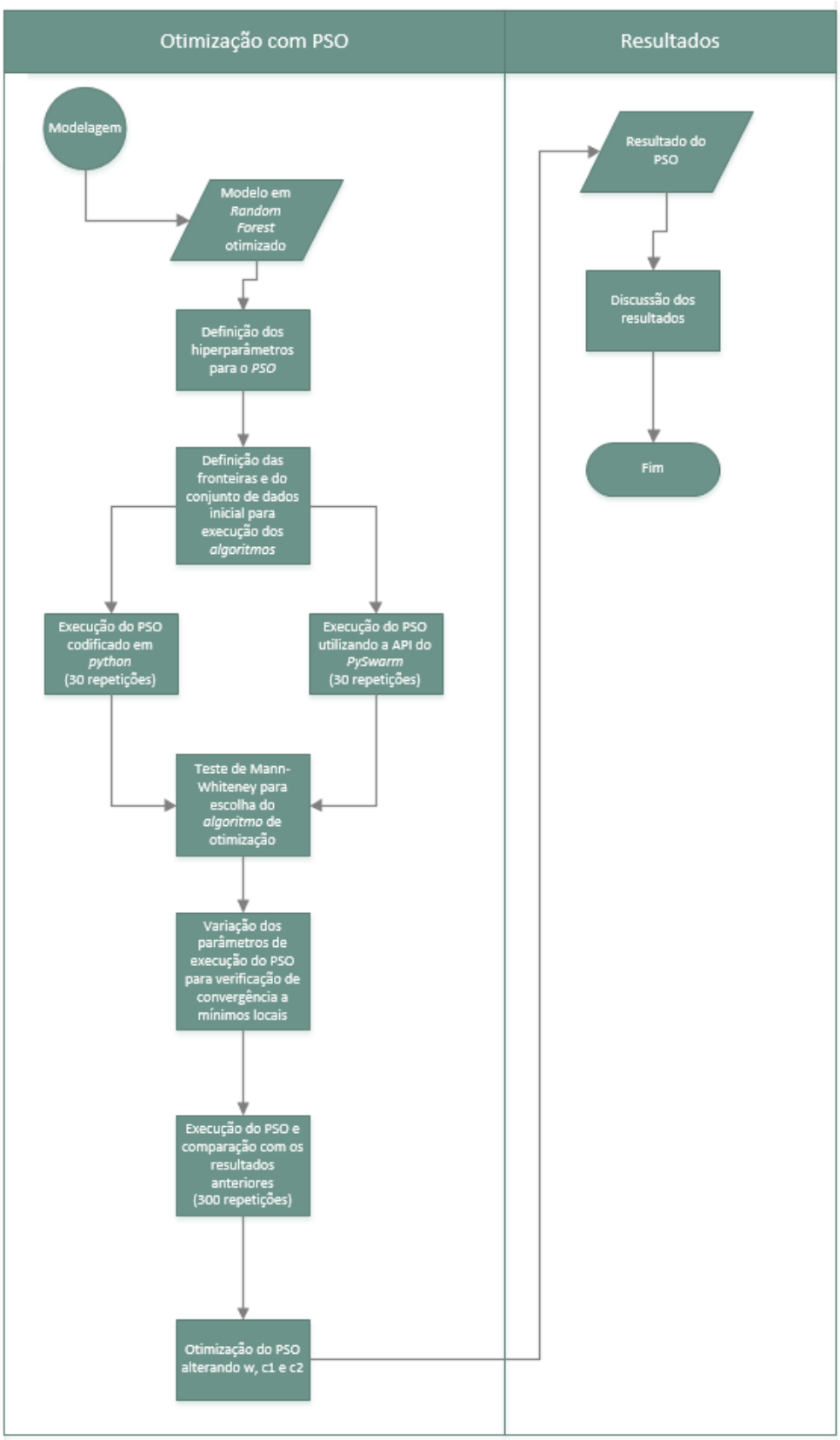
Fonte: Autoria Própria

Figura 16: Fluxograma da Metodologia - Modelagem



Fonte: Autoria Própria

Figura 17: Fluxograma da Metodologia - Otimização e Resultados



Fonte: Autoria Própria

3.1 PREPARAÇÃO DOS DADOS E PRÉ PROCESSAMENTO

Os dados de processo do moinho são armazenados em base estruturada, organizados por colunas, sendo inicialmente divididos em 20 colunas com os dados de entrada (parâmetros de operação do equipamento) e uma coluna com a saída (potência instantânea consumida no momento do registro). No total, foram usados para o estudo 38767 linhas, registradas entre 03 e 30/09/2020, com a frequência de um registro por minuto. Esses dados representam aproximadamente 646,11 horas de operação, sob as mais diversas condições intrínsecas e extrínsecas ao processo, contemplando também operação de cinco das seis receitas possíveis para a operação desse moinho, o que antecipadamente garante uma grande variabilidade nos dados de processo amostrados.

Como em qualquer modelo de *Machine Learning*, o pré processamento dos dados é uma etapa fundamental para obtenção de bons modelos e consequentemente de bons resultados. Neste trabalho, o pré processamento foi divido em duas etapas, formatação e limpeza.

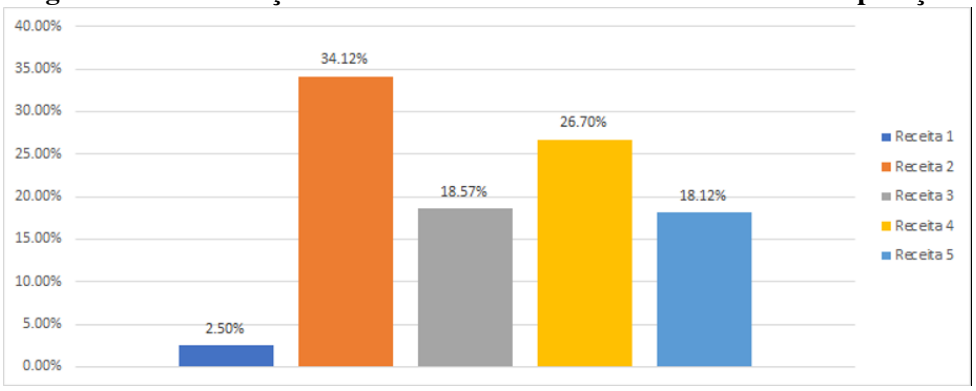
Na etapa de formatação, os dados foram convertidos do formato nativo do banco de dados, para o formato csv compatível com o Microsoft Excel. Ainda dentro da etapa de formatação, foram atribuídos nomes adequados a todos as colunas, já que no banco de dados o cabeçalho é formado pela identificação de cada uma das variáveis de processo.

Já na etapa da limpeza, um filtro foi aplicado para selecionar apenas os dados de processo amostrados com o moinho em pleno funcionamento, eliminando assim os registros realizados com o equipamento parado, em condição de falha, em processo de partida e também em processo de parada. Esse filtro foi necessário para que a modelagem pudesse ser realizada com dados que representassem realmente a operação normal do moinho e sob condições normais de operação já que o processo de otimização com o PSO, objetivo desse trabalho, busca otimizar o consumo de energia em operação.

Esse processo de filtragem foi realizado utilizando uma das *features* amostradas, que registra o status de operação do moinho. Após esse filtro, essa *feature* foi eliminada completamente (excluída a coluna) já que a mesma não tem relevância na modelagem do consumo de energia. Da mesma forma, a *feature* com a data da amostragem também foi eliminada pelo mesmo motivo.

Depois dessa primeira etapa de filtragem, a estrutura da base de dados para a modelagem passou a ter 23354 linhas e 19 colunas, relacionando a quantidade de dados com suas respectivas receitas, identificadas na figura 18.

Figura 18: Distribuição dos dados amostrados entre as receitas de operação

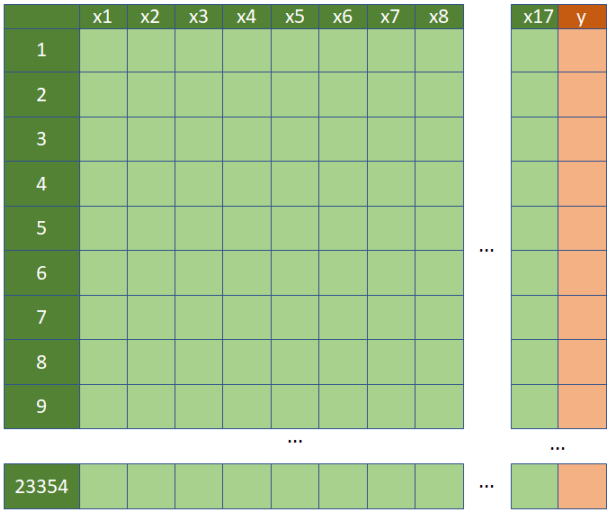


Fonte: Autoria Própria

Como o nome da receita por si só também não apresenta relevância na modelagem do consumo de energia do moinho, visto que seus parâmetros de processo com certeza o fazem, essa *feature* também foi eliminada da base de dados para a modelagem, ficando a estrutura final com 17 colunas representando as *features*, uma coluna representando a variável dependente y que queremos modelar e 23354 linhas, que foram posteriormente divididas em treino e teste.

A figura 19 representa a estrutura final dos dados para modelagem.

Figura 19: Estrutura do conjunto de dados para treinamento



Fonte: Autoria Própria

3.2 ANÁLISE PRELIMINAR DOS DADOS

A análise preliminar contempla o entendimento da estrutura dos dados que são usados na modelagem, suas distribuições e a relação estatística que possa eventualmente existir entre

eles, já considerando relação entre os dados de entrada com o dado de saída ou entre cada um dos dados de entrada. Essas relações nos permitem avaliar a possibilidade de, por exemplo, realizar uma redução na dimensionalidade da base de dados, trazendo ganhos de tempo de processamento ou resultado ao modelo.

3.2.1 ESTRUTURA E DISPERSÃO DOS DADOS

Dentro da estrutura dos dados, busca-se avaliar como esses dados estão organizados. No *Python*, uma função bastante útil nesse sentido é a *describe()*, dentro da biblioteca *Pandas* (MCKINNEY, 2010). Ela retorna oito resultados distintos, conforme abaixo:

1. Contagem de linhas onde os valores são diferentes de nulo, o que permite avaliar a integridade dos dados;
2. Média de todos os valores amostrados, dando uma ideia de onde está o centro da distribuição de cada coluna e apresentando um valor que pode ser usado como referência durante a etapa de análise;
3. Desvio padrão, representando a dispersão dos dados em torno da média;
4. Mínimo, demonstrando o valor mínimo encontrado dentro da distribuição dos dados de cada coluna;
5. 25, 50 e 75%, representando a distribuição dentro dos quartis e;
6. Máximo, demonstrando o valor máximo encontrado dentro da distribuição dos dados de cada coluna;

A tabela 2 representa o resultado da execução da função *describe()* na base de dados. A análise da quantidade de linhas nos permite afirmar que todas os parâmetros de entrada e também a variável dependente y que será modelada, encontram-se com todos os dados disponíveis para a etapa de modelagem. O fato de todas as colunas retornarem a mesma quantidade de linhas no permite afirmar que, pelo menos, nenhum dado foi corrompido ou está com valor nulo após as tratativas citadas no capítulo 3.1.

Tabela 2: Resultado da função *describe* para o conjunto de dados

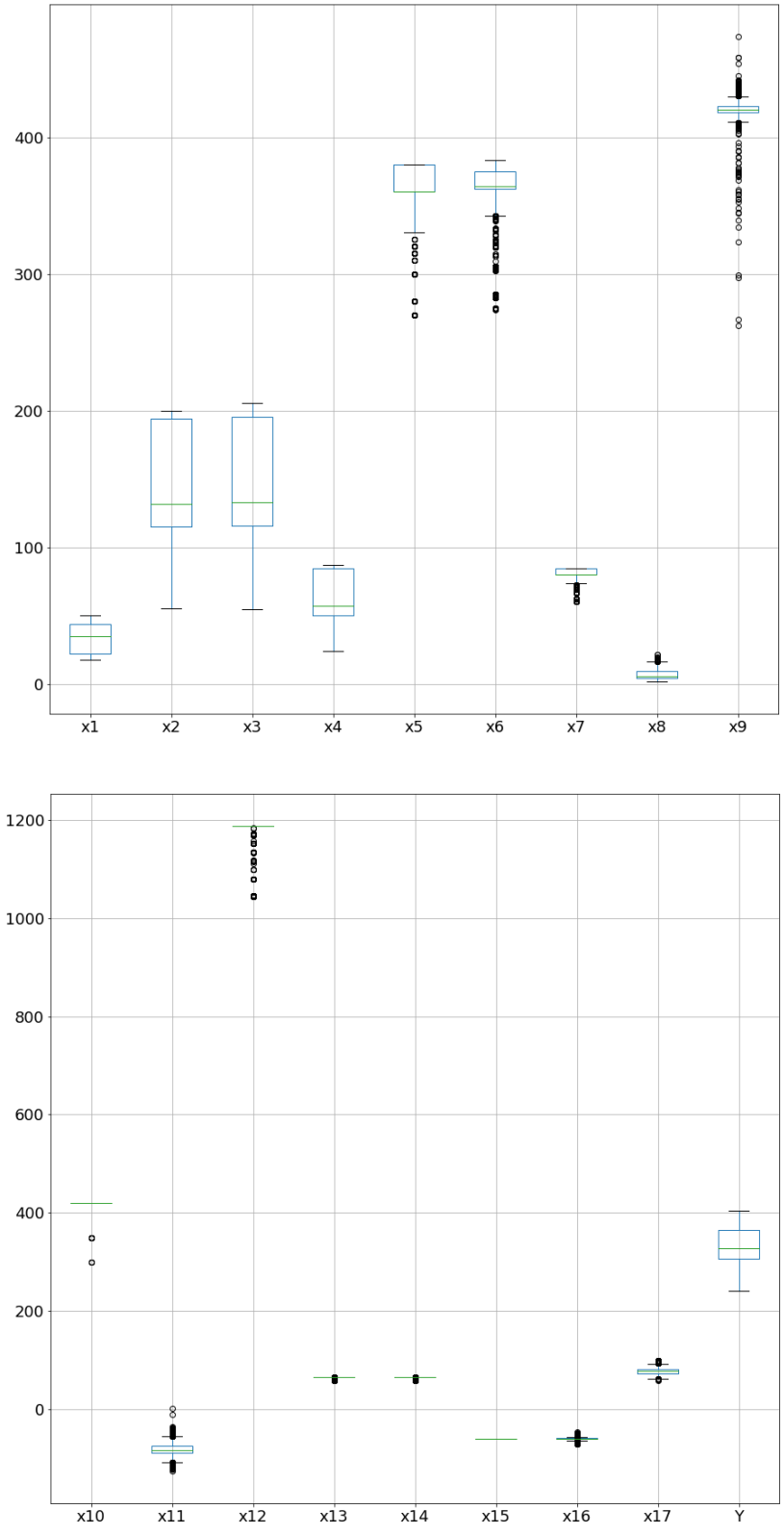
	Quantidade de Linhas	Média	Desvio Padrão	Minino	25%	50%	75%	Máximo
x1	23354	33.75	11.02	18	21.94	34.94	43.981	50
x2	23354	140.11	47.26	55	115	132	194	200
x3	23354	141.54	47.59	54.97	116.09	133.12	195.52	205.57
x4	23354	60.92	20.55	23.91	50	57.39	84.34	86.95
x5	23354	348.87	35.31	270	360	360	380	380
x6	23354	351.35	34.39	273.62	362.12	363.75	375.12	383.37
x7	23354	77.87	7.88	60.26	80.35	80.35	84.82	84.82
x8	23354	6.70	2.85	1.88	4.39	5.46	9.23	21.78
x9	23354	420.61	5.03	262.25	418	420.25	422.75	473.75
x10	23354	419.96	1.76	300	420	420	420	420
x11	23354	-81.62	10.23	-125.73	-88.57	-82.68	-75.18	1.83
x12	23354	1161.16	52.49	1044	1188	1188	1188	1188
x13	23354	64.50	2.91	58	66	66	66	66
x14	23354	64.50	2.91	58	66	66	66	66
x15	23354	-60	0	-60	-60	-60	-60	-60
x16	23354	-60	1.90	-71	-61	-60	-59	-46.25
x17	23354	77.27	4.93	58.09	73.47	77.19	81.29	100
y	23354	333.17	35.67	241.79	307.35	327.73	364.86	404.11

Fonte: Autoria Própria

A análise da média individualmente não permite muitas afirmações, porém, quando associada ao resultado do desvio padrão, é possível identificar que algumas *features* possuem pouca ou nenhuma variabilidade, demonstrando que mesmo com diferentes receitas e diferentes condições de operação, que o valor desse parâmetro de processo não se altera. Podemos verificar pouca variabilidade nos dados de pelo menos cinco parâmetros (x9, 10, 13, 14 e 16) e nenhuma variabilidade em um deles (x15). Esse resultado será usado posteriormente na etapa de seleção da *features*, capítulo 4.2.1

Finalmente, avaliar os mínimos, máximos e a distribuição entre os quartis nos permite avaliar a dispersão e a tendência central do conjunto de dados. Como a análise numérica torna-se mais complicada devido à grande quantidade de *features*, foram plotados os resultados em gráficos de caixa (*boxplot*) que permite uma análise visual desses resultados. A figura 20 abaixo representa o *boxplot* de cada uma das colunas da base de dados.

Figura 20: *Boxplot* dos dados amostrados



Fonte: Autoria Própria

Analisando os gráficos, nada mudou em relação a análise numérica realizada para os parâmetros x9, x10, x13, x14, x15 e x16. Além disso, é possível identificar outras características das *features* que podem, posteriormente, auxiliar no processo de entendimento do modelo e como citado anteriormente, na seleção das *features* com o objetivo de otimizar os resultados e/ou o tempo de execução.

É possível notar também, que as *features* x2, x3 e x4 (20) possuem as maiores dispersões dentre todas as existentes e que as mesmas possuem assimetria positiva, já que suas medianas estão próximas da linha divisória do primeiro quartil. Quanto ao tamanho das caudas, novamente os parâmetros que possuem pouca variabilidade nos dados se destacam, por não possuírem comprimentos grandes tanto nas inferiores quanto nas superiores, o que também indica dados bastante concentrados.

Por último, nota-se grande quantidade de *features* com muitos *outliers*. Isso se justifica quando analisamos o que cada uma dessas delas representa fisicamente no equipamento e se justifica pela forma como são calculados. Como exemplo, a x9, que possui a maior quantidade de *outliers*, representa o diferencial de pressão entre a entrada de matéria prima e a saída de produto acabado do moinho. Essa medida, em campo, é feita em milibares e possui bastante sensibilidade a qualquer alteração, mesmo que mínima, na entrada de matéria prima no moinho. Em relação ao cálculo, o *outlier* calculado por essa biblioteca em *Python* é dado pelas equações abaixo:

$$Inferior = PrimeiroQuartil - 1,5 * (TerceiroQuartil - PrimeiroQuartil)$$

$$Superior = TerceiroQuartil + 1,5 * (TerceiroQuartil - PrimeiroQuartil)$$

Sendo assim, as *features* x5, x6 e x7, que possuem suas medianas sobre o primeiro quartil (ou muito próximas deles) possuem somente *outliers* abaixo do limite da causa inferior. Em x12, os quartis e a mediana estão sobrepostos, porém não existem *outliers* acima do limite da cauda superior por uma limitação de processo, do próprio moinho.

3.2.2 DISTRIBUIÇÃO DOS DADOS

Continuando as análises sobre os dados para modelagem, foram verificadas as distribuições de cada uma das *features* quanto a sua normalidade e posteriormente entre elas. O teste de normalidade buscou saber se alguma delas possuia distribuição normal e o quanto isso poderiam impactar no resultado de y no pós modelagem. Já o teste de comparação

entre as mesmas, buscou saber se algumas delas possuíam origem na mesma distribuição e consequentemente se poderiam ser consideradas repetidas frente ao processo de modelagem.

Para o teste de normalidade, foi usado o teste de *Shapiro-Wilk*. Nesse teste, a hipótese nula é de que a população testada possui distribuição normal. Nesse caso, se o valor de *p-value* encontrado for menor que o valor atribuído para α, então a hipótese nula pode ser rejeitada e existe evidência que a população testada não é normalmente distribuída.

Para o teste foi atribuído α = 0,05, assumindo 95% de confiabilidade no teste. A tabela 3 apresenta os resultados do teste de *Shapiro-Wilk* para cada uma das *features* e também para a variável dependente y:

Tabela 3: Resultado do teste de *Shapiro-Wilk* para todo conjunto de dados

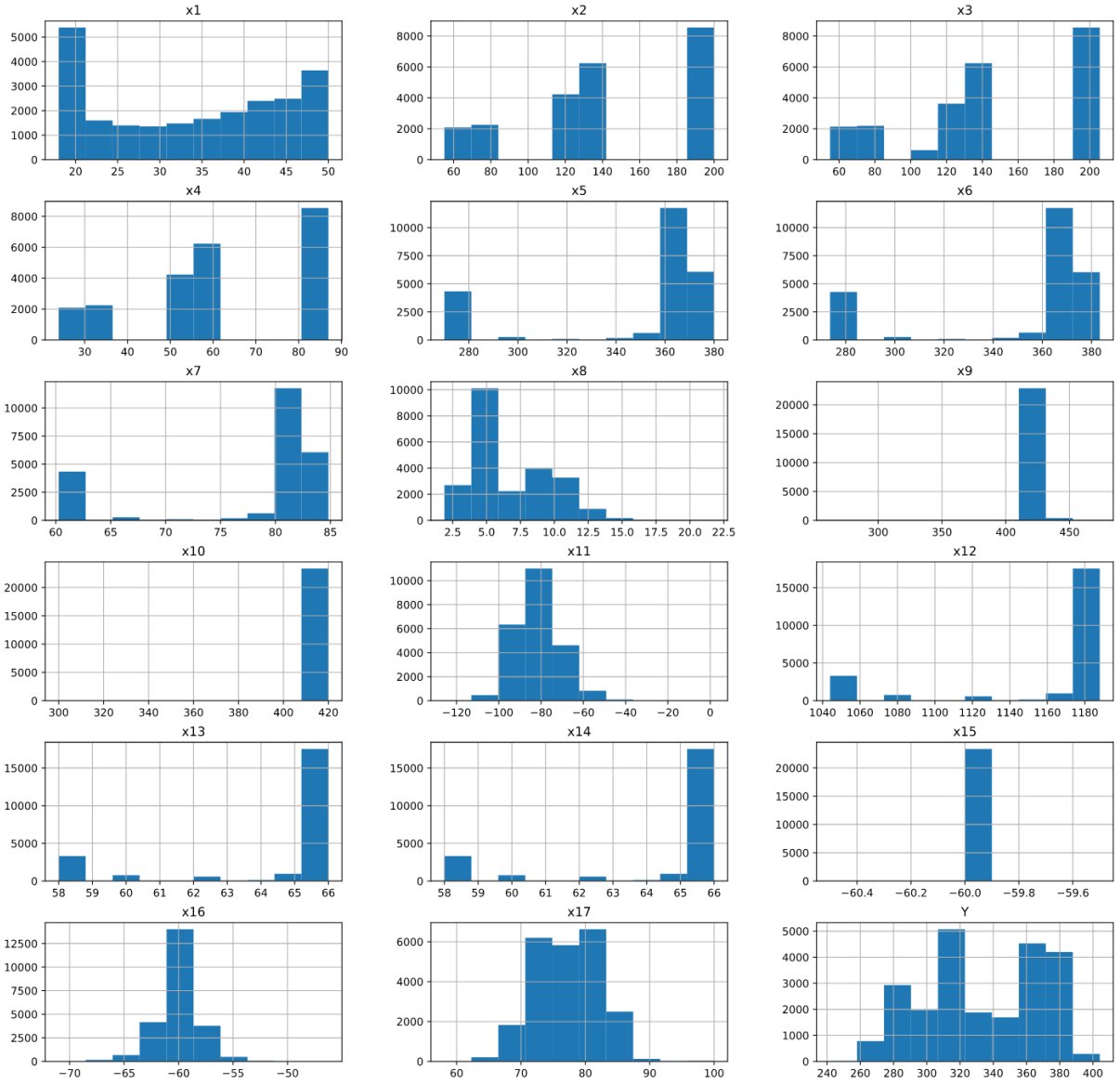
Feature	*p-value*	Resultado
x1	0.000	Não Normal
x2	0.000	Não Normal
x3	0.000	Não Normal
x4	0.000	Não Normal
x5	0.000	Não Normal
x6	0.000	Não Normal
x7	0.000	Não Normal
x8	0.000	Não Normal
x9	0.000	Não Normal
x10	0.000	Não Normal
x11	0.000	Não Normal
x12	0.000	Não Normal
x13	0.000	Não Normal
x14	0.000	Não Normal
x15	1.00	Normal
x16	0.000	Não Normal
x17	0.000	Não Normal
y	0.000	Não Normal

Fonte: Autoria Própria

Desconsiderando x15, que apresentou um falso positivo para normalidade, já que possui desvio padrão zero e média -60 podemos afirmar, com 95% de certeza, que não existem evidências de que alguma das *features* possua distribuição normal.

Da mesma forma, os histogramas abaixo podem confirmar o resultado numérico do teste de *Shapiro-Wilk*. A figura 21 representa os histogramas de todas as *features* e da variável dependente y, plotadas em 10 cestas:

Figura 21: Histograma dos dados amostrados



Fonte: Autoria Própria

Para o teste de comparação entre as *features* foi utilizado o teste estatístico de *Kruskal-Wallis*. Esse teste é usado para identificar se ao menos duas delas possuem origem na mesma distribuição de dados. Nesse teste, a hipótese nula é de que existem pelo menos duas *features* que possuem origem na mesma distribuição. Nesse caso, se o valor de *p-value* encontrado for menor que o valor atribuído para alfa, então a hipótese nula pode ser rejeitada e existe evidência de que na população testada, não existem pelo menos duas *features* com origem na mesma

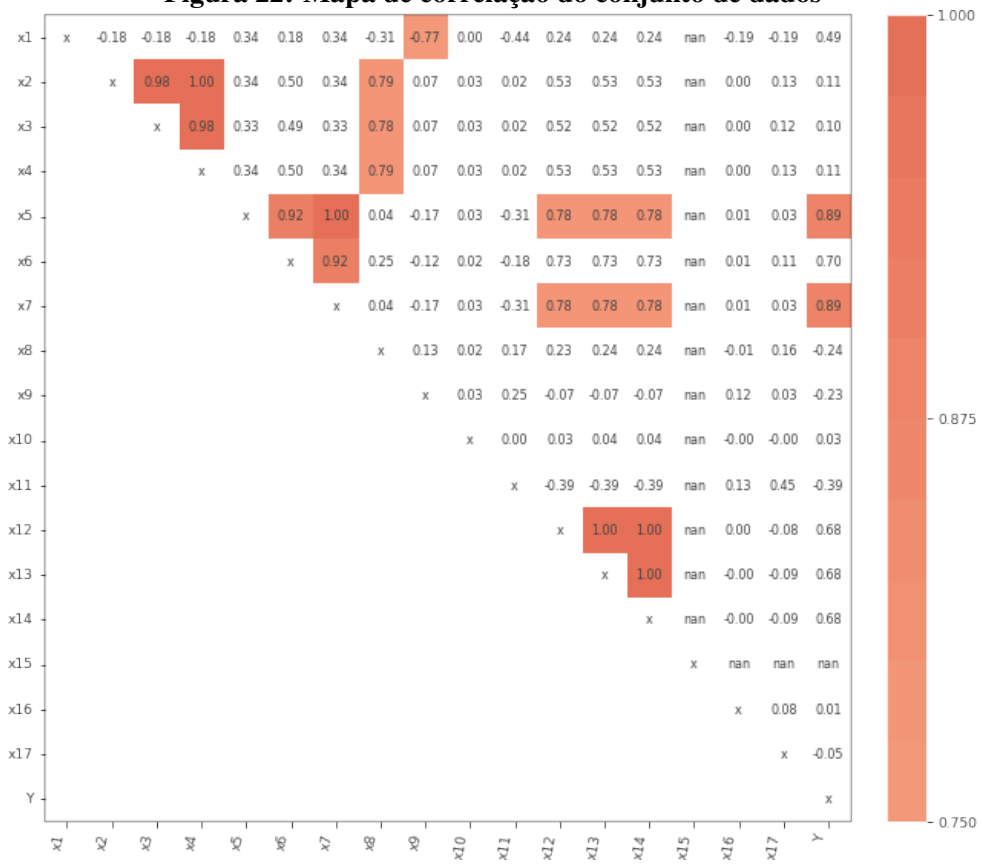
distribuição.

Para o teste, foi atribuído α = 0,05 e o resultado encontrado foi *p-value*=0,000 e portanto, com 95% de confiabilidade, podemos afirmar que não existem evidências de que pelo menos duas *features* possuem origem na mesma distribuição. Esse teste também considerou a comparação entre X e y e o resultado encontrado foi o mesmo.

3.2.3 CORRELAÇÃO DOS DADOS

O próximo teste realizado nos dados para modelagem foi o teste de correlação. O objetivo do teste foi identificar se existem *features* que se correlacionam fortemente e se alguma delas se correlaciona fortemente com y. Para o teste, foi usada a função *correlation heatmap* da biblioteca rfpimp em *Python*. A figura 22 representa o resultado do teste. O mapa de correlação identifica a existência de colinearidade entre as variáveis (correlação entre variáveis independentes) e de correlações entre alguma das *features* e a variável dependente y.

Figura 22: Mapa de correlação do conjunto de dados



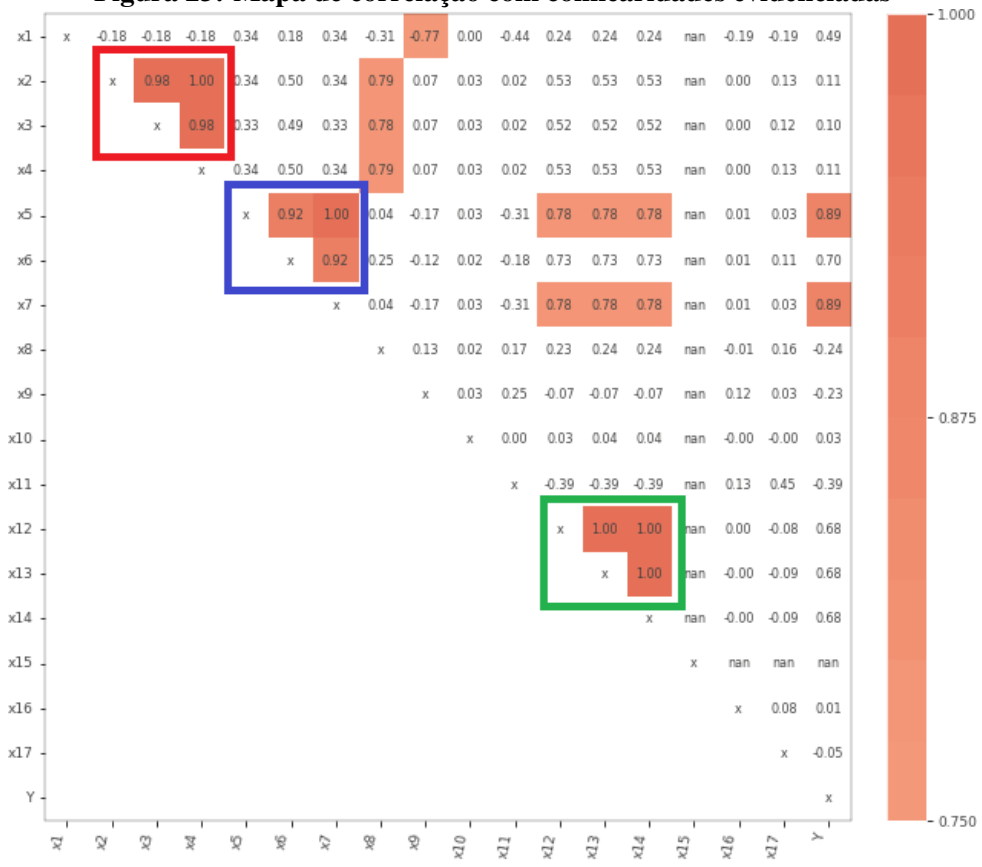
Fonte: Autoria Própria

A análise da figura 22 permite afirmar que existem colinearidades entre algumas

features, com R^2 acima de 0,92. Na figura 23 estão evidenciadas essas colinearidades. O quadro vermelho indica correlação forte entre x2, x3 e x4, o quadro azul indica correlação forte entre x5, x6 e x7 e por último, o quadro verde indica correlação forte entre x12, x13 e x14.

Essas correlações são explicadas pelas características de cada uma das *features* quando avaliadas do ponto de vista de processo, considerando o moinho de onde foram registrados os dados. As do quadro vermelho estão relacionadas ao classificador do moinho, as do quadro azul ao acionamento dos rolos enquanto as do quadro verde, ao ventilador de recirculação. Assim, as variáveis independentes, estando relacionadas ao mesmo sub-equipamento do moinho, estão fortemente correlacionadas entre si.

Figura 23: Mapa de correlação com colinearidades evidenciadas



Fonte: Autoria Própria

Ainda analisando as duas figuras com os resultados das correlações entre as variáveis, notamos a existência de uma coluna e uma linha com o resultado de ”nan”. Isso se dá pela característica da *feature* x15, de não possuir variabilidade e consequentemente, não possuir resultado possível de R^2, já que o mesmo considera justamente a variabilidade como parâmetro para o seu cálculo.

Outro resultado encontrado nos gráficos, diz respeito a correlação entre (x5,y) e (x7,y). Pode-se afirmar que existe uma correlação forte entre as duas variáveis independentes com a variável dependente y. Dessa forma, um modelo em *Machine Learning* eficiente, deve retornar um resultado maior que 0,89 de R^2 para que seja justificável o seu uso frente a um método linear, por exemplo.

4 MODELO *SURROGATE* BASEADO EM *RANDOM FOREST*

Este capítulo descreve a implementação de um modelo *Surrogate* para o consumo de energia do moinho de rolos através da técnica *Random Forest*, implementada utilizando a API do pacote *Scikit Learn* em *Python*.

Apesar desse trabalho ter considerado o *Random Forest* como método para obtenção do modelo de consumo de energia do moinho de rolos, o algoritmo original proposto por Breiman em (BREIMAN, 2001) não foi usado diretamente com esse propósito e portanto, não foi considerado durante a evolução do trabalho, a codificação desse algoritmo em alguma linguagem de programação. Nesse contexto, por questões práticas como tempo, quantidade de informações disponíveis, facilidade no uso e simplicidade, todo o trabalho de modelagem foi desenvolvido utilizando a API *Random Forest Regressor*, disponível no projeto *Scikit Learn*, desenvolvido e publicado em linguagem *Python*. Maiores detalhes sobre o projeto podem ser encontrados em Pedregosa et al. (2011) e maiores detalhes sobre as APIs em (BUITINCK et al., 2013). Outra fonte consistente de informações está disponível também em seu *website*, sob o endereço scikit-learn.org. O *Scikit*, conforme afirmam seus criadores, fornece ferramentas simples e eficientes para análise de dados, são ferramentas de código aberto, comercialmente utilizáveis e que são acessíveis a todos e reutilizáveis em variados contextos.

Não é objetivo desse trabalho avaliar e/ou comparar os resultados obtidos no uso das ferramentas do *Scikit* com outras bibliotecas e ferramentas disponíveis atualmente, nem mesmo comparar os resultados obtidos com o *Python* com outras linguagens como o R, apesar de existirem publicações em fóruns especializados que afirmam existirem diferenças sutis em algumas comparações de desempenho e resultados entre elas (XUE, 2015).

A API do *Random Forest* no *scikit learn* possui uma quantidade razoável de hiperparâmetros que podem ser usados durante a sua execução. Na sequência está um resumo de cada um deles (BUITINCK et al., 2013):

1. *N_estimators*: Total de arvores que serão criadas na floresta, durante a execução;

2. *Criterion*: Função que será usada para medir a qualidade de uma divisão em um nó;
3. *Max_depth*: Máxima profundidade que as arvores poderão atingir;
4. *Min sample split*: quantidade mínima de amostras disponíveis, necessárias para que seja feita uma nova divisão de um nó interno;
5. *Min_sample_leaf* : quantidade mínima de amostras necessárias para estar em um nó folha;
6. *Max features*: número de *features* que serão consideradas em busca da melhor divisão;
7. *Max_leaf_nodes*: número máximo de nós folhas permitido durante a execução;
8. *Min_impurity_spli*t: valor usado para limitar precocemente o crescimento das arvores;
9. *Bootstrap*: se amostras aleatórias dentro do conjunto de dados serão usadas para construir as arvores;
10. *OOB_score*: se serão usadas as amostras *out-of-bag* para calcular o $R\hat{2}$ com os dados não usados durante a execução;
11. *N_jobs*: número de trabalhos que serão executados em paralelo;
12. *Random_state*: controla a aleatoriedade na seleção das amostras usadas na construção das árvores e na seleção das *features* que serão usadas na busca da melhor divisão dos nós;
13. *Max_samples*: no caso do parâmetro *boostrap* ser verdadeiro, número máximo de amostras usados para treinar cada preditor.

Apesar da extensa lista de hiperparâmetros disponíveis na execução do algoritmo pelo *scikit learn*, é comum encontrarmos na literatura referências que indicam que apenas alguns deles são de fato relevantes para a grande maioria das aplicações. Os parâmetros básicos, que necessitam mais atenção para execução de modelos de regressão em *Random Forest*, conforme encontramos nas publicações de (BOULESTEIX et al., 2012; HUANG; BOUTROS, 2016; PROBST et al., 2019; TORRES-BARRÁN et al., 2019) são *max features, n_estimators, min_sample_leaf e max_samples*.

Uma explicação mais detalhada desses parâmetros pode ser encontrada no capítulo 4.2.2.

4.1 APLICAÇÃO DO *RANDOM FOREST*

No capítulo 3.1 foi descrita a estrutura final do conjunto de dados usado para a modelagem do consumo de energia do moinho. Essa estrutura, como visto, possui 17 colunas representando as *features* X, uma coluna representando a saída y do modelo e 23354 linhas que correspondem as amostras disponíveis. A divisão do conjunto de dados, conforme citado em tópico homônimo na fundamentação teórica desse trabalho, é parte importante do processo de obtenção de modelos em *Machine Learning*.

Para esse trabalho, duas abordagens distintas foram utilizadas. Para as etapas de modelagem (capítulo 4) e seleção de *features* (capítulo 4.2.1) foi usada divisão sem validação cruzada enquanto que, na etapa de *tuning* de parâmetros (capítulo 4.2.2) a validação cruzada foi empregada.

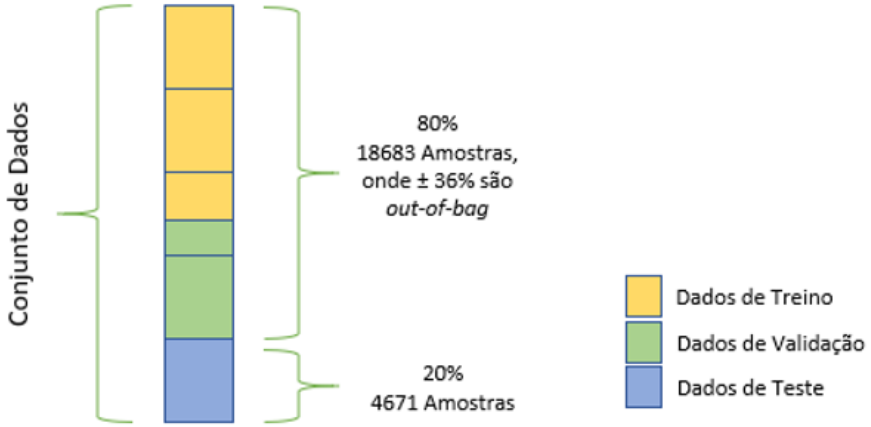
Considerando o modelo clássico de divisão de dados para utilização com algoritmos de aprendizagem supervisionada, deveriam ser considerados nessa etapa da modelagem, os dados para o treinamento, os dados para avaliação do resultado do treinamento e então dados para os testes de desempenho (BERK, 2008; ERTEL, 2017).

Nesse caso, cumprindo com o modelo clássico de divisão, optou-se por não utilizar a validação cruzada durante a modelagem e seleção de *features* por quatro fatores. O primeiro, devido a característica do *Random Forest*, que segundo cita Genuer et al. (2010), em geral, não utiliza próximo a 36% das amostras existentes e as classifica como *out-of-bag*, amostras essas usadas posteriormente na etapa de validação. O segundo, que o tempo necessário para executar a modelagem utilizando validação cruzada obriga a etapa de *fit* do modelo ser executada *k-folds* vezes para que todos os dados sejam posteriormente comparados. Assim, em situações onde a validação cruzada seja feita utilizando 5-*folds* por exemplo, o *fit* do modelo seria executado 5 vezes e o aumento de tempo para a execução da modelagem seria expressivo. O terceiro, é que existem publicações que afirmam terem obtido bons resultados usando as amostras *out-of-bag* para a etapa da validação do modelo, a citar Liaw et al. (2002), Genuer et al. (2010), Boulesteix et al. (2012), e por fim, o quarto e último fator, que não foi encontrado na literatura publicações que afirmam existir um número mínimo ideal de amostras que justifiquem o uso de validação cruzada ou não, enquanto alguns autores afirmam que esse tipo de decisão está muito associada ao contexto (BERK, 2008).

Portanto, considerando a quantidade de amostras disponíveis e os fatos citados no parágrafo anterior, a divisão dos dados para a etapa de aprendizagem da modelagem é representada pela figura 24, onde foram separados 80% do conjunto de dados para a etapa

de treino e validação, incluindo os dados que são *out-of-bag* e 20% para dados de teste.

Figura 24: Divisão dos dados aplicada a modelagem



Fonte: Autoria Própria

A próxima etapa dentro do processo de modelagem desse trabalho, trata da obtenção de um modelo padrão para comparação de desempenho nas etapas posteriores. Para obtenção desse modelo, que será citado a partir de agora como *baseline*, foram utilizados os hiperparâmetros com seus valores *default* na API do *Random Forest regressor* no *scikit learn* (BUITINCK et al., 2013).

Para a medida de desempenho do modelo obtido pelo *Random Forest*, a API utiliza o coeficiente de determinação, ou então R^2. Essa é uma forma quantitativa de medir o desempenho de um modelo de regressão e ela é derivada da decomposição da variância entre os resultados do modelo e o valor previamente existente para comparação (modelo de aprendizagem supervisionada). Nessa decomposição a variância total é separada em duas partes. A primeira é a soma dos quadrados da diferença entre o resultado do modelo e a média dos dados observados, representado por SQ_{exp} (HEUMANN et al., 2016):

$$SQ_{exp} = \sum_{i=1}^{n} (\hat{y}_i - \overline{y})^2$$

Onde $\hat{y}_i$ é o valor estimado de y_i e $\overline{y}$ é a média das observações. A segunda é a soma dos quadrados da diferença entre o valor observado e o valor estimado, representado por SQ_{erro}:

$$SQ_{erro} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

O termo SQ_{exp} mede a variabilidade explicada pelo modelo de regressão, enquanto SQ_{erro} reflete a variação advinda do erro aleatório envolvido no modelo.

O SQ_{total} é então a soma de SQ_{exp} e SQ_{erro}, representada matematicamente por:

$$SQ_{total} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

A interpretação possível para as duas equações acima, leva a crer que quanto maior o valor de SQ_{erro}, pior o resultado do modelo. Assim, o valor de R^2 pode ser representado pela relação entre SQ_{exp} e SQ_{total}, conforme a equação abaixo:

$$R^2 = \frac{SQ_{exp}}{SQ_{total}} = 1 - \frac{SQ_{erro}}{SQ_{total}}$$

Assim, pela definição acima, temos que $0 \leq R^2 \leq 1$ e que, quanto mais próximo de 1, melhor é o resultado do modelo encontrado, pois significa que o mesmo explica 100% da variância encontrada nos dados (HEUMANN et al., 2016).

Como resultados de $R^2 = 1$ não foram obtidos durante testes prévios no conjunto de dados, foi utilizado um segundo parâmetro de desempenho, sendo esse associado ao erro do modelo e conhecido como Erro Absoluto Médio - EAM.

O erro absoluto médio é dado pela equação abaixo e mede o erro absoluto entre o valor esperado y_i e o valor encontrado $\hat{y}_i$ (BUITINCK et al., 2013).

$$EAM(y, \hat{y}) = \frac{1}{n_{samples}} * \sum_{i=0}^{n_{samples}-1} |y_i - \hat{y}_i|$$

A opção pelo uso do erro absoluto médio como métrica para o modelo junto com o R^2, foi baseado no trabalho de (TORRES-BARRÁN et al., 2019), onde o autor afirma que usar o EAM ao invés de outras métricas que envolvem erros quadrados (como o erro quadrado médio, por exemplo), traz uma estimativa direta do desvio do modelo e consequentemente, nesse trabalho, resulta em erros que podem ser medidos em potência elétrica.

Por último, o tempo de execução do *fit* do modelo também foi usado como critério de análise de desempenho do *baseline* visto que executar a modelagem em menos tempo é um fator relevante para qualquer aplicação de *Machine Learning*.

O desempenho do *baseline* é listado na tabela 4. Para a obtenção desses resultados, foram inseridas algumas etapas em um laço com 300 repetições e ao final, uma média de cada um dos valores foi calculada. O algoritmo usado pode ser visto na figura 25:

Figura 25: Algoritmo de obtenção do modelo

Inicio: Carregar o conjunto de dados
Enquanto iterações diferente de 300x **faça**

Dividir o conjunto de dados, com 80% dos dados para treino e 20% dos dados para teste
onde os 36% resultantes do out-of-bag estão dentro dos 80% destinados a treino

Criar o modelo em RF usando parametros *default da API do scikit learn*

Fit do modelo, usando os dados out-of-bag para cálculo o R^2

Testar o desempenho do modelo usando os 20% de dados disponíveis após a divisão inicial,
calculando R^2, Erro Médio Absoluto

Calcular o tempo de execução da iteração

Salvar os resultados do teste de desempenho de cada iteração

Fim

Calcular as médias dos resultados abaixo e apresentar os valores
R^2 da etapa de fit do modelo
R^2 da etapa de teste do modelo
Erro Médio Absoluto da etapa de teste do modelo
Tempo de execução de cada iteração

Fonte: Autoria Própria

Um ponto importante que já foi citado anteriormente, é o uso dos dados *out-of-bag* para calcular o *fit* do modelo. Diferentemente disso, na etapa de teste, o cálculo do R^2 levou em consideração os 20% de dados separados inicialmente para teste, comparando o resultado do modelo com o resultado conhecido. Da mesma forma, o EAM considera os mesmos dados usados no cálculo do R^2, separando dessa forma os dados usados no *fit* e na predição, eliminando a possibilidade de dados já conhecidos pelo algoritmo tenham sido utilizados.

Outro ponto importante está relacionado com a etapa de divisão dos dados, como ela estava introduzida dentro do laço e possui uma característica estocástica para amostrar os dados que serão enquadrados em treino e teste, podemos afirmar que a cada iteração do laço, um novo conjunto de treino e teste foi criado para a execução da modelagem e consequentemente para as etapas de avaliação de desempenho também, o que novamente traz maior confiança nos resultados obtidos no *baseline*.

O resultado gráfico dessas 300 iterações podem ser vistos na figura 26, que traz a evolução dos valores de R^2 na etapa de *fit* (usando os dados *out-of-bag*), além do R^2 e EAM dos dados de teste e também do tempo de cada iteração.

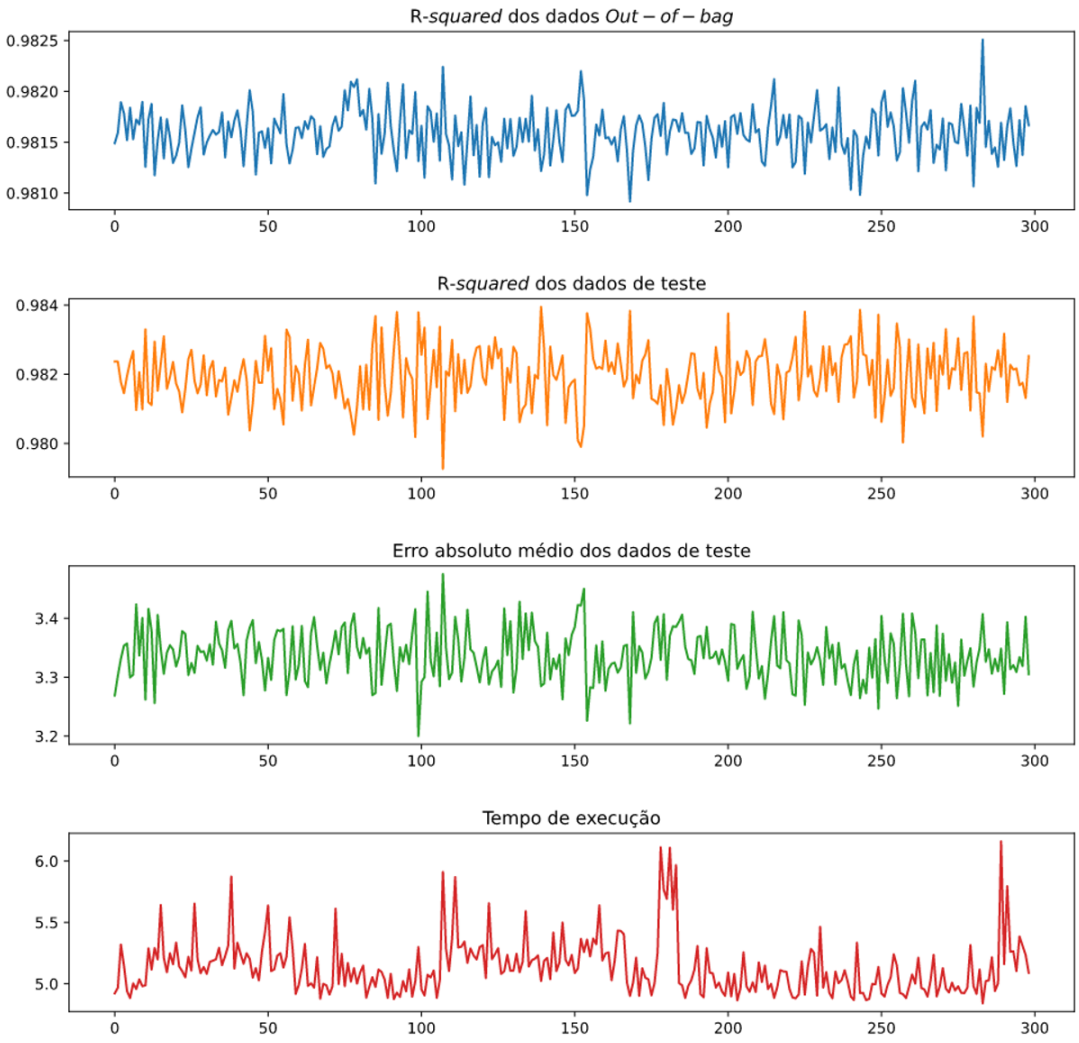
Tabela 4: Desempenho do *baseline* usando os parâmetros *default* do *Random Forest*

Desempenho		
R-squared (OOB)	0,981	
R-squared (Dados de Teste)	0,982	
Erro Absoluto Médio (Dados de Teste)	3,34	kW
Tempo de Execução	5,33	Segundos

Desvios Padrão		
R-squared (OOB)	0,0002	
R-squared (dados de teste)	0,0008	
Erro Absoluto Médio (Dados de Teste)	0,04	kW
Tempo de Execução	0,22	Segundos

Fonte: Autoria Própria

Figura 26: Resultado das 300 iterações para obtenção do *baseline*



Fonte: Autoria Própria

Levando em consideração os resultados do *baseline*, é possível identificar ótimo desempenho do *Random Forest* na modelagem do consumo de energia do moinho. O resultado de R^2 nos permite afirmar que pelo menos 98,1% de toda a variância dos dados pode ser representada pelo modelo encontrado. Da mesma forma, o erro absoluto médio encontrado na comparação entre a predição realizada pelo modelo, usando os 20% de dados destinados a teste, e os resultados previstos para esses dados, ficou muito próximo de 3,3kW, o que representa apenas 1,02% da mediana de todas as potências amostradas durante a coleta de dados de processo. Sobre o tempo de execução, podemos perceber que mesmo antes de aplicar qualquer método de otimização do modelo e considerando uma quantidade razoável de amostras no conjunto de dados, que o *Random Forest* foi capaz de apresentar resultados bastante satisfatórios, com tempo médio em torno de 5,33 segundos, provando ser realmente

bastante rápido.

Outro resultado importante pode ser visto nos desvios padrões calculados sobre as 300 iterações realizadas. Muito pouca variação é percebida entre os resultados de cada iteração, mostrando também a estabilidade do método.

Por fim, como já citado anteriormente nesse capítulo, o R^2 obtido com os dados *out-of-bag* apresenta praticamente os mesmos resultados obtidos calculando o R^2 com os dados de teste, o que comprova a não necessidade de dados de validação, quando utilizando o *Random Forest* como método de modelagem em *Machine Learning*.

4.2 OTIMIZAÇÃO DO MODELO

4.2.1 SELEÇÃO DE *FEATURES*

Os conjuntos de dados possuem uma característica de alta dimensionalidade atualmente, tornando desafiante a tarefa de promover boas análises sobre os dados envolvidos, assim como bons modelos que possam solucionar problemas. Uma parte importante do estudo de modelos em *Machine Learning* é a etapa de seleção de *features*. Essa tratativa é uma tentativa de manter o melhor conjunto de *features* dentre todas as disponíveis no conjunto de dados e que essas selecionadas sejam capazes de fornecer a melhor quantidade de informações que possam resolver o problema modelado, resultando assim em modelos com melhor desempenho (SALCEDO-SANZ et al., 2018; CAI et al., 2018).

O cerne dessa tarefa está no fato de que algumas *features* irrelevantes e/ou redundantes ao modelo podem, eventualmente, aumentar o custo de execução dos algoritmos (em termos de processamento) e também o tempo para que os modelos sejam processados, fazendo seu desempenho ser prejudicado de forma geral (SALCEDO-SANZ et al., 2018).

Segundo afirma Salcedo-Sanz et al. (2018) a seleção de *features* tem a capacidade de melhorar o desempenho e reduzir o tempo do processo de aprendizagem, além de simplificar os resultados. Além disso, o método aplicado deve ter alta acuracidade de resultados e baixo tempo de processamento

Não é objetivo desse trabalho aprofundar o estudo sobre seleção de *features*, promovendo análises detalhadas sobre os métodos aqui utilizados. O foco é melhorar o resultado do *baseline* aplicando diferentes métodos de seleção, buscando aquele com o melhor desempenho, aliado a baixo custo de processamento e tempo de execução.

Nessa etapa, foram utilizados os métodos *MIFS (Mutual Information-Based Feature*

Selection) (BATTITI, 1994), um *pipeline* do método *K-BEST* com o ANOVA *f-test score* (FISHER et al., 1921), o *RF-RFE (Recursive Feature Elimination) com Random Forest* (GRANITTO et al., 2006), o *Stability Selection com LASSO* (MEINSHAUSEN; BÜHLMANN, 2010), o método *CFS (Correlation-Based Feature Selection)* (HALL, 1999), e por último um ranqueamento manual realizado por um Engenheiro de Processos.

Os quatro primeiros métodos foram escolhidos pela sua disponibilidade de implementação no pacote de ferramentas do *scikit learn*. O método da correlação foi escolhido por se tratar de um método simples de avaliar, podendo ser feito inclusive visualmente utilizando o mapa de correlação apresentado no capítulo 3.2.3. Optou-se pelo ranqueamento manual, com o suporte de um Engenheiro de Processos, por efeito de comparação com os outros métodos e pelo entendimento de que bons resultados em modelos de *Machine Learning* também estão associados ao conhecimento daquilo que se está modelando.

Os testes foram realizados em três etapas. Na etapa 1, cada um dos cinco métodos computacionais foi implementado e testado individualmente e o resultado dos ranqueamentos pode ser encontrado nas tabelas 5, 6 e 7. As *features* foram ordenadas da menos importante para a mais importante, baseada nos critérios de cada método testado. Durante esse teste, uma *feature* aleatória foi introduzida e passou pelos mesmos critérios de testes de todas as outras, sem distinção. Os dados da mesma foram gerados usando uma distribuição normal padrão, com média = 0 e desvio padrão = 1. No conjunto de dados, a sua representação é x18.

Tabela 5: Resultado da execução dos métodos de seleção de *features*

MIFS		*K-BEST*		*RF-RFE*	
Feature	Resultado	*Feature*	Resultado	*Feature*	Resultado
x15	1,33227E-15	x15	nan	x15	18
x10	0,000555551	x18	0,061641432	x11	17
x18	0,001830189	x16	0,456262283	x17	16
x16	0,033290594	x10	13,32061352	x5	15
x9	0,13658547	x3	187,947756	x18	14
x11	0,229687212	x2	202,352551	x3	13
x1	0,267646407	x4	202,3846811	x10	12
x17	0,281113957	x17	231,6142797	x9	11
x14	0,616672929	x9	777,3762752	x16	10
x13	0,616972799	x8	2074,958644	x1	9
x12	0,629022124	x11	3710,597004	x8	8
x8	0,7043592	x1	6795,292631	x12	7
x5	0,994739356	x12	20986,59322	x6	6
x7	0,995290595	x14	20995,47208	x7	5
x3	1,296500914	x13	21000,8034	x2	4
x4	1,301390466	x6	33058,63825	x4	3
x2	1,301560898	x5	40825,65284	x14	2
x6	1,633175864	x7	40837,2708	x13	1

Fonte: Autoria Própria

Tabela 6: Resultado da execução dos métodos de seleção de *features - continuação*

Stability Selection		*CFS*		*Engenheiro de Processo*	
Feature	Resultado	*Feature*	Resultado	*Feature*	Resultado
x6	-2,667132822	x15	18	x2	9
x8	-1,937976369	x18	17	x4	9
x2	-0,198850593	x4	16	x5	9
x9	-0,091455589	x7	15	x7	9
x3	-0,090285815	x12	14	x8	9
x14	0	x14	13	x10	9
x15	0	x16	12	x13	9
x18	0,020500133	x10	11	x14	9
x11	0,034649437	x17	10	x15	9
x17	0,060551802	x3	9	x17	8
x10	0,11650824	x2	8	x16	7
x4	0,121472643	x9	7	x11	6
x16	0,131413451	x8	6	x1	5
x12	0,131502183	x11	5	x9	4
x1	0,164417021	x1	4	x12	3
x13	0,408019564	x13	3	x3	2
x5	0,874369176	x6	2	x6	1
x7	11,39391123	x5	1		

Fonte: Autoria Própria

Tabela 7: *Ranking* das *features* para cada teste aplicado

MIFS	*K-BEST*	*RF-RFE*	*Stability Selection*	*CFS*	Engenheiro de Processo
x15	x15	x15	x6	x15	
x10	x18	x11	x8	x18	x2
x18	x16	x17	x2	x4	x4
x16	x10	x5	x9	x7	x5
x9	x3	x18	x3	x12	x7
x11	x2	x3	x14	x14	x8
x1	x4	x10	x15	x16	x10
x17	x17	x9	x18	x10	x13
x14	x9	x16	x11	x17	x14
x13	x8	x1	x17	x3	x15
x12	x11	x8	x10	x2	x17
x8	x1	x12	x4	x9	x16
x5	x12	x6	x16	x8	x11
x7	x14	x7	x12	x11	x1
x3	x13	x2	x1	x1	x9
x4	x6	x4	x13	x13	x12
x2	x5	x14	x5	x6	x3
x6	x7	x13	x7	x5	x6

Fonte: Autoria Própria

Sem aprofundar a análise de comparação entre os métodos de seleção utilizados, vemos que os resultados, excluindo o apresentado pelo engenheiro de processo, se apresentam muito parecidos quando estamos considerando as menos e mais significativas, com uma variação considerável no ranqueamento das *features* intermediárias, entre os métodos. É evidente que alguns métodos classificaram x18 (laranja; tabela 7) de forma equivocada, porém considerando que a variável x15 não possui variabilidade, podemos destacar o *K-BEST* e o *CFS* como os únicos métodos que classificaram x18 corretamente. Na mesma linha de análise, a *feature* x6 parece ser a maia importante dentre todas as 17 existentes, por aparecer em posições de destaque em 4 dos 6 métodos adotados. Do ponto de vista de processo essa *feature* corresponde ao RPM do motor dos rolos, que é o maior motor da planta, o que corrobora o resultado encontrado.

Na segunda etapa da seleção foi implementado um *script* em *Python* para o teste de cada um dos resultados da etapa anterior. Como nenhum dos métodos anteriores

sugere a eliminação de uma *feature* (eles apenas criam um ranqueamento ou atribuem um coeficiente/peso as mesmas) esse algoritmo testou a eliminação de cada uma delas de cada um dos métodos usados, considerando a estratégia de *backward deletion*, onde o conjunto de dados inicia com todas as *features* e vai tendo uma a uma eliminada a cada iteração. Cada iteração é representada pela média de 50 execuções do *Random Forest*, sem validação cruzada, porém considerando os mesmos critérios usados para obtenção do *baseline*.

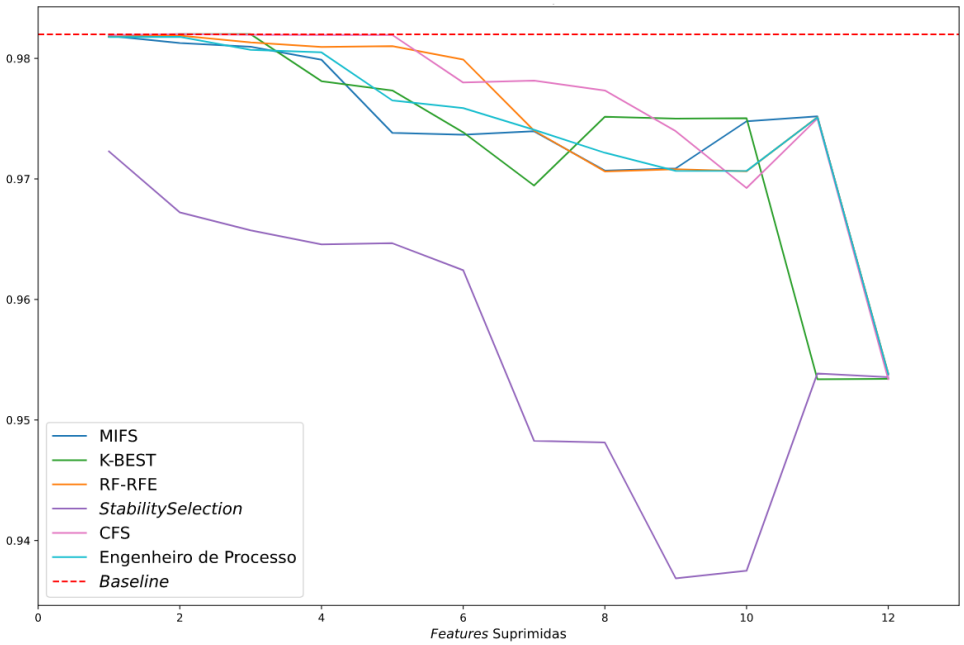
Nessa etapa, não foram eliminadas as *features* x2, x5, x10, x13 e x15 pois, como citado anteriormente, essas são as que serão usadas posteriormente na aplicação do PSO (estão marcadas em verde na tabela 7).

Como o *baseline* atingiu desempenho bastante relevante com todas as *features* disponíveis sendo utilizadas, essa etapa do processo de obtenção do modelo final foi focada em reduzir o tempo de execução ao invés de otimizar o R^2 ou diminuir o erro absoluto médio pelo entendimento de que, em etapas posteriores (*tuning* de hiperparâmetros e execução do PSO), o tempo é fator crucial no desempenho e aplicabilidade futura da solução criada para resolver o problema de otimização do consumo de energia do moinho.

Sendo assim, com critério adotado buscando a redução do tempo de execução, tentou-se não penalizar R^2 em mais de 1% e o erro absoluto médio em mais de 20%. O limite de R^2 foi arbitrado para evitar descolamento do desempenho obtido no *baseline* e o limite do erro absoluto médio, ao considerar que um acréscimo de 20% ao resultado obtido anteriormente (1,02% da mediana de todas as potências amostradas) levaria esse valor a apenas 1,22% da mediana, o que ainda é proporcionalmente bastante baixo.

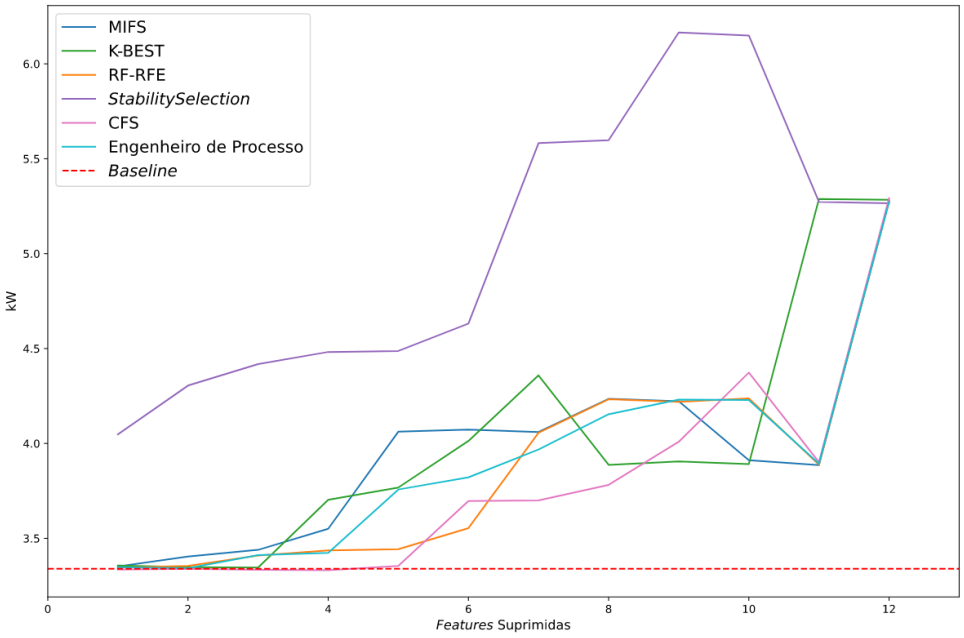
Foram plotados os gráficos representando os resultados do R^2, do erro absoluto médio e do tempo de execução da modelagem. Os mesmos podem ser vistos nas figuras 27, 28 e 29. O eixo X corresponde a *feature* eliminada em cada passo de iteração, usando como referência a tabela 7, onde a primeira eliminada é a menos relevante.

Figura 27: Resultado de R^2 durante a seleção de *features*



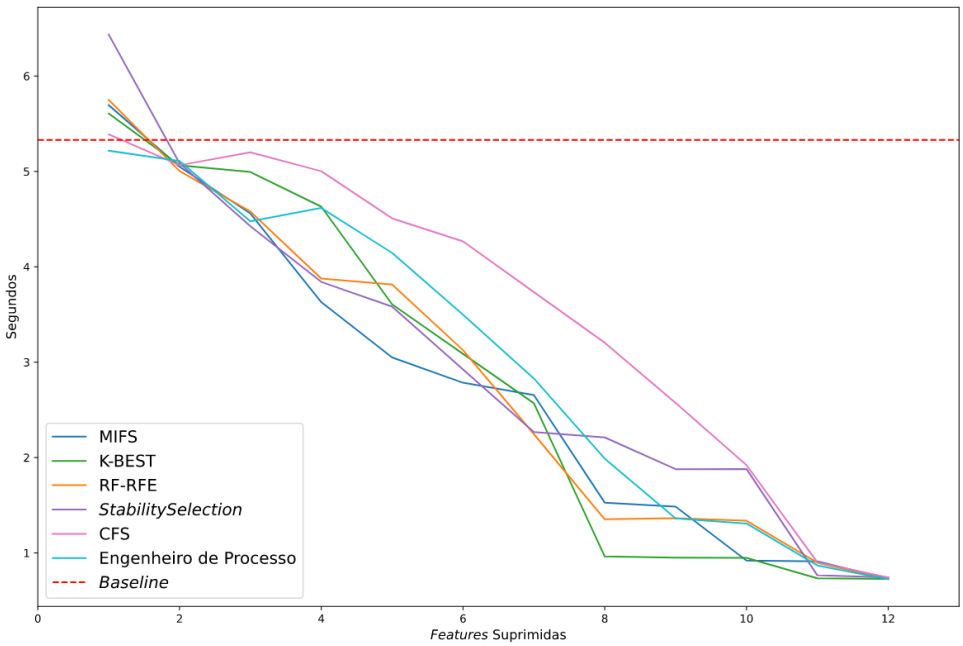
Fonte: Autoria Própria

Figura 28: Resultado de EAM durante a seleção de *features*



Fonte: Autoria Própria

Figura 29: Resultado de tempo de execução durante a seleção de *features*



Fonte: Autoria Própria

Analisando os três gráficos é possível identificar uma correlação negativa entre o R^2 e o erro absoluto médio. Utilizar menor quantidade de *features* para obtenção do modelo causa diminuição em R^2 e aumento no erro absoluto médio de forma muito proporcional. Quanto ao tempo, como esperado, reduzir a quantidade de *features* no modelo reduz drasticamente o tempo de execução.

Detalhando a análise dos gráficos e considerando os critérios usados para limitar a penalização do desempenho de R^2 e erro absoluto médio, foi montada a tabela 8 com os melhores resultados dentro de cada método de seleção testado. O *Stability Selection* não aparece na tabela pois não foi capaz de cumprir com os requisitos mínimos de erro absoluto médio, como pode ser visto na figura 28, apesar de ter conseguido um resultado razoável considerando apenas R^2.

O *K-BEST* se destacou como o melhor método de seleção, conseguindo suprimir um total de 10 *features* e ainda assim manter o desempenho do modelo dentro dos critérios estabelecidos para a etapa 2. A seleção manual realizada pelo engenheiro de processos, apesar de ter se apresentado bastante diferente dos resultados dos outros métodos (ver tabelas 5 e 6), resultou em um desempenho melhor que 4 dos métodos adotados nos testes. Os resultados estão listados na tabela 8.

Tabela 8: *Features* eliminadas por cada teste até atingir os critérios para seleção

Features	Baseline	MIFS	K-BEST	RF-RFE	CFS	Engenheiro de Processo
x1	x	x	x	x		
x2	x					
x3	x		x	x	x	
x4	x		x		x	x
x5	x					
x6	x					
x7	x				x	x
x8	x		x		x	x
x9	x	x	x	x	x	
x10	x					
x11	x	x	x	x		x
x12	x		x		x	
x13	x					
x14	x		x		x	x
x15	x					
x16	x	x	x	x	x	x
x17	x		x	x	x	x
Features	17	13	7	11	8	10
R-*squared*	0,982	0,980	0,975	0,980	0,974	0,974
Erro Absoluto Médio	3,34	3,55	3,89	3,55	4,01	3,97
Tempo de Execução	5,33	3,63	0,95	3,13	2,57	2,83
% de Ganho R-*squared*		-0,2%	-0,7%	-0,2%	-0,8%	-0,8%
% de Ganho Erro Absoluto Médio		-6,3%	-16,5%	-6,4%	-20,0%	-18,8%
% Ganho Tempo		31,9%	82,2%	41,3%	51,7%	46,9%

Fonte: Autoria Própria

Na etapa 3 do processo de seleção, foi realizado novo *resampling* com 300 iterações, dessa vez considerando somente as *features* que sobraram após a eliminação realizada na etapa 2. Os critérios e metodologia foram os mesmos adotados para encontrar o primeiro *baseline*.

Diante disso, os valores atualizados podem ser encontrados na tabela 9 e a comparação com os ganhos obtidos, na tabela 10

Tabela 9: Desempenho do modelo após seleção de *features*

Desempenho		
R-squared (OOB)	0,975	
R-squared (Dados de Teste)	0,974	
Erro Absoluto Médio (Dados de Teste)	3,90	kW
Tempo de Execução	0,91	Segundos
Desvios Padrão		
R-squared (OOB)	0,0026	
R-squared (dados de teste)	0,0010	
Erro Absoluto Médio (Dados de Teste)	0,05	kW
Tempo de Execução	0,08	Segundos

Fonte: Autoria Própria

Tabela 10: Comparativo entre os resultados do primeiro e segundo *baseline*

	Primeiro Baseline	Segundo Baseline	Ganho em %
R-*squared* (Dados de Teste)	0,982	0,974	-0,80%
Erro Absoluto Médio (Dados de Teste)	3,34	3,9	-16,90%
Tempo de Execução	5,33	0,91	83,00%

Fonte: Autoria Própria

O capítulo seguinte trata do *tuning* de parâmetros para o *Random Forest* e o segundo *baseline* será usado para efeitos de comparação com os resultados encontrados. Após a seleção da *features* com o *K-BEST*, o conjunto de dados ficou reduzido a 7 delas, ordenadas da seguinte forma: x2, x5, x6, x7, x10, x13, x15.

É possível que exista uma combinação de *features* que modele melhor o consumo de energia do moinho porém, como o objetivo desse trabalho é otimizar o consumo através da variação dos parâmetros de operação da planta, a única forma possível de criar um modelo que pudesse passar pelo PSO e ter seus parâmetros variados precisou considerar as entradas X que pudessem ter seus valores manipulados fisicamente no equipamento. Caso o objetivo fosse apenas modelar o consumo de energia e eventualmente fazer algum tipo de previsão futura

de consumo, sem alterar parâmetros de operação do processo, então esse modelo com essas *features* provavelmente não é o mais adequado.

4.2.2 *TUNING* DE HIPERPARÂMETROS

O processo de tuning está relacionado com a busca por hiperparâmetros que sejam considerados ótimos para uma técnica de *Machine Learning* aplicada a um determinado conjunto de dados. Uma das estratégias mais simples empregadas na busca por valores ótimos é o *grid search*, onde todas as combinações possíveis entre todos os hiperparâmetros, tem seu desempenho avaliado (PROBST et al., 2019).

O *Random Forest* possui certa robustez frente a escolha de seus hiperparâmetros, com seu desempenho estando menos relacionados a parametrização quando comparado a outras conhecidas técnicas de *Machine Learning*, apesar de existirem resultados comprovados de aumento de desempenho em alguns casos (COURONNÉ et al., 2018; MOON et al., 2018). Nesse contexto, Huang e Boutros (2016) cita que os hiperparâmetros *default* propostos por Breiman (2002) levam a bons resultados.

Ao comparar vários trabalhos publicados onde foram implementados modelos em *Random Forest*, Huang e Boutros (2016) encontraram estudos de parametrização somente em metade deles. O autor sugere que essa deficiência está relacionada a dificuldade desse tópico, a experiência limitada ao aplicar o método, a ausência de heurísticas prontas e de fácil implementação para serem aplicadas aos mecanismo de busca e também recursos limitados para implementação.

Como citado anteriormente no capítulo 4 foram escolhidos quatro hiperparâmetros baseados em estudos recentes de diferentes autores, onde os mesmos sugerem que os escolhidos são, em geral, os que apresentam resultados mais significantes para a etapa de *tuning*.

Os quatro hiperparâmetros que foram testados são:

1. *Max features*;
2. *Min_samples_leaf*;
3. *Max_samples*;
4. *N_estimators*.

O hiperparâmetro *max feature* relaciona a quantidade de *features* que serão aleatoriamente escolhidas como candidatas a cada nova divisão. Um valor pequeno aumenta

a chance de uma com baixa relevância ao modelo ser escolhida, o que leva a um melhor resultado, visto que a possibilidade dela ser suprimida por outra com mais relevância, é menor. Por outro lado, valores maiores reduzem o risco de apenas *features* com baixa relevância serem escolhidas, o que derrubaria o desempenho do modelo (COURONNÉ et al., 2018; BOULESTEIX et al., 2012; PROBST et al., 2019).

Sobre *max feature*, Biau e Scornet (2016) contrapõe os resultados dos trabalhos publicados por Díaz-Uriarte e Andres (2006) e Genuer et al. (2010), onde o primeiro afirma que esse hiperparâmetro possui impacto sensível ao desempenho do modelo, onde valores muito altos tendem a piores resultados, enquanto o outro, afirma que o valor atribuído deve ser o maior possível dentro da dimensionalidade das *features* existentes. Esse contraponto de resultados corrobora a opinião de Torres-Barrán et al. (2019), que afirma não existir em geral uma estratégia de tuning que possa ser usada em todos os problemas existentes atualmente e de Boulesteix et al. (2012), que cita que os dados disponíveis em mãos influenciam no valor ótimo para esse hiperparâmetro.

Assim como proposto por Wang et al. (2018) em seu trabalho e por não haver concordância dentro da academia, a proposta de *tuning* de *max feature* aqui relacionada não considerou sugestões existentes na literatura, onde diferentes valores baseado em diferentes premissas, são adotados para esse hiperparâmetros (TORRES-BARRÁN et al., 2019; BIAU; SCORNET, 2016; BREIMAN, 2002).

Com a dimensionalidade do conjunto de dados tendo sido reduzida na etapa de seleção (de 17 para 7), o entendimento de que todas as combinações poderiam ser usadas foi adotado, garantindo assim um valor ótimo para *max feature* quando testado em conjunto com os outros hiperparâmetros.

O próximo hiperparâmetro relacionado a etapa de *tuning* é o *min_samples_leaf*. Atribuir um valor k a ele significa que um nó precisa ter pelo menos k amostras antes de ser novamente dividido, portanto ele funciona no sentido de limitar o crescimento em altura da árvore (COURONNÉ et al., 2018).

Sendo assim, valores pequenos ocasionam o crescimento de árvores maiores, levando a necessidade de mais divisões até atingir o suficiente para os nós serem considerados folhas (PROBST et al., 2019).

Breiman (2002) afirma que em geral, o valor *default* (*min_samples_leaf* = 1) sempre reproduz bons resultados. Probst et al. (2019) por sua vez, cita que muitos dos pacotes de software que disponibilizam o *Random Forest* em sua API, possuem esse valor *default* = 1 para

problemas de classificação e 5 para problemas de regressão. Nesse trabalho, optou-se por testar valores próximos de 5, buscando alguma variação que possa desempenhar melhores resultados.

O terceiro hiperparâmetro relacionado é *max_samples*, que representa quantas amostras serão usadas durante o treinamento de cada árvore da floresta. Reduzir o valor de *max_samples* leva a melhorias de desempenho do modelo por possibilitar o crescimento de árvores mais diversas e com isso reduzir a correlação entre elas. Contudo, isso leva a uma diminuição do desempenho individual de cada árvore, já que menos amostras são usadas para treinamento (PROBST et al., 2019). O mesmo autor afirma que estudos recentes mostram que o valor ótimo para esse hiperparametro depende do problema a ser modelado e que bons desempenhos foram obtidos com valores menores que o *default*, somando-se a isso uma redução grande no tempo de execução também.

Nesse trabalho, foram testados valores proporcionais variando de 10 a 100% da quantidade de amostras disponíveis para a etapa de treino.

Por último, *n_estimators* está relacionado a quantidade de árvores que irão crescer na floresta e seu valor deveria aumentar proporcionalmente a quantidade de *features* existentes, dando assim a oportunidade suficiente para que todas sejam selecionadas como candidatas (BOULESTEIX et al., 2012).

Torres-Barrán et al. (2019) afirma não ter observado uma sensibilidade muito grande no resultado dos testes ao variar a quantidade de arvores criadas desde que as mesmas existam em quantidade suficientes para lidar com a complexidade do modelo proposto. Já Biau e Scornet (2016) e Probst et al. (2019) citam que a variância do modelo diminui com o crescimento da floresta e que portanto, um desempenho melhor deve ser obtido com a escolha de um valor grande para esse hiperparametro, levando em consideração que o custo computacional de aumentar a quantidade de árvores na floresta, cresce linearmente conforme aumenta-se o valor de *n_estimators*.

Probst et al. (2019) comparou alguns estudos com diferentes tamanhos de conjunto de dados e afirma que os resultados mostram que os maiores desempenhos podem frequentemente ser atingidos com 100 árvores apenas. Já Boulesteix et al. (2012) sugere aumentar os valores e parar assim que a medida de desempenho estabilizar ou ultrapassar um valor arbitrado.

Alguns autores afirmam não considerar *n_estimators* um hiperparâmetro (COURONNÉ et al., 2018; BOULESTEIX et al., 2012), visto que ele deve ser o maior possível já que aumenta assim a confiança nos resultados. Independente disso, após realizar o *tuning* de *max_features, min_samples_leaf e max_samples*, optou-se por testar algumas

combinações buscando melhorar o resultado da variância e consequentemente de R^2, sem comprometer o tempo de execução.

Optou-se por utilizar o *grid search* como estratégia de busca devido a sua simplicidade de implementação (3 laços de FOR, sendo um para cada hiperparâmetro) e por testar todas as combinações possíveis dentre todos os hiperparâmetros selecionados.

O *tuning* buscou otimizar o valor do erro absoluto médio, que havia sido mais penalizado na etapa anterior de seleção de *features* porém, sem penalizar as outras duas medidas de desempenho (R^2 e tempo de execução).

Apesar da afirmação de Breiman (2002) de que o *Random Forest* não precisa de validação cruzada para estimar um resultado sem viés para os dados de validação/teste, optou-se por utilizar o *k-fold* nessa etapa de *tuning*, devido a grande quantidade de combinações testadas levar a condições bastante diferentes das propostas originalmente por Breiman, quando o mesmo definiu os valores *default* para os hiperparâmetros.

Diante do que foi exposto nos parágrafos anteriores, o *grid* de busca ficou definido pelos itens 1 a 3 e o item 4 combinado com o melhor resultado do *grid* posteriormente:

1. *Max features* = [1, 2, 3, 4, 5, 6, 7];
2. *Min_samples_leaf* = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
3. *Max_samples* = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0], representando frações do total de amostras disponíveis no conjunto de testes;
4. *N_estimators* = [100, 105, 110, 115, 120]

Portanto, o *grid* considera 700 combinações diferentes de hiperparâmetros testados e somando aos testes realizados pós *grid*, já com as variações previstas de *n_estimators*, foram ao total 705 combinações distintas testadas na etapa de *tuning*.

O *script* para o *grid search*, criado em *Python*, ajuda na interpretação dos resultados, mostrando ao final da execução quais os hiperparâmetros obtiveram os melhores resultados, apresentando também os mesmos.

Dentre as 700 combinações testadas, a que obteve melhor desempenho foi a da tabela 11, com seus respectivos resultados. Como esperado, *min_samples_leaf* apresentou como melhor resultado o valor *default* e isso está associado a escolha do erro absoluto médio como critério de seleção. Se o tempo de execução fosse o critério, provavelmente valores

Tabela 11: Valores selecionados pelo *Grid Search* e seu respectivo desempenho

Hiperparâmetro	Valor
max_features	4
min_samples_leaf	1
sample_size	0,6

Desempenho		
R-*squared*	0,975	
Erro Absoluto Médio	3,884	kW
Tempo de Execução	0,739	Segundos
Tempo Total de Execução	2417,81	Segundos

Fonte: Autoria Própria

que provocassem um menor crescimento das árvores seriam os escolhidos. No que se refere a *max features*, um valor intermediário resultou em desempenho melhor, assim como em *max_samples*, que performou melhor com 60% da amostra disponível para treino sendo usada para o treinamento de cada árvore. Novamente, caso outro critério de escolha fosse usado, seguramente os resultados seriam diferentes. Interessante ressaltar novamente a velocidade de execução do *Random Forest*, que mesmo usando validação cruzada, concluiu o teste de todas as 700 combinações em aproximadamente 2418 segundos. Sobre o tempo baixo de execução, Breiman (2002) já havia notado excelente desempenho em seus testes, que mesmo sendo realizados em um computador com baixa capacidade de processamento quando comparado aos atuais (CPU de 250mhz), atingiu resultados muito interessantes ao criar uma arvore a cada 4 segundos em um cenário com 15mil amostras e 16 *features*, com *max feature* = 4.

Os hiperparâmetros escolhidos pelo *script* em *Pytho*n foram então combinados com os casos de teste para *n_estimators* e o resultado final do *tuning* pode ser visto na tabela 12. Nesse caso, o desempenho do modelo após a inclusão de *n_estimators* na rotina não superou o desempenho do erro absoluto médio encontrado com o *grid search* e portanto, não será considerado.

Tabela 12: Resultados considerando *Grid Search + n_estimators*

Hiperparâmetro	Valor
max_features	4
min_samples_leaf	1
sample_size	0,6
n_estimators	110

Desempenho		
R-*squared*	0,975	
Erro Absoluto Médio	3,891	kW
Tempo de Execução	0,763	Segundos

Diante disso, novamente foi realizando o *resampling* com 300 iterações, utilizando os resultados presentes na tabela 11 com os resultados do *grid search* e com isso foi gerado um novo *baseline*, que pode ser visto na tabela 14. Na tabela 15 a comparação final desde o primeiro até o terceiro *baseline*.

Tabela 13: Desempenho do modelo após *tuning* de hiperparâmetros

Desempenho		
R-*squared* (OOB)	0,975	
R-*squared* (Dados de Teste)	0,975	
Erro Absoluto Médio (Dados de Teste)	3,90	kW
Tempo de Execução	0,87	Segundos

Desvios Padrão		
R-*squared* (OOB)	0,0003	
R-*squared* (dados de teste)	0,0011	
Erro Absoluto Médio (Dados de Teste)	0,05	kW
Tempo de Execução	0,08	Segundos

Fonte: Autoria Própria

Tabela 15: Comparativo entre os resultados dos três *baselines*

	Primeiro Baseline	Segundo Baseline	**Ganho em %**	Terceiro Baseline	**Ganho em %**	**Ganho Total em %**
R-*squared*	0,982	0,974	**-0,8%**	0,975	**0,1%**	**-0,7%**
Erro Absoluto Médio	3,34	3,90	**-16,9%**	3,90	**0,2%**	**-16,7%**
Tempo de Execução	5,33	0,91	**83,0%**	0,87	**4,4%**	**83,8%**

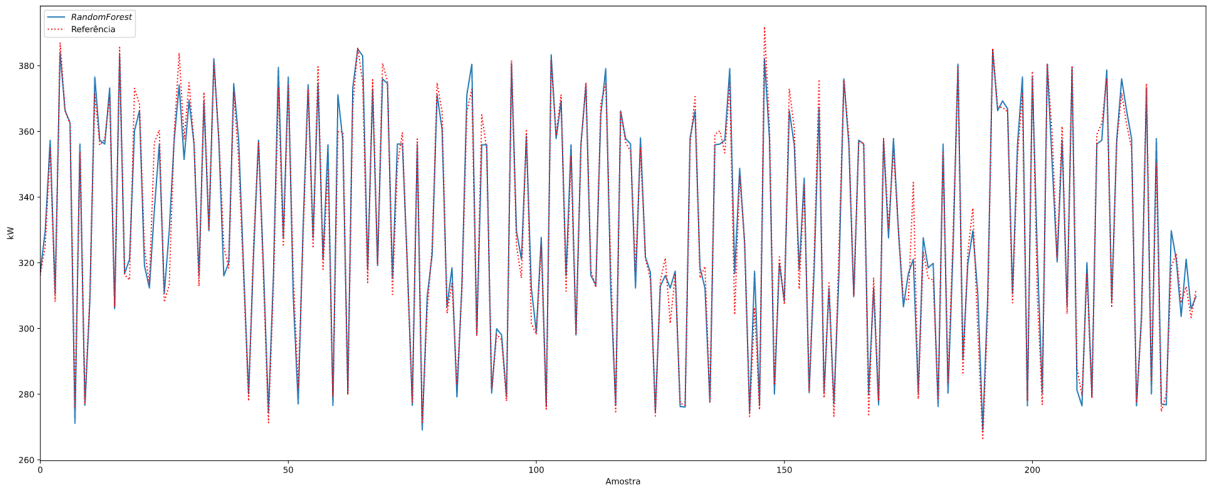
Fonte: Autoria Própria

Portanto, os resultados encontrados no processo de *tuning* replicaram conceitos encontrados na literatura e que foram revisados nos parágrafos anteriores.

Pela tabela 11 pode-se perceber a importância e aplicar o *tuning* dos hiperparâmetros ao modelo. Com um tempo de execução próximo de 40min, foi possível melhorar todos os parâmetros de desempenho encontrados após a etapa de seleção de *features*, compensando um pouco a perda evidenciada nessa etapa e reduzindo ainda mais o tempo de execução do modelo.

Para finalizar, plotou-se o resultado de um *predict* do modelo final em *Random Forest* utilizando 5% de amostras aleatórias existentes no conjunto de teste anterior, resultando em 234 amostras totais. O resultado pode ser visto na figura 30.

Figura 30: Resultado de um *predict* com o modelo final



Fonte: Autoria Própria

Analisando os resultados finais, conseguiu-se um ganho bastante relevante no tempo

de execução do modelo, de 83,8% comparando o primeiro com o último *baseline*. Apesar de penalizarmos o erro absoluto médio em 16,7%, a otimização do desempenho permitiu que esse valor não ultrapassasse os 20% arbitrado como critério, o que apresenta pouco impacto no valor final em kW. Por fim, o valor de R^2 ficou bastante próximo do encontrado no primeiro *baseline*, não impactando com severidade o desempenho global do modelo.

O modelo encontrado aqui e referenciado como terceiro *baseline*, foi usado como função *fitness* do PSO nos resultados apresentados no capítulo seguinte. Importante ressaltar que o *tuning* dos hiperparâmetros não buscou encontrar a combinação ótima global dos mesmos e sim uma combinação que melhorasse o desempenho e que não ultrapassasse os critérios limites arbitrados, funcionando como um ótimo local nesse contexto.

5 OTIMIZAÇÃO DO CONSUMO DE ENERGIA

5.1 EXECUÇÃO DO PSO

A escolha do PSO como método de otimização para o consumo de energia do moinho de rolos se deu por alguns fatores, já citados anteriormente na fundamentação teórica desse trabalho, porém, novamente evidenciados nesse capítulo.

Primeiramente, o fato de ser classificado com uma metaheurística e, por isso, não ser problema-dependente, torna o PSO mais vantajoso quando comparado a soluções matemáticas tradicionais ou a algoritmos desenvolvidos para resolver problemas específicos. A solução matemática, por ser inviável do ponto de vista da complexidade (multidimensionalidade dos espaço de busca, devido a quantidade de *features* envolvidas no modelo, levariam no melhor cenário a uma equação de grau 7, após a seleção de *features*), não foi considerada durante a elaboração desse trabalho (BOUSSAÏD et al., 2013).

Outro fator preponderante é a conhecida facilidade de implementação do PSO, que envolve apenas matemática básica na sua codificação e portanto, caracteriza-se por consumir poucos recursos computacionais na execução. Além disso, é reconhecidamente útil na resolução de problemas que envolvem não-linearidades e podem convergir a soluções desses problemas enquanto a maioria dos métodos analíticos iriam falhar (KENNEDY; EBERHART, 1995; EBERHART; KENNEDY, 1995; MARTINS et al., 2013; BANKS et al., 2007; VALLE et al., 2008).

Quando comparado a outros métodos semelhantes de resolução de problemas, possui vantagens como a pouca quantidade de parâmetros necessários para sua execução, ser dotado de memória, o que permite comparar os melhores resultados obtidos anteriormente com os resultados atuais, possui maior diversidade no enxame devido a cada uma de suas partículas ter a capacidade de utilizar informações de outras partículas para melhorar seus próprios resultados, suas partículas são todas atualizadas em paralelo e as novas velocidades e posições dependem não apenas da velocidade e posição anteriores da própria partícula, como também das de suas vizinhas e por fim, todas as suas partículas são atualizadas em relação a posição e velocidade

seguindo a mesma regra (VALLE et al., 2008).

Finalmente, a grande quantidade de implementações disponíveis na internet, em repositórios de códigos como o *Github*, além de API específicas para a execução do PSO, como o *PySwarms*, reforçam a praticidade de aplicação do método, sem a necessidade de implementar as soluções do zero.

Para a execução do PSO na etapa de otimização do consumo de energia, foram adotadas duas soluções distintas disponíveis na internet e implementadas baseadas no PSO original proposto inicialmente por (KENNEDY; EBERHART, 1995; EBERHART; KENNEDY, 1995). A primeira delas, disponível em (THIYAGARAJ, 2018) e a segunda, disponível através da API do *PySwarms* (MIRANDA, 2018)

Na implementação disponibilizada por Thiyagaraj (2018), foram alteradas algumas sessões do código buscando otimização da execução, além de adapta-lo as necessidades do problema proposto nesse trabalho. Com esse fim, as alterações na implementação foram concentradas em desenvolver uma nova função de *fitness*, utilizando assim o modelo em *Machine Learning* desenvolvido com o *Random Forest* e que foi otimizado com seleção de *features* e com *tuning* de hiperparâmetros, além de um novo critério de parada para diminuir o tempo de execução, quebrando o *loop* principal do código após sucessivas iterações apresentarem o mesmo resultado, o que demonstra convergência da solução do problema e consequentemente a possibilidade de encerrar a execução antes de todas as iterações serem concluídas. Quanto ao *PySwarms*, nenhuma alteração foi realizada no código original.

Além disso, para efeitos de comparação, foi criado um *script* em *Python* para gerar automaticamente e aleatoriamente os valores iniciais para as 30 execuções do PSO, usando uma distribuição uniforme, e esse vetor foi apresentado de forma igual as duas implementações avaliadas. Da mesma forma, ambos receberam as mesmas fronteiras para cada uma das *features* testadas, limitando o espaço de busca para cada variável independente de forma igual para ambos. Quanto aos parâmetros de execução, a mesma abordagem foi adotada. Sendo assim, apesar de eventualmente terem sido implementados de formas distintas, ambos tiveram os mesmos critérios adotados durante a execução, permitindo assim uma comparação entre seus resultados, com o objetivo de, primeiramente, estudar se as implementações realmente funcionam e posteriormente, de escolher um deles para a sequência do processo de otimização.

Para a execução, foram adotados os parâmetros listados na tabela 16. Segundo afirma Poli et al. (2007), a definição do tamanho da amostra geralmente é realizada empiricamente, assim como os valores de c1 e c2, porém com especial atenção aos dois últimos, visto que os mesmos possuem capacidade de influenciar radicalmente o comportamento do PSO. Como

ele possui características aleatórias, torna-se necessário a execução do mesmo múltiplas vezes, permitindo assim uma medição de desempenho geral. Portanto, um parâmetro com a quantidade de execuções também foi arbitrado. (MARTINS et al., 2013).

Com o objetivo de reduzir o espaço de busca do PSO, foi definido um vetor com dados de fronteira inferior e superior para cada uma das 7 *features* existentes no modelo. Essas fronteiras também foram adotadas na geração do vetor de valores iniciais e seus valores estão listados na tabela 17.

Tabela 16: Parâmetros adotados para a execução do PSO

Quantidade de execuções	30
Número de iterações	100
Tamanho do *swarm*	100
w	0,5
c1	1
c2	2

Fonte: Autoria Própria

Tabela 17: Fronteiras adotada durante a execução do PSO

Feature X	Valor Mínimo	Valor Máximo
x2	192	198
x5	355	365
x6	355	365
x7	66	81
x10	415	425
x13	60	66
x15	-65	-55

Fonte: Autoria Própria

Importante ressaltar que o processo de otimização levou em conta somente uma das receitas disponíveis para o moinho de rolos estudado. O fato de operar com 5 receitas distintas exige um processo de otimização para cada uma delas, visto que as fronteiras definidas para o espaço de busca de cada uma devem ser diferentes, considerando que às *feature*s são atribuídos valores distintos quando o equipamento está em operação com cada uma das receitas usadas. Sendo assim, optou-se por utilizar a Receita 2 pois, historicamente, ela é a que possui a maior taxa de ocupação do tempo de operação da planta de moagem, o que inclusive pode

ser evidenciado pela figura 18 onde é demonstrada a distribuição das receitas. A figura é apresentada no capítulo 3.1.

Para a definição dos valores de mínimos e máximos para a fronteira, levou-se em consideração os valores limites disponíveis dentre as 7968 amostras de Receita 2 existentes no conjunto de dados aplicado na modelagem, usando as funções *min()* e *max()* do Excel. Além disso, para algumas *features* foi acrescentado um valor extra aos limites inferior e superior, alinhados com a capacidade do equipamento em manter as mesmas características do produto final, não comprometendo assim as especificações técnicas da receita em questão.

O resultado das 30 execuções de cada uma das implementações do PSO pode ser encontrado na tabela 18. Ao resultado, foi aplicado o teste de *Mann-Whitney* para comparar as duas amostras, onde a hipótese nula H0 do teste afirma que as duas amostras podem vir de uma mesma distribuição e portanto não apresentarem diferenças estatísticas. Para o teste foi considerado nível de confiança de 95% (α = 0,05).

Tabela 18: Resultados das 30 execuções de cada uma das implementações do PSO

PySwarms	Tradicional
315,81	325,35
312,78	317,26
294,85	294,09
290,84	290,84
305,70	304,12
304,99	323,96
346,00	316,52
302,35	309,29
304,56	304,56
323,33	323,51
312,75	327,61
297,94	322,22
300,14	298,43
318,63	318,89
327,71	320,57
308,74	328,73
298,66	290,84
305,07	305,07
310,00	322,40
325,60	326,36
300,04	306,28
334,23	334,23
323,74	323,74
304,88	290,84
305,30	323,51
323,33	336,29
322,04	325,31
321,65	339,60
321,85	312,73

Média	312,32	Média	316,60
Desvio Padrão	12,67	Desvio Padrão	14,14
Tempo de Execução	8 min	Tempo de Execução	22 min

p-value = 0,070

Fonte: Autoria Própria

O resultado mostra que a hipótese nula H0 não foi rejeitada (*p-value* = 0,070), o que indica não haver diferença entre os resultados das duas amostras. Sendo assim, optou-se por utilizar a implementação do *PySwarm* na sequência do trabalho, visto que o mesmo apresentou um tempo de execução inferior ao apresentado pela implementação tradicional e consequentemente, menor custo computacional. Apesar da busca na literatura por uma explicação que pudesse satisfazer essa diferença, não foram encontradas evidencias publicadas sobre as mesmas. Acredita-se que o uso de listas na implementação tradicional seja o motivo, visto que a *PySwarms* utiliza *Python numpy-arrays*.

Portanto, considerando os resultados apresentados pela execução do *PySwarms* temos um valor médio para a potência de saída do moinho em 312,32 kW e um *GBEST*, comparando os resultados das 30 execuções, de 290,84 kW.

5.2 OTIMIZAÇÃO DO PSO

O pouco esforço requerido para parametrização do PSO torna-o reconhecido como uma das soluções mais atrativas para resolução de problemas de otimização, onde somente alguns poucos ajustes nos parâmetros são necessários, para resolver problemas em uma grande variedade de aplicações (MARTINS et al., 2013).

Porém, uma das preocupações em relação a execução do PSO é a possibilidade do mesmo convergir a um mínimo local, provocando a estagnação do enxame e consequentemente a uma solução sub-ótima. Sendo assim, Banks et al. (2007) afirma que atribuir um valor de coeficiente de inércia ω a velocidade é uma das técnicas conhecidas para resolver o problema.

Inicialmente, já havia sido arbitrado um valor de coeficiente de inércia para a execução porém, outra proposta de valor e a combinação do mesmo com alterações no tamanho do *swarm* foram realizadas. O vetor com os valores iniciais foi mantido, permitindo uma comparação justa ao alterarmos os parâmetros. As variações de parametrização podem ser vistas na tabela 19.

Tabela 19: Variação de parâmetros adotados para a execução do PSO

	Original	Variação 1	Variação 2	Variação 3	Variação 4	Variação 5
Quantidade de execuções	30	30	30	30	30	30
Tamanho do *swarm*	100	500	1000	100	500	1000
w	0,5	0,5	0,5	1	1	1
c1	1	1	1	1	1	1
c2	2	2	2	2	2	2

Fonte: Autoria Própria

O resultado das execuções das cinco variações, comparadas com a execução com os parâmetros inicialmente definidos, encontra-se na tabela 20

Tabela 20: Resultado da execução do PSO com variações nos parâmetros

	Variação Original	Variação 1	Variação 2	Variação 3	Variação 4	Variação 5
	315,81	300,57	300,57	297,79	297,79	297,99
	312,78	305,17	312,78	297,79	294,67	294,67
	294,85	290,84	290,84	290,84	290,84	290,84
	290,84	290,84	294,09	290,84	290,84	290,84
	305,70	304,12	304,12	290,84	290,84	290,84
	346,00	334,79	327,15	305,97	305,56	303,70
	302,35	302,35	302,35	294,67	290,84	290,84
	304,56	303,70	303,94	303,70	303,70	303,70
	323,33	305,30	323,33	297,79	297,79	297,79
	312,75	322,04	322,04	297,79	297,99	297,99
	297,94	297,79	297,94	297,79	297,79	297,79
	300,14	298,43	299,67	290,84	290,84	290,84
	318,63	313,24	313,24	313,24	302,27	313,24
	327,71	313,24	327,71	304,86	304,86	304,86
	308,74	308,74	323,33	297,79	297,79	297,79
	298,66	294,67	298,66	294,67	294,67	294,67
	305,07	303,70	304,99	303,70	303,70	303,70
	310,00	302,35	324,35	303,70	303,70	290,84
	325,60	305,17	305,30	297,79	297,79	297,79
	300,04	298,64	306,28	294,67	305,97	294,67
	334,23	310,58	317,52	300,62	297,79	300,61
	323,74	323,74	323,74	300,61	302,35	297,79
	304,88	304,36	297,78	290,84	303,70	290,84
	305,30	323,51	323,33	297,79	297,79	297,79
	323,33	305,30	297,79	297,79	324,73	297,79
	322,04	315,51	297,79	297,79	297,79	297,79
	321,65	321,65	312,41	305,17	305,17	305,17
	305,96	305,96	290,84	303,70	290,84	290,84
	321,85	312,70	297,79	297,79	297,79	297,79
Média	312,32	307,47	308,82	298,76	299,40	297,52
Desvio Padrão	12,56	10,08	11,94	5,35	6,90	5,54

Fonte: Autoria Própria

A essas amostras, foi aplicado o teste estatístico de *Shapiro-Wilk*, com o objetivo de avaliar a normalidade de cada uma delas e então compará-las em conjunto. A hipótese nula H0 afirma que a amostra testada é proveniente de uma distribuição normal, para *p-value* $< 0,05$. A tabela 21 apresenta o resultado do teste de normalidade.

Tabela 21: Resultado do teste de *Shapiro-Wilk* para as 6 amostras, com α = 0,05

	p-value	Interpretação
Variação Original	0,238	Falha em Rejeitar H0
Variação 1	0,101	Falha em Rejeitar H0
Variação 2	0,009	Rejeita H0
Variação 3	0,030	Rejeita H0
Variação 4	0,000	Rejeita H0
Variação 5	0,005	Rejeita H0

Fonte: Autoria Própria

Das seis amostras testadas, a Amostra Original e a Variação 1 possuem distribuição normal, enquanto as amostras Variação 2 a 5 não possuem. Analisando a tabela 20 com os resultados de média e desvio padrão das 6 amostras, percebe-se que a alteração de parâmetros promovida a partir da Variação 3, quando se altera o valor do coeficiente de inércia ω, leva a uma diminuição do desvio padrão e da dispersão da amostra. Essa diminuição já justificaria o fato das amostras referentes a essa variações de parâmetro não terem apresentado normalidade no teste de *Shapiro-Wilk*. A diminuição das médias, nas mesmas amostras, levam a crer que a execução do PSO convergiu melhor quando comparada com as outras três execuções restantes. Quanto a amostra Variação 2, apesar de não ter sido afetada pela alteração do coeficiente de inércia, sofre forte influência na alteração do tamanho do *swarm* e isso pode ter levado a um resultado de não-normalidade no seu teste.

Podemos afirmar também que todas as amostras convergiram para um *GBEST* de 290,84 em pelo menos uma das 30 execuções. Esse valor aparenta ser o *GBEST* possível dentro das fronteiras definidas para execução do PSO.

Para definir a parametrização ótima para a execução do PSO, baseado no resultado do teste de *Shapiro-Wilk*, optou-se pela aplicação do teste não-paramétrico de *Kruskal-Wallis*, buscando avaliar se as 6 amostras se originam da mesma distribuição, o que nos levaria a optar pela de menor média para escolha da melhor. A hipótese nula H0 desse teste afirma que as amostras são originadas de uma mesma distribuição caso *p-value* $< 0,05$. O resultado do teste está na tabela 22

Tabela 22: Resultado do teste de *Kruskal-Wallis* para as 6 amostras, com α = 0,05

	p-value	Interpretação
Comparação das 6 amostras	0,000	Rejeita H0

Fonte: Autoria Própria

O resultado nos indica que a hipótese nula H0 foi rejeitada e portanto, que as amostras não são originadas da mesma distribuição. Nesse caso, adotou-se outro teste estatístico, o teste *post-hoc* de *Dunn-Sidak*, que faz comparações múltiplas entre as amostras, par a par, buscando analisar se as mesmas são estatisticamente diferentes ou não. A hipótese nula H0 para o teste indica que os pares de amostras são diferentes para $p\text{-}value < 0,05$.

Tabela 23: Resultado do teste de *Dunn-Sidak* para as 6 amostras, com α = 0,05

	Variação Original	Variação 1	Variação 2	Variação 3	Variação 4	Variação 5
Variação Original	1,000000	0,984269	0,964025	0,000025	0,000065	0,000001
Variação 1	0,984269	1,000000	1,000000	0,004415	0,009202	0,000298
Variação 2	0,964025	1,000000	1,000000	0,006820	0,013904	0,000494
Variação 3	0,000025	0,004415	0,006820	1,000000	1,000000	0,999982
Variação 4	0,000065	0,009202	0,013904	1,000000	1,000000	0,999527
Variação 5	0,000001	0,000298	0,000494	0,999982	0,999520	1,000000

Fonte: Autoria Própria

Analisando a tabela 23 com os resultados do teste de *Dunn-Sidak*, pode-se afirmar a existência de amostras estatisticamente iguais, onde os valores de *p-value* ficaram próximos ou iguais a 1,0. Nesse caso, assim como na análise feita com o teste de *Shapiro-Wilk*, a variação do parâmetro ω parece influenciar na formação da amostra, diferenciando-as daquelas que tiveram seu valor mantido conforme a Amostra Original.

Sendo assim, analisando as amostras com diferença estatística e tomando como parâmetro de decisão a menor média entre elas, a Variação 5 dos parâmetros do PSO é a escolhida.

Diante dessa afirmação e considerando o *GBEST* de 290,84kW, a tabela 24 traz os valores atribuídos a cada uma das *features* na execução do PSO usando o modelo *randon forest* como *fitness*. Esses são então os valores dos parâmetros que devem ser atribuídos fisicamente ao moinho em estudo, para que o mesmo obtenha um desempenho de consumo de energia

otimizado, dentro dos critérios estabelecidos nesse trabalho.

Tabela 24: Valores atribuídos a cada uma das *features* no melhor *GBEST* encontrado

Feature X	Valor
x2	195,35
x5	359,77
x6	365
x7	67,79
x10	420,45
x13	60
x15	-55,78

Fonte: Autoria Própria

6 CONCLUSÃO

Este trabalho propôs a utilização da técnicas *surrogate* baseada em *Random Forest* para a obtenção de um modelo do consumo de energia de um moinho do tipo *Raymond*, utilizado para moagem de carbonato de cálcio natural. A técnica de *Random Forest* se mostrou bastante eficiente durante as três etapas de modelagem. Na primeira etapa, quando foram apresentadas todas as *features* amostradas do equipamento em campo, essa técnica de *Machine Learning* foi capaz de modelar o problema com médias de 5,33 segundos por *sampling*, R^2 de 0,982 e com um erro absoluto médio de 3,3kW, que representa apenas 1,02% da mediana de todos os valores de potência registrados no conjunto de dados. Além disso, mostrou-se muito flexível ao entregar um conjunto de dados de validação, chamado de *out-of-bag*, permitindo que não fosse empregado validação cruzada durante a fase de aprendizagem, o que diminui bastante o custo computacional para obtenção do modelo.

Com o objetivo de reduzir a dimensionalidade do conjunto de dados, na etapa de seleção de *features*, foram aplicadas 5 técnicas computacionais, a citar *MIFS*, *K-BEST*, *RF-RFE*, *CFS*, *Stability Selection* e uma técnica manual, realizada por um Engenheiro de Processo. O resultado demonstrado pelo método *K-BEST* permitiu a eliminação de um total de 10 *features*, reduzindo o tempo de execução em 82,2% quando comparado com o primeiro modelo obtido com todas as 17 *features*. Nessa etapa, apesar do bom resultado em otimizar o tempo de execução do processo de aprendizagem em *Random Forest*, optou-se por penalizar em até 1% o resultado de R^2 e em até 20% o resultado do erro absoluto médio, pois esses valores não comprometem o resultado final da modelagem e permitem maximizar a otimização de tempo. Nesse processo, o método manual obteve bons resultados quando comparados aos 5 métodos computacionais, evidenciando a importância de se conhecer o processo e a característica dos dados modelados, quando *Machine Learning* for usado para modelar um problema.

Seguindo com o processo de otimização do modelo, o *Grid Search* foi usado como técnica computacional para testar 705 combinações distintas de hiperparâmetros para o *Random Forest*, concentrando as buscas em apenas 4 dos mais de 10 hiperparâmetros encontrados na API do *Scikit Learn*, disponível para *Python*. O resultado do *tuning* permitiu melhorar em 4,4%

o tempo de execução obtido na etapa anterior de seleção de *features*, além de reduzir o erro absoluto médio em 0,2% quando comparado com a mesma etapa.

Por fim, o processo de otimização realizado com o PSO encontrou resultados muito satisfatórios no que tange ao consumo de energia do moinho. Comparado com a média de potência dissipada durante a produção da Receita 2 do moinho, aplicar essa metaheurística permitiu um ganho de aproximadamente 9%, o que representa uma potência instantânea de 290,84kW versus 317,91kW atualmente.

No processo de otimização da execução do PSO, foram arbitrados diferentes valores para o coeficiente de inércia e para o tamanho do enxame, o que trouxe uma melhoria significativa na média geral das 30 execuções realizadas para cada variação testada. Essas variações de parâmetros não mudaram o *GBEST* encontrado com a técnica, dentro das fronteiras definidas para o espaço de busca, e que viabilizou o processo de convergência sem comprometer os parâmetros introduzidos no moinho para a produção da Receita 2, garantindo assim que o produto final ainda esteja dentro da especificação de qualidade.

Portanto, o estudo apresentado nessa dissertação se mostrou viável, já que demonstrou resultados plausíveis que permitem uma redução significativa do consumo de energia de moinhos de rolos do tipo *Raymond*.

6.1 TRABALHOS FUTUROS

- Avaliar outras técnicas de *Machine Learning* como *Support Vector Machine Regression* ou uma Rede Neural do tipo RBF, na obtenção de um modelo de consumo de energia para os moinhos usados nesse trabalho;
- Avaliar outras técnicas de *tuning* para o *Random Forest*, que permitam testar outras combinações de hiperparâmetros e que não tenham o alto custo computacional do método de *Grid Search*;
- Aplicar a metodologia para otimizar outros recursos de produção de moinhos de rolos do tipo Raymond que busquem, por exemplo, aumentar a produtividade do moinho sem comprometer a qualidade do produto final ou a vida útil do equipamento;
- Aplicar a metodologia a outros tipos de moinhos industriais, como moinhos de bolas, moinhos de anéis, moinhos autógenos, buscando modelar e inclusive simular esses equipamentos;

REFERÊNCIAS

AGGARWAL, C. C. **Data mining: the textbook**. [S.l.]: Springer, 2015.

ALBERT, L. A. **Efficient genetic algorithms using discretization scheduling**. Tese (Doutorado) — Citeseer, 2001.

ARLOT, S.; CELISSE, A. et al. A survey of cross-validation procedures for model selection. **Statistics surveys**, The author, under a Creative Commons Attribution License, v. 4, p. 40–79, 2010.

BANKS, A.; VINCENT, J.; ANYAKOHA, C. A review of particle swarm optimization. part i: background and development. **Natural Computing**, Springer, v. 6, n. 4, p. 467–484, 2007.

BATTITI, R. Using mutual information for selecting features in supervised neural net learning. **IEEE Transactions on neural networks**, IEEE, v. 5, n. 4, p. 537–550, 1994.

BERK, R. A. **Statistical learning from a regression perspective**. [S.l.]: Springer, 2008.

BIAU, G.; SCORNET, E. A random forest guided tour. **Test**, Springer, v. 25, n. 2, p. 197–227, 2016.

BLUM, A.; KALAI, A.; LANGFORD, J. Beating the hold-out: Bounds for k-fold and progressive cross-validation. In: **Proceedings of the twelfth annual conference on Computational learning theory**. [S.l.: s.n.], 1999. p. 203–208.

BONYADI, M. R.; MICHALEWICZ, Z. **Particle swarm optimization for single objective continuous space problems: a review**. [S.l.]: MIT Press, 2017.

BOULESTEIX, A.-L. et al. Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics. **Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery**, Wiley Online Library, v. 2, n. 6, p. 493–507, 2012.

BOUSSAÏD, I.; LEPAGNOT, J.; SIARRY, P. A survey on optimization metaheuristics. **Information sciences**, Elsevier, v. 237, p. 82–117, 2013.

BRAMER, M. **Principles of data mining**. [S.l.]: Springer, 2007.

BREIMAN, L. Bagging predictors. **Machine learning**, Springer, v. 24, n. 2, p. 123–140, 1996.

BREIMAN, L. Random forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001.

BREIMAN, L. Manual on setting up, using, and understanding random forests v3. 1. **Statistics Department University of California Berkeley, CA, USA**, v. 1, p. 58, 2002.

BREIMAN, L. et al. **Classification and regression trees**. [S.l.]: CRC press, 1984.

BUITINCK, L. et al. API design for machine learning software: experiences from the scikit-learn project. In: **ECML PKDD Workshop: Languages for Data Mining and Machine Learning**. [S.l.: s.n.], 2013. p. 108–122.

CAI, J. et al. Feature selection in machine learning: A new perspective. **Neurocomputing**, Elsevier, v. 300, p. 70–79, 2018.

CHAOYANG, M. **Raymond Mill**. 2020. Disponível em: <http://chaozhongchina.com>.

CNI. Indicador de custos industriais 2019. **CNI**, 2020.

COURONNÉ, R.; PROBST, P.; BOULESTEIX, A.-L. Random forest versus logistic regression: a large-scale benchmark experiment. **BMC bioinformatics**, Springer, v. 19, n. 1, p. 270, 2018.

DASARI, S. K.; CHEDDAD, A.; ANDERSSON, P. Random forest surrogate models to support design space exploration in aerospace use-case. In: SPRINGER. **IFIP International Conference on Artificial Intelligence Applications and Innovations**. [S.l.], 2019. p. 532–544.

DÍAZ-URIARTE, R.; ANDRES, S. A. D. Gene selection and classification of microarray data using random forest. **BMC bioinformatics**, Springer, v. 7, n. 1, p. 3, 2006.

EBERHART, R.; KENNEDY, J. A new optimizer using particle swarm theory. In: IEEE. **MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science**. [S.l.], 1995. p. 39–43.

ENERGÉTICA, E. E. D. P. Anuário estatístico de energia elétrica 2019: ano base 2018. **Rio de Janeiro: EPE**, 2019.

ENERGIA, M. de Minas e. Boletim mensal de monitoramento do sistema elétrico brasileiro. **Empresa de Pesquisa Energética. Brasília: MME/EPE**, 2020.

ENERGIA, P. D. d. E. de et al. Ministério de minas e energia. **Empresa de Pesquisa Energética. Brasília: MME/EPE**, 2020.

ERTEL, W. **Introduction to artificial intelligence**. [S.l.]: Springer, 2017.

FANELLI, G. et al. Random forests for real time 3d face analysis. **International journal of computer vision**, Springer, v. 101, n. 3, p. 437–458, 2013.

FIGUEIRA, H. V.; LUZ, A. B. d.; ALMEIDA, S. L. M. d. Britagem e moagem. In: . [S.l.]: CETEM/MCT, 2010.

FISHER, R. A. et al. 014: On the"probable error"of a coefficient of correlation deduced from a small sample. 1921.

GAO, D.; ZHANG, Y.-X.; ZHAO, Y.-H. Random forest algorithm for classification of multiwavelength data. **Research in Astronomy and Astrophysics**, IOP Publishing, v. 9, n. 2, p. 220, 2009.

GENUER, R.; POGGI, J.-M.; TULEAU-MALOT, C. Variable selection using random forests. **Pattern recognition letters**, Elsevier, v. 31, n. 14, p. 2225–2236, 2010.

GRANITTO, P. M. et al. Recursive feature elimination with random forest for ptr-ms analysis of agroindustrial products. **Chemometrics and Intelligent Laboratory Systems**, Elsevier, v. 83, n. 2, p. 83–90, 2006.

GYPMAK. **Raymond Mill**. 2020. Disponível em: <http://www.gypmak.com/plasterboard-plant-machinery-manufacturer/sarkac-toplu-degirmen-ureticisi-raymond-roller-mill-manufacturer>.

HALL, M. A. Correlation-based feature selection for machine learning. Citeseer, 1999.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The elements of statistical learning: data mining, inference, and prediction**. [S.l.]: Springer Science & Business Media, 2009.

HAUSCHILD, M.; PELIKAN, M. An introduction and survey of estimation of distribution algorithms. **Swarm and evolutionary computation**, Elsevier, v. 1, n. 3, p. 111–128, 2011.

HENAN, M. **Raymond Mill**. 2020. Disponível em: <https://sidorsokolow.pl/products/prodetail10.html>.

HEUMANN, C.; SCHOMAKER, M. et al. **Introduction to statistics and data analysis**. [S.l.]: Springer, 2016.

HUANG, B. F.; BOUTROS, P. C. The parameter sensitivity of random forests. **BMC bioinformatics**, Springer, v. 17, n. 1, p. 331, 2016.

HUANG, J.; LI, Y.-F.; XIE, M. An empirical analysis of data preprocessing for machine learning-based software cost estimation. **Information and software Technology**, Elsevier, v. 67, p. 108–127, 2015.

IEA. Global energy review 2020. **IEA, Paris**, 2020.

IEA. Investment estimates for 2020 continue to point to a record slump in spending. **IEA, Paris**, 2020.

IEA. Paving the way to recovery with utility-funded energy efficiency. **IEA, Paris**, 2020.

IGUAL, L.; SEGUÍ, S. **Introduction to Data Science**. [S.l.]: Springer, 2017.

JIN, Y. Surrogate-assisted evolutionary computation: Recent advances and future challenges. **Swarm and Evolutionary Computation**, Elsevier, v. 1, n. 2, p. 61–70, 2011.

JUNIOR, H. D. Cominuição. In: . [S.l.: s.n.], 2013.

KANT, G.; SANGWAN, K. S. Predictive modelling for energy consumption in machining using artificial neural network. **Procedia Cirp**, Elsevier, v. 37, p. 205–210, 2015.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: IEEE. **Proceedings of ICNN'95-International Conference on Neural Networks**. [S.l.], 1995. v. 4, p. 1942–1948.

KOTSIANTIS, S.; KANELLOPOULOS, D.; PINTELAS, P. Data preprocessing for supervised leaning. **International Journal of Computer Science**, Citeseer, v. 1, n. 2, p. 111–117, 2006.

KUBAT, M. **An introduction to machine learning**. [S.l.]: Springer, 2017.

KUHN, M.; JOHNSON, K. **Applied predictive modeling**. [S.l.]: Springer, 2013.

LIAW, A.; WIENER, M. et al. Classification and regression by randomforest. **R news**, v. 2, n. 3, p. 18–22, 2002.

MARTINS, M. S. et al. Pso with path relinking for resource allocation using simulation optimization. **Computers & Industrial Engineering**, Elsevier, v. 65, n. 2, p. 322–330, 2013.

MCKINNEY Wes. Data Structures for Statistical Computing in Python. In: WALT Stéfan van der; MILLMAN Jarrod (Ed.). **Proceedings of the 9th Python in Science Conference**. [S.l.: s.n.], 2010. p. 56 – 61.

MEINSHAUSEN, N.; BÜHLMANN, P. Stability selection. **Journal of the Royal Statistical Society: Series B (Statistical Methodology)**, Wiley Online Library, v. 72, n. 4, p. 417–473, 2010.

MIRANDA, L. J. Pyswarms: a research toolkit for particle swarm optimization in python. **Journal of Open Source Software**, The Open Journal, v. 3, n. 21, p. 433, 2018. Disponível em: <https://doi.org/10.21105/joss.00433>.

MOON, J. et al. A short-term electric load forecasting scheme using 2-stage predictive analytics. In: IEEE. **2018 IEEE International Conference on Big Data and Smart Computing (BigComp)**. [S.l.], 2018. p. 219–226.

MUTANGA, O.; ADAM, E.; CHO, M. A. High density biomass estimation for wetland vegetation using worldview-2 imagery and random forest regression algorithm. **International Journal of Applied Earth Observation and Geoinformation**, Elsevier, v. 18, p. 399–406, 2012.

ONG, Y. S.; NAIR, P. B.; KEANE, A. J. Evolutionary optimization of computationally expensive problems via surrogate modeling. **AIAA journal**, v. 41, n. 4, p. 687–696, 2003.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PELIKAN, M.; SASTRY, K. Fitness inheritance in the bayesian optimization algorithm. In: SPRINGER. **Genetic and Evolutionary Computation Conference**. [S.l.], 2004. p. 48–59.

POLI, R. Analysis of the publications on the applications of particle swarm optimisation. **Journal of Artificial Evolution and Applications**, Hindawi, v. 2008, 2008.

POLI, R.; KENNEDY, J.; BLACKWELL, T. Particle swarm optimization. **Swarm intelligence**, Springer, v. 1, n. 1, p. 33–57, 2007.

PRASAD, A. M.; IVERSON, L. R.; LIAW, A. Newer classification and regression tree techniques: bagging and random forests for ecological prediction. **Ecosystems**, Springer, v. 9, n. 2, p. 181–199, 2006.

PROBST, P.; WRIGHT, M. N.; BOULESTEIX, A.-L. Hyperparameters and tuning strategies for random forest. **Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery**, Wiley Online Library, v. 9, n. 3, p. e1301, 2019.

QIN, Z. et al. Application of machine learning methodologies for predicting corn economic optimal nitrogen rate. **Agronomy Journal**, The American Society of Agronomy, Inc., v. 110, n. 6, p. 2596–2607, 2018.

QUEIPO, N. V. et al. Surrogate-based analysis and optimization. **Progress in aerospace sciences**, Elsevier, v. 41, n. 1, p. 1–28, 2005.

REITERMANOVA, Z. Data splitting. In: **WDS**. [S.l.: s.n.], 2010. v. 10, p. 31–36.

SALCEDO-SANZ, S. et al. Feature selection in machine learning prediction systems for renewable energy applications. **Renewable and Sustainable Energy Reviews**, Elsevier, v. 90, p. 728–741, 2018.

SAMPAIO, J. A.; ALMEIDA, S. L. M. d. Calcário e dolomito. In: . [S.l.]: CETEM/MCT, 2005.

SANTANA, P. H. de M.; BAJAY, S. V. New approaches for improving energy efficiency in the brazilian industry. **Energy reports**, Elsevier, v. 2, p. 62–66, 2016.

SASTRY, K. **Evaluation-relaxation schemes for genetic and evolutionary algorithms**. Tese (Doutorado) — Citeseer, 2001.

SASTRY, K.; GOLDBERG, D. E.; PELIKAN, M. Don't evaluate, inherit. In: SAN FRANCISCO, CALIFORNIA, USA: MORGAN KAUFMANN. **Proceedings of the Genetic and Evolutionary Computation Conference**. [S.l.], 2001. p. 551–558.

SASTRY, K.; PELIKAN, M.; GOLDBERG, D. E. Efficiency enhancement of genetic algorithms via building-block-wise fitness estimation. In: IEEE. **Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)**. [S.l.], 2004. v. 1, p. 720–727.

SBMGROUP. **Raymond Mill**. 2020. Disponível em: <http://www.sbm-mill.com>.

SERAPIÃO, A. B. d. S. Fundamentos de otimização por inteligência de enxames: uma visão geral. **Sba: Controle & Automação Sociedade Brasileira de Automatica**, SciELO Brasil, v. 20, n. 3, p. 271–304, 2009.

SHAO, J. Linear model selection by cross-validation. **Journal of the American statistical Association**, Taylor & Francis Group, v. 88, n. 422, p. 486–494, 1993.

SKIENA, S. S. **The data science design manual**. [S.l.]: Springer, 2017.

SMARRA, F. et al. Data-driven model predictive control using random forests for building energy optimization and climate control. **Applied energy**, Elsevier, v. 226, p. 1252–1272, 2018.

SMITH, R. E.; DIKE, B. A.; STEGMANN, S. Fitness inheritance in genetic algorithms. In: **Proceedings of the 1995 ACM symposium on Applied computing**. [S.l.: s.n.], 1995. p. 345–350.

SUN, C. et al. Surrogate-assisted cooperative swarm optimization of high-dimensional expensive problems. **IEEE Transactions on Evolutionary Computation**, IEEE, v. 21, n. 4, p. 644–660, 2017.

THIYAGARAJ, T. **Particle-Swarm-Optimisation-Demo**. [S.l.]: GitHub, 2018. https://github.com/tstreamDOTh/Particle-Swarm-Optimisation-Demo.

TOHRY, A. et al. Power-draw prediction by random forest based on operating parameters for an industrial ball mill. **Advanced Powder Technology**, Elsevier, 2019.

TORRES-BARRÁN, A.; ALONSO, Á.; DORRONSORO, J. R. Regression tree ensembles for wind energy and solar radiation prediction. **Neurocomputing**, Elsevier, v. 326, p. 151–160, 2019.

VALLE, Y. D. et al. Particle swarm optimization: basic concepts, variants and applications in power systems. **IEEE Transactions on evolutionary computation**, IEEE, v. 12, n. 2, p. 171–195, 2008.

WANG, Z. et al. Random forest based hourly building energy prediction. **Energy and Buildings**, Elsevier, v. 171, p. 11–25, 2018.

WHEELDON, M.; GALK, J.; WIRTH, K.-E. Investigation of the comminution process in pendular roller mills. **International Journal of Mineral Processing**, Elsevier, v. 136, p. 26–31, 2015.

XINGYANG, M. **Raymond Mill**. 2020. Disponível em: <http://www.ore-processing.com/>.

XUE, Y. **Compare RandomForest models in R and in Python. It seems there is some difference in model results**. 2015.

YADAV, S.; SHUKLA, S. Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification. In: IEEE. **2016 IEEE 6th International conference on advanced computing (IACC)**. [S.l.], 2016. p. 78–83.

ZENIT, M. **Raymond Mill**. 2020. Disponível em: <https://www.zenith-mills.com/>.