

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA QUÍMICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA QUÍMICA

**ESTUDOS SOBRE A APLICAÇÃO DE AUTOENCODER PARA
CONSTRUÇÃO DE INFERÊNCIAS NA INDÚSTRIA QUÍMICA**

DISSERTAÇÃO DE MESTRADO

Henrique Binotto Menegolla

Porto Alegre

2019

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA QUÍMICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA QUÍMICA

***Estudos sobre a Aplicação de Autoencoder para
Construção de Inferências na Indústria Química***

Henrique Binotto Menegolla

Dissertação de Mestrado apresentada como
requisito parcial para obtenção do título de
Mestre em Engenharia

Área de concentração: Pesquisa e
Desenvolvimento de Processos

Linha de Pesquisa: Engenharia de Sistemas –
Projeto, Simulação, Modelagem, Controle e
Otimização de Processos

Orientadores:

Prof. Dr. Jorge Otávio Trierweiler

Prof. Dr. Marcelo Farenzena


Porto Alegre

2019


UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA QUÍMICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA QUÍMICA

A Comissão Examinadora, abaixo assinada, aprova a Dissertação *Estudos sobre a Aplicação de Autoencoder para Construção de Inferências na Indústria Química*, elaborada por Henrique Binotto Menegolla, como requisito parcial para obtenção do Grau de Mestre em Engenharia.

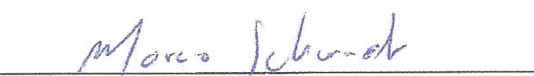
Comissão Examinadora:



Prof. Dr. Maurício Bezeira de Souza Júnior – EQ / UFRJ



Profa. Dra. Viviane Rodrigues Botelho -- UFCSPA




Prof. Dr. Márcio Schwaab – DEQUI / UFRGS


UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA QUÍMICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA QUÍMICA

A Comissão Examinadora, abaixo assinada, aprova a Dissertação *Estudos sobre a Aplicação de Autoencoder para Construção de Inferências na Indústria Química*, elaborada por Henrique Binotto Menegolla, como requisito parcial para obtenção do Grau de Mestre em Engenharia.

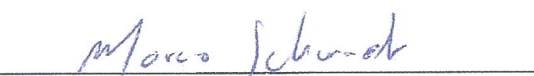
Comissão Examinadora:



Prof. Dr. Maurício Bezeira de Souza Júnior – EQ / UFRJ



Profa. Dra. Viviane Rodrigues Botelho -- UFCSPA





Prof. Dr. Márcio Schwaab – DEQUI / UFRGS

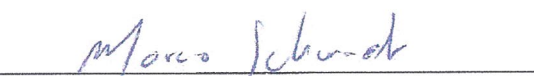
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA QUÍMICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA QUÍMICA

A Comissão Examinadora, abaixo assinada, aprova a Dissertação *Estudos sobre a Aplicação de Autoencoder para Construção de Inferências na Indústria Química*, elaborada por Henrique Binotto Menegolla, como requisito parcial para obtenção do Grau de Mestre em Engenharia.

Comissão Examinadora:


Prof. Dr. Maurício Bezerra de Souza Júnior – EQ / UFRJ


Profa. Dra. Viviane Rodrigues Botelho -- UFCSPA


Prof. Dr. Márcio Schwaab – DEQUI / UFRGS

Resumo

Na indústria moderna os processos são otimizados para uma produção mais segura, mais limpa e mais eficiente em termos energéticos. Para isso, sistemas avançados de monitoramento e controle vêm ganhando destaque nas fábricas e refinarias de petróleo. No entanto, processos industriais enfrentam problemas na medição de algumas variáveis, como por exemplo, a qualidade dos produtos, concentração dos componentes. O uso de analisadores em linha ou de medições laboratoriais não viabiliza o controle direto, por conta do tempo de amostragem e incerteza das medições dos analisadores. Para contornar esse problema e gerar informações frequentes e confiáveis, este trabalho faz um estudo de *autoencoder*, uma ferramenta baseada em redes neurais, que ainda não foi estudada visando o desenvolvimento e a manutenção de inferências. Este trabalho apresenta algumas técnicas de *machine learning* visando o tratamento prévio dos dados e em seguida, é feito o treinamento de uma rede neural de aprendizagem não-supervisionada que através da compressão e descompressão das entradas, chamada de *autoencoder*, que cria um espaço latente de menor dimensão capaz de concentrar as informações contidas nos dados empregados no desenvolvimento das inferências. A metodologia proposta para o desenvolvimento das inferências é feita a partir deste espaço reduzido pelo *autoencoder*. O espaço latente do *autoencoder* pode ser visto como a generalização dos componentes principais obtidos através da metodologia PCA. Desta forma, também são desenvolvidas inferências utilizando a metodologia PCA para comparação entre a mais utilizada atualmente e o objeto de estudo deste trabalho. A primeira etapa, é feito o pré-processamento dos dados. Posteriormente os dados são separados em conjuntos de treino, validação e teste, utilizando a metodologia *k-rank*. Em seguida os modelos são construídos através de regressões lineares com métodos (*Ridge e Lasso, Lars*) que realizam a seleção de variáveis, impondo restrições às variáveis desnecessárias aos modelos, os quais são validados utilizando diversas métricas de avaliação. Essa metodologia é testada com dois estudos de caso. Um com dados gerados artificialmente de modo a se ter uma base de dados com relações entre variáveis conhecida. E outro com dados de uma simulação em *Aspen Plus* de uma unidade de separação de propeno/propano. Essa unidade tem o objetivo de produzir propeno com pureza de 99,6%, a partir de uma carga de GLP. A qual é composta por três colunas de destilação. Na primeira coluna, retira-se os compostos pesados pelo fundo e a corrente de topo segue para a segunda coluna que tem o objetivo de retirar o etano da corrente. Com isso, a corrente de fundo dessa coluna segue para uma terceira coluna, a qual separa o propeno do propano. As informações úteis para ajudar no controle da unidade são: concentração de pesados no topo da primeira coluna para que possa ser reduzida a quantidade dessa impureza; na segunda coluna é importante reduzir o propeno que escapa junto com o etano pelo topo da coluna, para aumentar a produção final da planta; e na terceira coluna é importante manter a corrente de topo dentro da especificação, com isso é necessário estimar a impureza de propano no topo da coluna. Os resultados obtidos não conseguem apontar para uma clara superioridade na qualidade das inferências geradas em relação ao método com PCA dentro dos casos estudados com a metodologia proposta.

Abstract

In the modern industry, processes are constantly being optimized to seek safer, cleaner and more energy-efficient production. To this end, advanced monitoring and control systems have been gaining prominence in chemical factories and refineries. However, industrial processes face problems in the measurement of some variables, such as product quality and component concentration. The use of in-line analyzers or laboratory measurements does not allow direct control, due to the sampling time and uncertainty of the analyzer measurements. To circumvent this problem and finally generate frequent and reliable information, this work studies the autoencoder, a tool based on neural networks not yet applied on the development and maintenance of inferences. Some techniques of machine learning will be presented and used, as pretreatment of data following the training of a neural network of unsupervised learning through the compression and decompression of the input information, called autoencoder, produces a latent space with lower dimension concentrating the information within the data and therefore being capable of developing soft-sensors. The proposed methodology for the development of the soft-sensors is to use the reduced space by the autoencoder to predict the desired variables of the system. The latent space of the autoencoder can be seen as a generalization of the principal components of the PCA methodology. Then, a comparison of the results obtained with PCA, the current standard and the autoencoder is made to evaluate the object of this work. The first step is the pre-processing of the data. Subsequently the data are separated into calibration and test sets using the k-rank methodology. Then the models are constructed through linear regression with methods (Ridge and Lasso-Lars) that perform the selection of variables, discarding unnecessary variables to the models, which will be validated using several evaluation metrics. This methodology is tested with two case studies. One with artificial data generated with known relation between its variables. And one with data from an Aspen simulation of a propene/propane separation unit. This unit has the objective of producing propene with purity of 99.6%, from a load of GLP. It is composed of three columns of distillation. In the first column, the heavy compounds are withdrawn from the bottom and the top stream goes to the second column which has the purpose of removing the ethane from the stream. Thus, the bottom stream of this column goes to a third column, which separates the propene from the propane. The useful information to help control the unit are: concentration of heavy at the top of the first column so that the amount of this impurity can be reduced; in the second column it is important to reduce propene that escapes along with ethane at the top of the column to increase the final production of the plant; and in the third column it is important to maintain the top current within the specification, whereby it is necessary to estimate the impurity of propane at the top of the column. The results show that at this point there is no clear superiority in the quality of the predictions when comparing to the ones of the PCA method for the case-studies and the proposed methodology.

"Given the pace of technology, I propose we leave the math to the machines and go play outside"

-Calvin by Bill Waterson.

Agradecimentos

À minha família pelo inesgotável apoio ao longo de toda minha vida. Aos meus pais, Gilmar e Eneide, por todas as orientações e conselhos de vida. Ao meu irmão Arthur pela parceria e companheirismo de sempre.

Aos meus orientadores, Jorge e Marcelo, pela idealização, apoio e principalmente pela insistência durante a elaboração deste trabalho.

Aos meus amigos e meus colegas do GIMSCOP pelos bons momentos e companhia no dia a dia.

Ao Programa de Pós-Graduação em Engenharia Química da UFRGS, a CAPES e a FEENG pelo auxílio durante o desenvolvimento desta dissertação.

E por fim, a todos os professores que participaram direta ou indiretamente da minha formação acadêmica por toda dedicação e conhecimento compartilhado.

SUMÁRIO

Capítulo 1 – Introdução	1
1.1 Motivação.....	1
1.2 Inferências	2
1.3 Autoencoder.....	3
1.4 Objetivo do trabalho	5
1.5 Estrutura da dissertação	6
Capítulo 2 – Fundamentação Teórica e Revisão Bibliográfica	7
2.1 Redes Neurais.....	7
2.1.1 Terminologia e Arquitetura	7
2.1.2 Funções de Ativação	9
2.1.3 Otimizadores	11
2.1.4 Treinamento da rede	12
2.1.5 Ferramentas computacionais	13
2.2 Autoencoder.....	14
2.3 Inferências	16
2.3.1 Base de dados.....	17
1. Dados faltantes	17
2. Dados espúrios	17
3. Dados enviesados	17
4. Dados correlacionados	18
5. Taxas de amostragem e atraso em medições	18
2.3.2 Segregação dos dados	19
2.3.3 Modelagem da inferência.....	19
2.4 Critérios de Avaliação de Modelos.....	21
2.4.1 Erros de reconstrução do autoencoder: RMSE e MSLE	21
2.4.2 Erros de predição da inferência gerada: R² Ajustado, AIC e BIC.....	21
Capítulo 3 – Metodologia Proposta	23
3.1 Análise da Dimensionalidade do Problema	23
3.2 Pré-processamento dos Dados	24
3.3 Treinamento do Autoencoder.....	25
3.4 Desenvolvimento das Inferências	26
Capítulo 4 – Estudos de Caso	27
4.1 Unidade de Separação de Propeno/Propano	27
4.1.1 Modelagem da Unidade	31
4.2 Bases de Dados Geradas Artificialmente	35
4.2.1 Bases de Dados LL.....	35
4.2.2 Bases de Dados NL.....	35
4.2.3 Bases de Dados LN.....	36
4.2.4 Bases de Dados NN.....	36
Capítulo 5 – Resultados e Discussão.....	37
5.1 Coluna T-01	37
5.1.1 Pré-Processamento dos Dados da Coluna T-01	38
5.1.2 Treinando o Autoencoder para a Coluna T-01	40
5.1.3 Inferindo a Concentração dos componentes pesados (C4+) na Corrente de Topo da Coluna T-01.....	43

5.1.4	Efeito do ruído na inferência gerada a partir do espa latente	46
5.2	Coluna T-02	47
5.2.1	Pré-Processamento dos Dados da Coluna T-02	48
5.2.2	Treinando o Autoencoder para a Coluna T-02	49
5.2.3	Inferindo a Concentração de propano (C3+) na Corrente de Topo da Coluna T-02.....	50
5.2.4	Efeito do ruído na inferência gerada a partir do encodado	52
5.3	Coluna T-03	53
5.3.1	Pré-Processamento dos Dados da Coluna T-03	53
5.3.2	Treinando o Autoencoder para a Coluna T-03	55
5.3.3	Inferindo a Concentração de propano (C3+) na Corrente de Topo da Coluna T-03 ...	56
5.3.4	Efeito do ruído na inferência gerada a partir do encodado	57
5.4	Base de dados artificiais	58
5.4.1	Base de dados LL.....	59
5.4.2	Base de dados NL.....	61
5.4.3	Base de dados LN.....	66
5.4.1	Base de dados NN.....	69
5.5	Conclusões.....	73
Capítulo 6 – Considerações Finais		74
6.1	Sugestões para trabalhos futuros	75
Referências		1
Apêndice A		9
Apêndice B		11
Apêndice C		29
Apêndice D		48

LISTA DE FIGURAS

Figura 1.1: Metodologia simplificada para o desenvolvimento de analisadores virtuais.....	3
Figura 1.2 Esquema ilustrativo da arquitetura de um autoencoder.....	4
Figura 1.3 Diferenças nos agrupamentos gerados para os dígitos escritos à mão através das metodologias baseadas em PCA e autoencoder.	4
Figura 1.4 Projeção espacial da classificação de assunto de documentos feita, a esquerda, com PCA e a direita com autoencoder.	5
Figura 2.1: Classificação de diferentes tipos de autoencoders pela sua arquitetura.	8
Figura 2.2 Quadro demonstrativo de diferentes funções de ativação, suas equações e derivadas – (KETKAR, 2017).....	10
Figura 2.3 Mais representações de funções de ativação não-lineares.	11
Figura 2.4 Pesquisas relacionadas a diferentes bibliotecas de treinamento de redes neurais. – Chollet 2017.....	14
Figura 2.5 Esquema do autoencoder comprimindo as entradas que são passadas a um modelo caixa preta que fará a inferência da variável alvo.	16
Figura 2.6 Diagrama de classificação entre os diferentes métodos de diagnóstico de falhas.....	20
Figura 4.1: Fluxograma simplificado da unidade de separação de propeno extraído de Schultz (2015).	29
Figura 4.2: Modelo da unidade criado no Aspen Plus.....	31
Figura 4.3: Modelo de entradas e saídas do processo	32
Figura 5.1 Variância explicada acumulada e individual para as variáveis de entrada da coluna T-01.	38
Figura 5.2: Gráfico do T^2 versus as amostras da coluna T-01.....	39
Figura 5.3 Gráfico de controle T^2 estatístico sem amostras acima do limite.....	39
Figura 5.4 Comparativo entre pontos de operação reconstruídos pelo autoencoder e dados de entrada, separados por variável. – coluna T-01	41
Figura 5.5 Comparativo entre pontos de operação reconstruídos pelo autoencoder e dados de entrada, separados por variável – coluna T-01	42
Figura 5.6 Erro de Reconstrução dos dados de teste pelo autoencoder – coluna T-01	42
Figura 5.7 Inferências utilizando o espaço latente e componentes principais – inferência dos pesados no topo da coluna T-01.....	45
Figura 5.8 Inferências e valores esperados ao longo dos pontos de operação	45
Figura 5.9 Inferências geradas a partir do encodado com dados puros e dados dopados	46
Figura 5.10 Diagrama de inferências e erro de reconstrução dos pontos de operação pelo autoencoder	47
Figura 5.11 Variância explicada acumulada e individual para as variáveis de entrada da coluna T-02.	49
Figura 5.12: Gráfico do T^2 versus as amostras da coluna T-02.....	49
Figura 5.13 Erro de Reconstrução dos dados de teste pelo Autoencoder – coluna T-02... ..	50
Figura 5.14 Inferências utilizando o espaço encodado e principais componentes – inferência de propeno no topo da coluna T-02.....	51
Figura 5.15 Inferências e valores esperados ao longo dos pontos de operação - coluna T-02	52
Figura 5.16 Diagrama de inferências e erro de reconstrução dos pontos de operação pelo autoencoder	52

Figura 5.17 Variância explicada acumulada e individual para as variáveis de entrada da coluna T-03.	54
Figura 5.18: Gráfico do T^2 versus as amostras da coluna T-03.....	55
Figura 5.19 Erro de Reconstrução dos dados de teste pelo Autoencoder.....	56
Figura 5.20 Inferências utilizando o espaço encodado e principais componentes – inferência de propano no topo da coluna T-03.....	57
Figura 5.21 Inferências e valores esperados ao longo dos pontos de operação – coluna T-03	57
Figura 5.22 Diagrama de inferências e erro de reconstrução dos pontos de operação pelo autoencoder	58
Figura 5.23 Variância explicada por cada principal componente do PCA.	59
Figura 5.24 Inferências de y a partir dos espaços reduzidos por PCA e pelo autoencoder	60
Figura 5.25 Diagrama de inferências e erro de reconstrução dos pontos de operação.....	61
Figura 5.26 Variância Explicadas por cada principal componente do PCA	62
Figura 5.27 Comparação visual entre inferências a partir do espaço latente e dos principais componentes	63
Figura 5.28 Diagrama de inferências e erro de reconstrução do autoencoder com 4 neurônios no espaço latente.	65
Figura 5.29 Diagrama de inferências e erro de reconstrução com 2 neurônios no espaço latente.....	65
Figura 5.30 Variância explicada por cada principal componente do PCA.....	66
Figura 5.31 Comparação visual entre inferências a partir do espaço latente e dos componentes principais.	68
Figura 5.32: Diagrama de inferências e erro de reconstrução do autoencoder – dados LN	69
Figura 5.33: Variância explicada por cada principal componente do PCA.....	70
Figura 5.34: Comparação visual entre inferências a partir do espaço latente e dos principais componentes	71
Figura 5.35: Diagrama de inferências e erro de reconstrução do autoencoder – dados NN	72

LISTA DE TABELAS

Tabela 3.1: Hiperparâmetros Utilizados para Treinamento.....	25
Tabela 4.1: Lista de representações dos equipamentos da unidade Schultz (2015).....	28
Tabela 4.2: Códigos dos equipamentos da unidade de Schultz (2015).....	28
Tabela 4.3: Especificação da corrente de alimentação, composta po GLP.....	30
Tabela 4.4: Notação das variáveis do processo.....	33
Tabela 4.5: Principais correntes da unidade.	33
Tabela 4.6: Dados de operação da coluna T-01 gerados por SHULTZ.....	34
Tabela 4.7: Dados de operação da coluna T-02 gerados por Schultz (2015)	34
Tabela 4.8: Região utilizada para a coluna T-03.....	35
Tabela 5.1 Descrição da variável de saída $ZC4 + 4$ (concentração dos pesados no topo da coluna T-01).....	37
Tabela 5.2: Variáveis disponíveis, da coluna T-01, que podem ser utilizadas como entrada para o modelo.	38
Tabela 5.3: Melhores Rede Rasa Treinadas.....	40
Tabela 5.4: Resultados dos critérios de avaliação para o conjunto de teste – inferência dos pesados no topo da coluna T-01.	43
Tabela 5.5: Parâmetros do modelo linear ajustado – inferência dos pesados no topo da coluna T-01.	43
Tabela 5.6: Resultados dos critérios de avaliação para os modelos a partir de principais componentes – inferência dos pesados no topo da coluna T-01.	44
Tabela 5.7: Coeficientes da regressão linear utilizando Lasso-Lars – inferência dos pesados no topo da coluna T-01.....	44
Tabela 5.8: Descrição da variável de saída $ZC3 - 8$ – (fração mássica de propeno no topo da coluna T-02).	48
Tabela 5.9: Variáveis da coluna T-02 que podem ser utilizadas como entrada para o modelo.....	48
Tabela 5.10: Melhores Redes Rasas Treinadas – coluna T-02.....	50
Tabela 5.11: Resultados dos critérios de avaliação para os modelos utilizando o conjunto de dados de teste – coluna T-02	51
Tabela 5.12: Descrição da variável de saída $ZC3 + 15$ (fração mássica de propano no topo da coluna T-03).	53
Tabela 5.13: Variáveis, da coluna T-03, que podem ser utilizadas como entrada para o modelo.....	53
Tabela 5.14: Melhores Redes Rasas Treinadas – coluna T-03.....	55
Tabela 5.15: Resultados dos critérios de avaliação para os modelos utilizando o conjunto de dados de teste – coluna T-03	56
Tabela 5.16: Descrição da variável de saída y	59
Tabela 5.17: Melhores Rede Rasa Treinadas.....	59
Tabela 5.18: Resultados dos critérios de avaliação para os modelos utilizando o conjunto de dados de teste.	60
Tabela 5.19: Descrição da variável de saída y	61
Tabela 5.20: Melhores Rede Rasa Treinadas.....	62
Tabela 5.21: Resultados dos critérios de avaliação para os modelos utilizando o conjunto de dados de teste.	63

Tabela 5.22: Melhores redes com 2 neurônios no espaço latente	64
Tabela 5.23: Resultados dos critérios de avaliação para os modelos utilizando o conjunto de dados de teste.	64
Tabela 5.24: Descrição da variável de saída y	66
Tabela 5.25 Melhores Rede Rasas Treinadas.	66
Tabela 5.26 Resultados dos critérios de avaliação para os modelos utilizando o conjunto de dados de teste.	67
Tabela 5.27: Descrição da variável de saída y	69
Tabela 5.28: Melhores Redes Rasas Treinadas com 5 neurônios na camada interna	70
Tabela 5.29: Resultados dos critérios de avaliação para os modelos utilizando o conjunto de dados de teste.	70
Tabela 5.30 Coeficientes das regressões lineares utilizando Lasso-Lars.....	71

NOTAÇÃO E SIMBOLOGIA

<i>ACO</i>	<i>Ant colony optimization</i>
<i>AIC</i>	<i>Akaike information criterion</i>
<i>API</i>	<i>Application Programming Interface</i>
<i>BIC</i>	<i>Bayesian information criterion</i>
<i>LSTM</i>	<i>Long short-term Memory</i>
<i>GLP</i>	Gás liquefeito de petróleo
<i>KPCA</i>	<i>Kernel principal component analysis</i>
<i>LARS</i>	<i>Least angle regression</i>
<i>LASSO</i>	Penalização aplicada aos coeficientes de uma regressão
<i>LASSOLARS</i>	Regressão <i>LARS</i> com penalização L1 na regressão
<i>PET</i>	Politereftalato de Etileno
<i>PIMS</i>	<i>Plant Information Management System</i>
<i>GIMSCOP</i>	Grupo de Intensificação, Modelagem, Simulação, Controle e Otimização de Processos
<i>RBM</i>	<i>Restricted Boltzman Machines</i>
<i>MSE</i>	<i>Mean Squared Error</i>
<i>MSLE</i>	<i>Mean Squared Logarithmic Error</i>
<i>AE</i>	<i>Autoencoder</i>
<i>PC</i>	<i>Principal Component</i>
<i>PCA</i>	<i>Principal Component Analysis</i>
<i>PCR</i>	<i>Principal Component Regression</i>
<i>PLS</i>	<i>Partial Least Squares</i>
R^2	Coefficiente de determinação
<i>RMSE</i>	<i>Root Mean Squared Error</i>
<i>SGD</i>	<i>Stochastic Gradient Descent</i>
<i>SOC</i>	<i>Self-Optimizing Control</i>
<i>SSE</i>	<i>Sum of squared error</i>

Capítulo 1 – Introdução

1.1 Motivação

Estamos em uma situação de abundância de dados, principalmente na indústria, onde há o monitoramento e algumas vezes uma redundância de leituras dos processos produtivos. O desafio atual é encontrar a melhor forma de tratar e utilizar a quantidade avassaladora de dados, levando em consideração que a redundância e monitoramento de variáveis desnecessárias, além de encarecer pelo custo de instrumentação, podem afetar o julgamento e ações tomadas pelos usuários, dada a dificuldade em interpretar os múltiplos sinais do processo.

A busca pela especificação de produtos sempre fez parte do trabalho de engenheiros químicos na indústria, bem como o cumprimento de regulamentações ambientais sobre emissões decorrentes do processo produtivo. Esta busca por padrões de qualidade e condições específicas exigem um controle de processo minucioso que está fortemente atrelado à viabilidade econômica do mesmo. Um exemplo disto são os analisadores em linha, que apesar de medir a composição de uma dada corrente, agregam uma maior complexidade na instalação e manutenção da planta, além do custo e tem uma defasagem nas suas medidas, mesmo que bem reduzidas em comparação com análises laboratoriais, que podem impactar negativamente no desempenho da planta. Isso afeta diretamente os custos de produção e retorno econômico do processo.

Metodologias que utilizam variáveis de processo de fácil medição para inferir grandezas de mais difícil ou demorada obtenção desempenham um papel de grande importância na otimização da produção. São alternativas para substituir instrumentos físicos ou inferir variáveis de difícil monitoramento, são encontrados na literatura como *soft-sensors*, da união de *software* e sensores, ou quando acompanhadas de metodologias de manutenção e atualização das inferências, são chamadas de analisadores virtuais.

Sabendo disto, uma solução para este problema e finalmente para gerar informações frequentes e confiáveis a partir da grande quantidade de dados de operação já disponíveis, as inferências vêm sendo utilizadas na indústria de processos químicos.

1.2 Inferências

Inferências são modelos utilizados para estimar variáveis de difícil medição e geralmente variáveis relativas à qualidade ou interesse econômico do processo. Com frequência essas variáveis são obtidas por análises laboratoriais com um atraso significativo, impedindo um controle mais rápido e desperdício de produção. Suas aplicações na indústria são inúmeras e entre elas é possível salientar o monitoramento puro e simples, detecção de falhas, redundância de sensores e o controle avançado. A qualidade da inferência gerada é diretamente correlacionada com a qualidade dos dados utilizados, sendo então imprescindível uma prévia checagem dos dados nos quais se baseia a inferência (QIN; YUE; DUNIA, 1997).

O termo mais comum na literatura para inferências é atribuído a *soft-sensors*, sendo este a junção de *sensor*, devido a seu uso como instrumento de medição, e *software* remetendo à característica do modelo matemático em si. Entretanto é importante salientar que o mesmo termo é também utilizado quando se trabalha com analisadores virtuais, porém há uma diferença esquemática entre ambos. Como Facchin (2005) menciona, a inferência é o modelo e o analisador virtual é a união da inferência com a medição da variável inferida e uma ferramenta de correção da predição da variável inferida.

A modelagem de processos industriais envolve tipicamente a estimação de um número finito de variáveis. Certas características a serem medidas podem ser custosas e muito demoradas para serem obtidas. Modelos fenomenológicos são derivados de conceitos físicos e químicos do processo. Os modelos baseados em dados, empíricos, têm grande ascensão pela revolução digital na indústria, fazendo com que a abundância de dados disponíveis seja processada e aproveitada (BAKIROV; GABRYS; FAY, 2017).

A metodologia proposta por Kadlec & Gabrys (2009) para desenvolvimento de *soft-sensors* é sumarizada na Figura 1.1. Na primeira etapa, é realizada uma inspeção nos dados com o objetivo de se obter uma visão geral de sua estrutura e identificar quaisquer problemas como por exemplo, variáveis bloqueadas com valor constante ou dados faltantes. A segunda etapa parte para a seleção dos dados a serem utilizados para ajuste e avaliação do modelo. Também são identificadas e selecionadas as regiões dos dados de operação estacionária. Na terceira etapa, o objetivo é filtrar os dados de forma a viabilizar a modelagem, removendo *outliers* e realizando um escalonamento adequado. A quarta etapa é o ajuste do modelo da inferência. Existem diversas metodologias para escolha de modelos, o autor salienta que não há uma forma teórica fortemente estabelecida e propõe uma abordagem que parte de um modelo simples, aumentando gradualmente a complexidade do mesmo. Para o desenvolvimento de analisadores virtuais entraria a quinta etapa que consiste na manutenção e ajuste fino do *soft-sensor*. Esta etapa é necessária tendo em vista as alterações sofridas pelo sistema inferido e que não estão contempladas na modelagem, desvio que diminuem o desempenho do mesmo.

Atualmente, a maioria dos *soft-sensors* não fornece mecanismos automatizados para sua manutenção, dado seu custo, baseando sua avaliação e julgamento na inspeção visual do operador entre o valor correto e a dada predição (KADLEC; GRBIĆ; GABRYS, 2011).

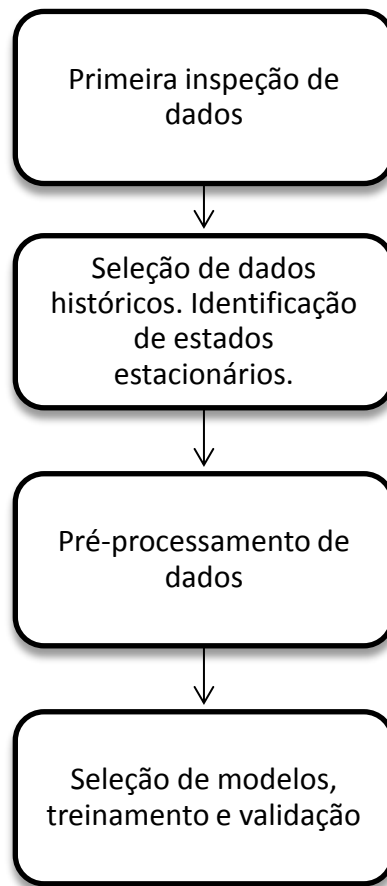


Figura 1.1: Metodologia simplificada para o desenvolvimento de analisadores virtuais.

1.3 Autoencoder

Com o avanço das técnicas de aprendizado de máquina, um campo específico tem tido maior destaque nas pesquisas atuais: o *deep learning*. Conceito que compreende os métodos de aprendizagem que vão além do simples reconhecimento de uma tarefa específica e conseguem fazer abstrações e definir representações a partir dos dados alimentados. Suas aplicações são notórias em áreas como as de visão computacional, reconhecimento de padrões, reconhecimento e análise de fala, processamento de linguagem natural e sistemas de recomendação (LIU et al., 2017).

Dentro deste contexto, as redes neurais se destacam e têm uma crescente utilização dadas as vastas bases de dados disponíveis na indústria. Autoencoders são um esquema de aprendizado sem supervisão das características de um sistema, em que a camada interna age como um extrator de características gerais (HONG et al., 2015).

O autoencoder pode ser apresentado como uma generalização não linear da análise de principais componentes, PCA, que usa uma rede neural para codificação, ou 'encodamento', capaz de reduzir a dimensão dos dados. Associado a esta primeira parte da rede temos outra que decodifica o espaço reduzido para recuperar os dados comprimidos (HINTON; SALAKHUTDINOV, 2006).

Dada como um exemplo para ilustrar a estrutura simétrica de um autoencoder, tem-se a Figura 1.2, mostrando à esquerda as entradas, ou as variáveis do sistema, na parte central se encontra o espaço latente, também chamado de codificado, que representa a

condensação das informações passadas que serão então decodificadas para as saídas que devem se assemelhar às entradas (LECUN et al., 1998a).

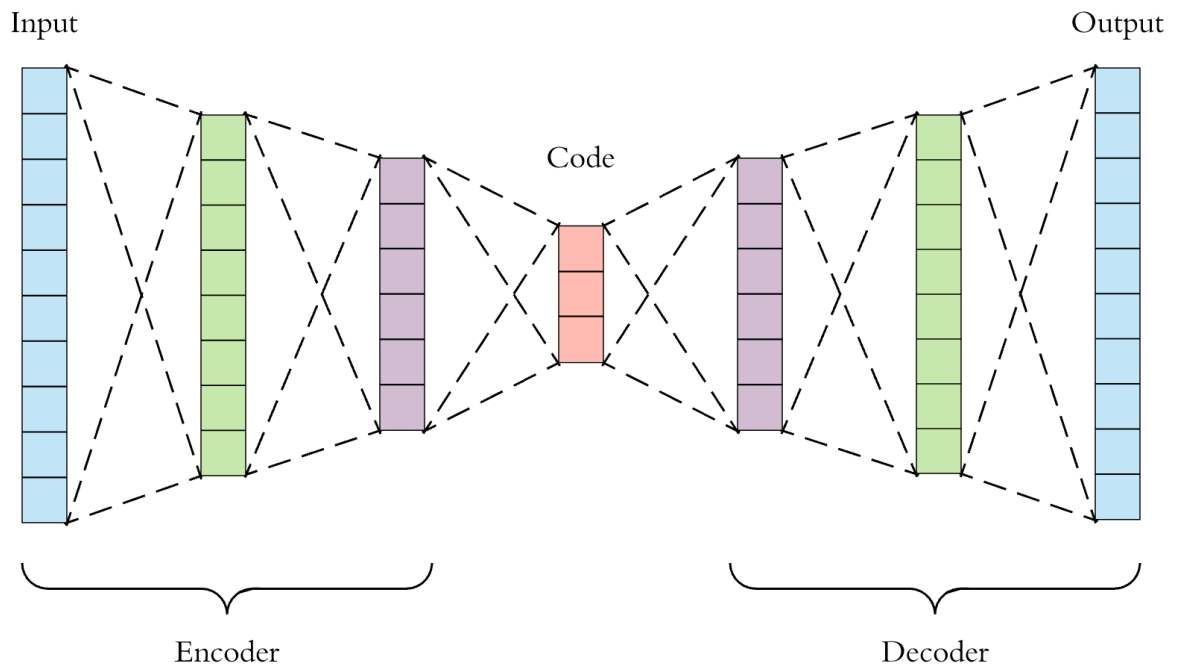


Figura 1.2 Esquema ilustrativo da arquitetura de um autoencoder.

Há na literatura diversos trabalhos comparativos entre PCA e autoencoders salientando a capacidade superior de generalização, adaptação a não linearidades e melhores resultados na separação dos dados. Como o trabalho de HINTON, SALAKHUTDINOV (2006) mostra na Figura 1.3, a diferença na classificação de arquivos utilizando PCA e autoencoder na separação de um subgrupo de dígitos escritos à mão, o MNIST, amplamente utilizado na literatura sobre autoencoder (MANNING-DAHAN, 2017).

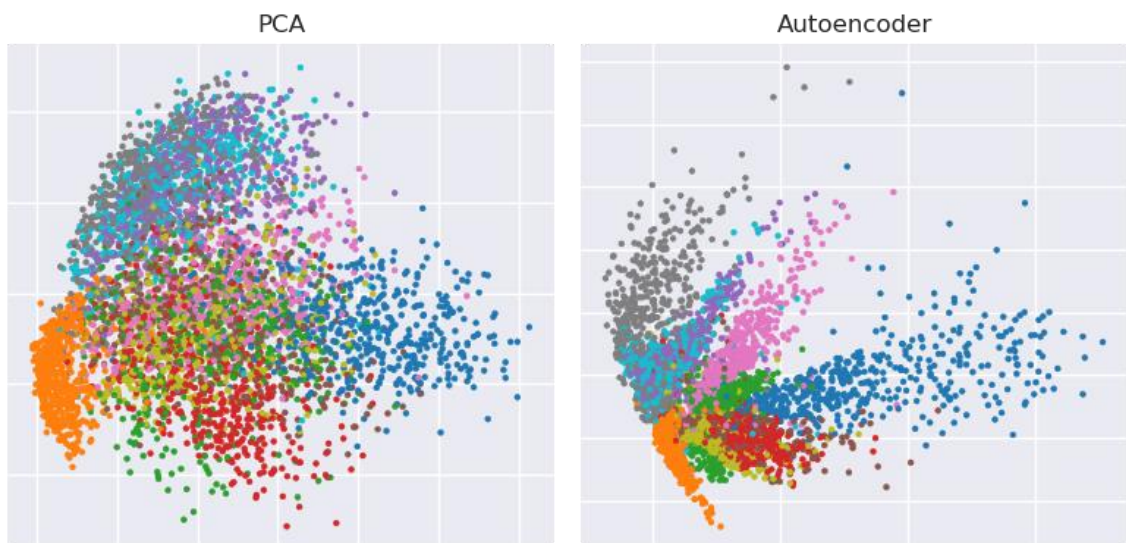


Figura 1.3 Diferenças nos agrupamentos gerados para os dígitos escritos à mão através das metodologias baseadas em PCA e autoencoder.

Ou a classificação atingida bidimensionalmente tanto por PCA, novamente a esquerda na Figura 1.4, perante resultados obtidos com o autoencoder apresentados no mesmo trabalho de Hinton (2001), mostrando a capacidade superior de compressão de informações e generalização das características do sistema.

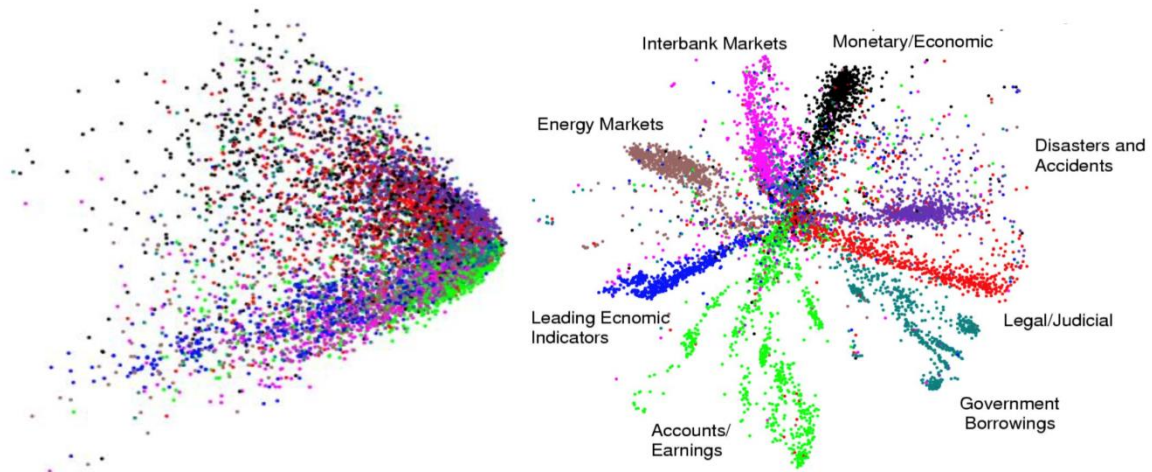


Figura 1.4 Projeção espacial da classificação de assunto de documentos feita, a esquerda, com PCA e a direita com autoencoder.

Analogamente, seria possível utilizar o espaço reduzido pelo autoencoder para inferência de variáveis chaves de um processo. Em processos em que há uma divergência entre a frequência amostral de certas medidas o autoencoder poderia ser utilizado na condensação das informações *online* e inferir outras que necessitam um maior tempo de análise ou teste.

Adicionalmente, devido sua característica de aprendizado não supervisionado com a replicação das entradas, há uma checagem inerente dos dados a ser avaliado. Ou seja, quando um ponto de operação anômalo ou não conhecido é utilizado na inferência será possível distingui-lo de um ponto de operação normal.

1.4 Objetivo do trabalho

O objetivo geral deste trabalho é utilizar o autoencoder como uma ferramenta para desenvolvimento de inferências a partir de dados de operação fazendo paralelamente a checagem destes dados. Para tanto os objetivos específicos que guiaram este trabalho foram:

- Desenvolvimento de uma metodologia para geração das inferências empíricas a partir do autoencoder;
- Caracterizar as vantagens da ferramenta utilizada frente ao atual e mais utilizado método de geração de inferências.

- Verificar a eficiência da rede autoencoder em reconhecer dados de entrada espúrios e não conformes, sinalizando que sua utilização pode levar a inferências não confiáveis.

1.5 Estrutura da dissertação

Neste capítulo foi realizada a apresentação geral desse trabalho, juntamente com os principais objetivos e motivações para o desenvolvimento do mesmo.

O segundo capítulo traz uma fundamentação teórica para auxiliar e dar base para uma melhor compreensão do trabalho desenvolvido. Também é apresentada uma revisão bibliográfica sobre as informações e conceitos que utilizados como base neste trabalho, bem como algumas definições e nomenclaturas.

No terceiro capítulo é explicada a metodologia proposta para o desenvolvimento do autoencoder e desenvolvimento das inferências a partir do mesmo.

No capítulo quatro, são ilustrados os estudos de caso, uma unidade de separação de Propeno/Propano simulada e algumas bases de dados artificialmente geradas para este trabalho para facilitar o entendimento e caracterizar a sua utilização.

O quinto capítulo apresenta os resultados obtidos utilizando os estudos de caso.

E por fim, o último e sexto capítulo apresenta as conclusões e sugestões de trabalhos futuros.

Capítulo 2 – Fundamentação Teórica e Revisão Bibliográfica

Este capítulo apresenta uma pesquisa bibliográfica dos principais temas sobre o desenvolvimento de inferências especificamente baseadas no autoencoder. Iniciando por uma apresentação sobre redes neurais, fazendo uma caracterização dos autoencoders e falando da utilização de inferências a partir de redes neurais. Concluindo o capítulo com a classificação e modos de comparação de inferências.

2.1 Redes Neurais

Redes neurais são aproximadores universais (HOSKINS; HIMMELBLAU, 1988). Redes neurais são utilizadas em diversos setores da indústria, sendo o mais comum na análise de dados (SARLE, 1994). Seus modelos podem ser modelos não-lineares de proposta geral e flexíveis, que, dando unidades e dados suficientes podem aproximar virtualmente qualquer função a qualquer grau de acurácia (CYBENKO, 1989)(HORNIK; STINCHCOMBE; WHITE, 1989).

Pode-se entender o paralelo computacional das redes neurais, comparando às condições lógicas, reforçando a ideia de que podemos arranjar e ajustar as unidades lógicas para desenvolvimento de um modelo complexo que seja aderente ao sistema em questão. Eles consistem em unidades, também chamadas de neurônios, interconectados entre si de diversas formas sendo mais comum sua organização em camadas (NIELSEN, 2015).

2.1.1 Terminologia e Arquitetura

As redes neurais são compostas por unidades, organizadas geralmente entre camadas de entrada, ocultas e de saída. Segundo, Nielsen (2015), o termo camada oculta, ou *hidden layer* do inglês, não significa nada além de 'nem uma saída nem uma entrada'. O número de camadas ocultas separa as redes entre *shallow*, ou rasas, quando só há uma camada oculta, ou *deep*, profundas, quando há mais camadas internas. Alguns autores argumentam que somente redes muito complexas podem ser chamadas de *deep neural networks* entretanto, o limite constantemente muda e torna tal terminologia confusa. A Figura 2.1 apresenta quatro exemplos ilustrativos de redes neurais. A primeira linha traz as redes

rasas, onde temos as entradas, uma camada interna e as saídas. E as redes situadas na parte de baixo da figura apresentam redes mais camadas internas.

Os neurônios podem ser completamente interconectados, isto é cada neurônio de uma camada anterior tem uma ligação com cada neurônio da camada seguinte. Este tipo é também chamado de unidade densa, cujo antônimo, unidade escassa, denomina o caso inverso, quando não há ligação completa de todas as unidades entre camadas. A arquitetura de uma rede neural é definida pelo arranjo de ligações entre suas unidades. Quando temos camadas internas com mais unidades que entradas, é dada a denominação de *overcomplete* e quando há a redução do número de neurônios elas são chamadas de redes *undercomplete*, como é possível visualizar na Figura 2.1 (CHARTE et al., 2018).

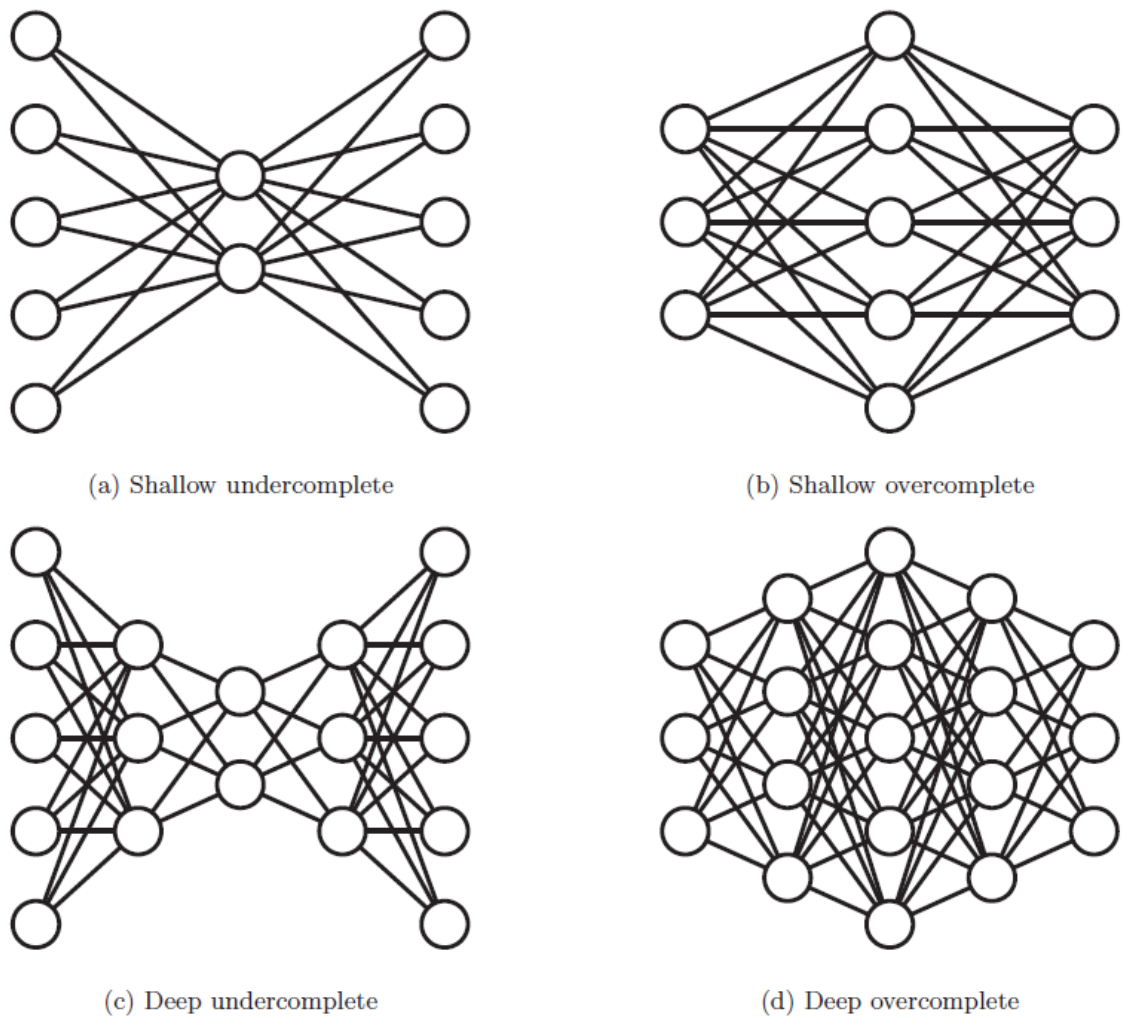


Figura 2.1: Classificação de diferentes tipos de autoencoders pela sua arquitetura.

A ligação entre as unidades da rede possui um peso que denota a sua importância e cada neurônio possui um *bias*, exceto os da entrada. Assim sendo, quando há um peso inexpressivo, muito próximo a zero, nota-se que não há influência do neurônio, ou unidade, da camada anterior sobre aquele da camada subsequente. O sinal que chega a uma unidade é a combinação linear, o somatório, dos pesos das unidades ligadas a ela com a adição do *bias* da mesma e tendo seu valor aplicado a uma função de ativação para

computar sua saída (HINTON et al., 2012). Uma breve exposição das mesmas se fará na próxima seção.

Ainda com respeito a arquitetura das redes, o tipo mais comum é quando se têm as saídas de uma camada sendo as entradas da subsequente. Chamada de rede *feedforward* tem-se a informação sendo passada adiante, sem laços ou *loops*, sem que uma função de ativação tenha dependência de sua própria saída. Característica das redes recorrentes, idealizadas para trabalhar com um atraso na transmissão da informação, porém utilizadas em menor escala até então pela menor eficácia dos seus algoritmos de treinamento (SCHWENK; BENGIO, 2000).

O aprendizado da rede é dado pelo algoritmo de *backpropagation*, onde uma função objetivo é minimizada para ajustar a rede a uma determinada saída. É necessário, portanto, que a função objetivo seja função das saídas da rede neural e que possa ser escrita como uma média das funções objetivo para cada amostra de treino. Assim o algoritmo relaciona alterações nos parâmetros (pesos e bias) às alterações na função objetivo. Essas alterações podem ser calculadas camada a camada obtendo através das derivadas parciais dos desvios, ou erros, associados a cada uma, que determinam a alteração dos parâmetros buscando a minimização da função objetivo. Há o cálculo da função objetivo com uma inicialização aleatória dos parâmetros e o erro, que rege a alteração dos parâmetros, é propagado de maneira reversa, ou seja, alterando os mesmos a partir da última camada até a primeira (LECUN et al., 1998b).

A magnitude da alteração dos parâmetros das camadas é comumente chamada de taxa de aprendizado. Dependendo da função de ativação utilizada e das entradas da unidade, pode-se ter a saturação do neurônio, sendo que isto acontece quando a taxa de aprendizado fica muito próxima de zero ou de um, não havendo assim qualquer atualização daqueles parâmetros (NIELSEN, 2015).

2.1.2 Funções de Ativação

Uma unidade, ou neurônio, da rede recebe sinais de todas as unidades da camada anterior e sobre o somatório destes é aplicada uma certa operação, chamada então de função de ativação para o cálculo do sinal de saída desta unidade (CHARTE et al., 2018).

As funções de ativação eram geralmente restritas a condições de contorno e não-monótonas (caso das baseadas em funções trigonométricas). Existem as mais diversas funções de ativação, por exemplo: função linear, função binária ou booleana, função sigmoidal ou logística, função exponencial, função tangente hiperbólica e as mais atuais variantes de unidades lineares como: ReLU, leaky ReLU, PReLU, ELU e SELU. (KLAMBAUER et al., 2017)(HE et al., 2015)(GODFREY; GASHLER, 2015)(MISHKIN; SERGIEVSKIY; MATAS, 2016)(CYBENKO, 1989). A Figura 2.2 apresenta algumas destas funções mencionadas, em conjunto com seus gráficos, suas equações representativas e primeira derivada.




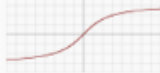



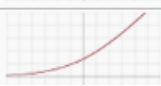
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Figura 2.2 Quadro demonstrativo de diferentes funções de ativação, suas equações e derivadas – (KETKAR, 2017)

A função booleana é pouco utilizada, sendo mais vista em exemplos didáticos para facilitar o entendimento da função de ativação e seu papel nas redes neurais. A linear tem sua principal utilização como camada de saída (CHARTE et al., 2018). A função sigmoial foi, por muitos anos, a mais utilizada com redes neurais é encontrada também como função logística e muito utilizada para classificações binárias em redes neurais (GODFREY; GASHLER, 2015). Softmax é resultante da divisão da exponencial pela soma de todas as unidades da camada, podendo ser interpretada como uma distribuição de probabilidade e utilizada em classificações não binárias (NIELSEN, 2015). A tangente hiperbólica é também uma função sigmoial, mas apresenta melhorias para treinamento devido sua derivada produzir gradientes mais robustos que a antecessora (LECUN et al., 1998b).

A Figura 2.3 mostra algumas das funções não lineares mais comuns utilizadas.

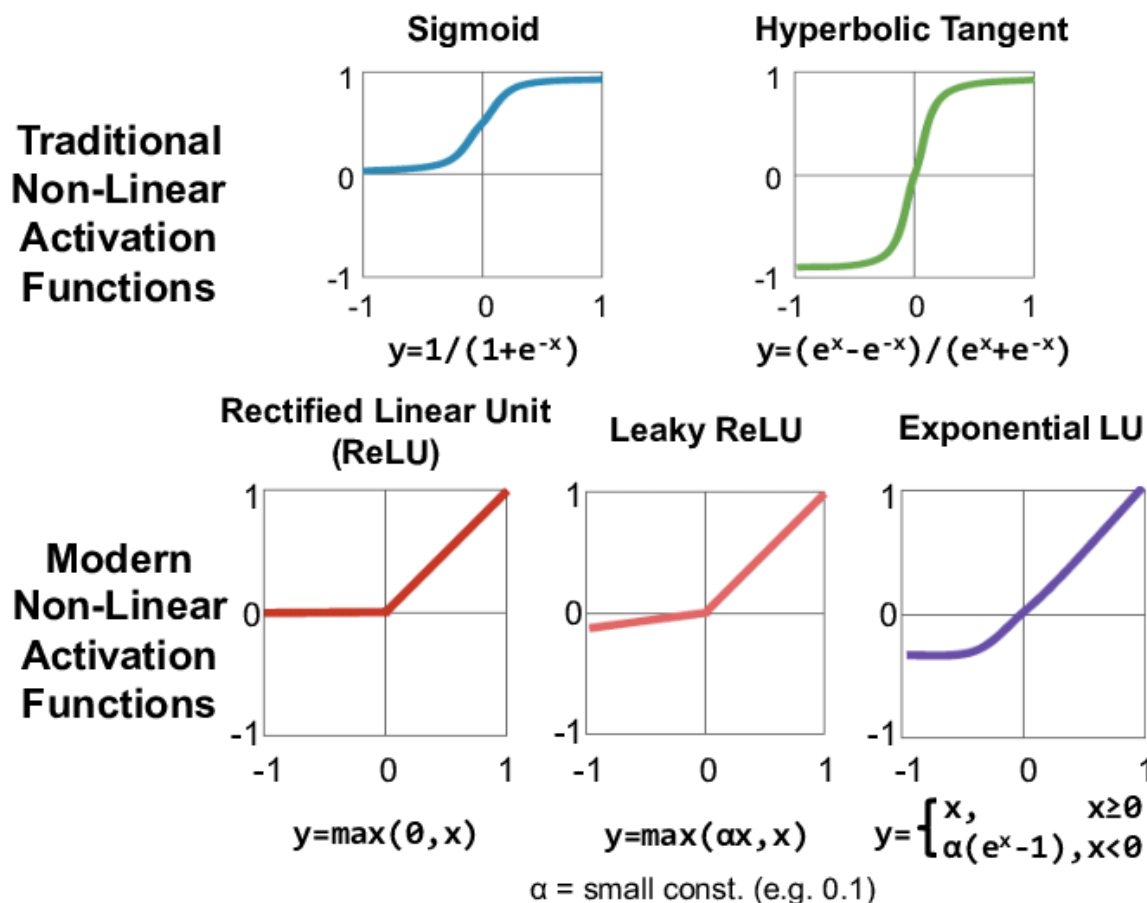


Figura 2.3 Mais representações de funções de ativação não-lineares.

As funções derivadas de unidades lineares, ou seja, funções compostas são bastante utilizadas devido sua maior versatilidade e funcionamento em uma gama mais ampla de problemas. São funções parametrizadas que buscam solucionar problemas frequentes no treinamento de redes mais complexas da mesma forma que a tangente hiperbólica fez com a função logística (NAIR; HINTON, 2010)(ZEILER, 2012).

2.1.3 Otimizadores

Os otimizadores utilizados para a aprendizagem da rede neural, se baseiam principalmente no algoritmo do gradiente descendente, ou SGD do inglês (*Stochastic Gradient Descent*). O algoritmo pode ser aplicado retroativamente nos parâmetros da rede em pequenos grupos de dados e de forma aleatória, caracterizando o gradiente descendente estocástico. Entretanto, o algoritmo agrega hiperparâmetros (como por exemplo a taxa de aprendizado - *lambda*) que são difíceis de serem ajustados e ainda assim estão propensos a estagnarem em pontos de sela da função objetivo (DAUPHIN et al., 2014). Com avanços nas pesquisas e crescente utilização de redes neurais, surgiram inúmeros otimizadores mais complexos e robustos, RUDER (2016) traz uma revisão dos mais importantes atualmente e segue aqui uma breve descrição deles.

Com a dificuldade em trabalhos com dados muito dispersos em termos de frequência, (ZEILER, 2012) propõe o algoritmo do Adagrad, uma variante do SGD com taxas de aprendizado separadas por variáveis, atualizando menos os parâmetros associados a características menos frequentes e mais os parâmetros associados a característica mais frequentes. O algoritmo se atualiza com relação a taxa de aprendizado, mas a forma com

que isso ocorre acumula mínimos quadrados no denominador da taxa de aprendizado, fazendo com que após certo número de iterações, o algoritmo não consegue alterar os parâmetros.

Para solução deste problema, Duchi, Hazan, & Singer (2011), alterou o algoritmo para carregar apenas um intervalo dos gradientes passados no denominador, chamando o novo de Adadelta. De forma semelhante, em trabalho não publicado e posteriormente utilizado em (GRAVES, 2013), buscando reduzir o monótono aumento do denominador, adiciona-se mais hiperparâmetros ao dividi-lo pela média exponencial. Este algoritmo foi apresentado em notas de aula do Professor Hinton e é referido como RMSProp.

Em (KINGMA; BA, 2015) o ADAM é proposto, *Adaptive Momentum Estimation*, que além de ajustar a taxa de aprendizado com relação a cada parâmetro e incorporar restrições que impedem a redução da taxa de aprendizagem, o otimizador leva em conta o primeiro e segundo momento do gradiente. Ainda assim, em treinamento muito intensivos, o método pode instabilizar, e para isto os autores já propuseram uma extensão, ADAMax, que possui menor propensão a redução da taxa de aprendizado e menos inclinado a zerar os *bias*.

Ainda são citados na literatura algoritmos de gradiente conjugado e baseados no método de Broyden-Fletcher-Goldfarb-Shanno(BFGS) (BOLLAPRAGADA et al., 2018). DOZAT (2016) combina o Momento de Nasterov com o ADAM, mas apenas encontra melhorias em alguns exemplos específicos. (REDDI; KALE; KUMAR, 2019) faz a própria proposição utilizando médias móveis exponenciais com o Adagrad, criando o AMSGrad (*Average Moving Stochastic Gradient*) para tentar resolver problemas no treinamento de redes neurais recorrentes.

2.1.4 Treinamento da rede

O treinamento de uma rede é então dado pela minimização de uma função objetivo que será feita por um algoritmo, como os citados anteriormente, alterando os parâmetros da rede, geralmente camada a camada. Entretanto são muitas características relativas tanto a arquitetura, quantidade de amostras por batelada, forma de inicialização dos parâmetros, algoritmo de minimização e função de ativação, que são chamadas de hiperparâmetros das redes (NIELSEN, 2015).

Destes hiperparâmetros, as funções de ativação e otimizadores já foram comentados, restando o tamanho da batelada e a arquitetura.

A quantidade de dados passada para treinamento em cada época é um balanço entre um treinamento mais minucioso e uma maior velocidade de obtenção de resultados. Mishkin, Sergievskiy, & Matas (2016) já reportavam melhores resultados obtidos quando se treinava redes neurais convolucionais com bateladas menores de dados. Há uma deterioração no treinamento da rede neural quando temos tamanhos de batelada muito grandes, por exemplo quando é passada todas as amostras de treinamento em uma batelada (RADIUK, 2018).

A arquitetura está diretamente relacionada ao número de parâmetros da rede neural, já que contém a quantidade de camadas e unidades por camadas. Há uma busca pelo tamanho de rede que tenha capacidade suficiente de generalização das características do sistema em questão sem que ela seja capaz de apenas memorizar os dados passados para treinamento da rede (Chollet, 2015). Neste sentido entram em cena trabalhos novos em que há uma busca por uma regularização do modelo de forma a evitar este *overfitting* da rede, trabalhos que trazem diretrizes no treinamento das redes como o embaralhamento das amostras a cada iteração de batelada do treinamento (LECUN et al., 1998b), a inclusão de regularização L1 ou L2 dos pesos (mais influentes que os *bias*) camada a camada pelo otimizador (ZEILER, 2012), critérios de parada do treinamento quando não há uma melhoria na métrica utilizada para avaliação da função objetivo, normalização do sinal passado camada a camada (IOFFE; SZEGEDY, 2015) até as mais recentes formas de regularização dos pesos da rede, como métodos de *dropout* (DONG; CHEN; PAN, 2017).

Ainda dentro das redes *feedforward* existem diversos tipos diferentes de arquiteturas idealizadas para objetivos distintos como as redes convolucionais, que agregam a rede informações espaciais das entradas e por tanto muito importantes para imagens. Sua recente utilização é maior devido ao crescente campo de visão computacional. Mais sobre pode ser encontrado nos trabalhos de (HONG et al., 2015)(CIREŞAN et al., 2010)(ALMOTIRI; ELLEITHY; ELLEITHY, 2017).

Com tantas configurações possíveis, trabalhos como HOGWILD (NIU et al., 2011), Método de Downpour (DEAN et al., 2012), buscas randômicas (BERGSTRA et al., 2015; BERGSTRA; YAMINS; COX, 2013; JAMES; YOSHUA, 2012), médias elásticas (HE et al., 2015), surgem na busca do ajuste dos hiperparâmetros por diferentes métodos. Noel & Osindero (2014) foca parte dos seus estudos para a paralelização dos cálculos tensoriais, aproveitando o maior número de núcleo das placas de vídeo, tornando permissivo essa otimização pelo menor tempo necessário para treinamento de tantas redes diferentes.

2.1.5 Ferramentas computacionais

Existem diversas ferramentas acadêmicas utilizadas para treinamento de redes neurais, entre as mais conhecidas e utilizadas, se destacam as alternativas *opensource* Caffe da Universidade de Berkeley (JIA et al., 2014), CNTK da Microsoft (YU et al., 2015), PyTorch do Facebook - baseado no Torch de Collobert, Kavukcuoglu, & Farabet (2011), Theano (THE THEANO DEVELOPMENT TEAM et al., 2016) e TensorFlow da Google (ABADI et al., 2016; SHI et al., 2017).

A Figura 2.4 retirada de Chollet (2017) demonstra qualitativamente a quantidade de pesquisas feitas no buscador Google entre as bibliotecas de cálculo tensorial mais conhecidas com a adição do Keras, uma API para construção, treinamento e uso de redes neurais. O Keras é uma abstração dos modelos para implementação de aprendizado de máquina na linguagem Python, pode ser utilizada sobre algumas das bibliotecas citadas anteriormente (Theano, Tensorflow e CNTK) sendo oficialmente incorporada como parte do próprio Tensorflow a partir de janeiro de 2017.

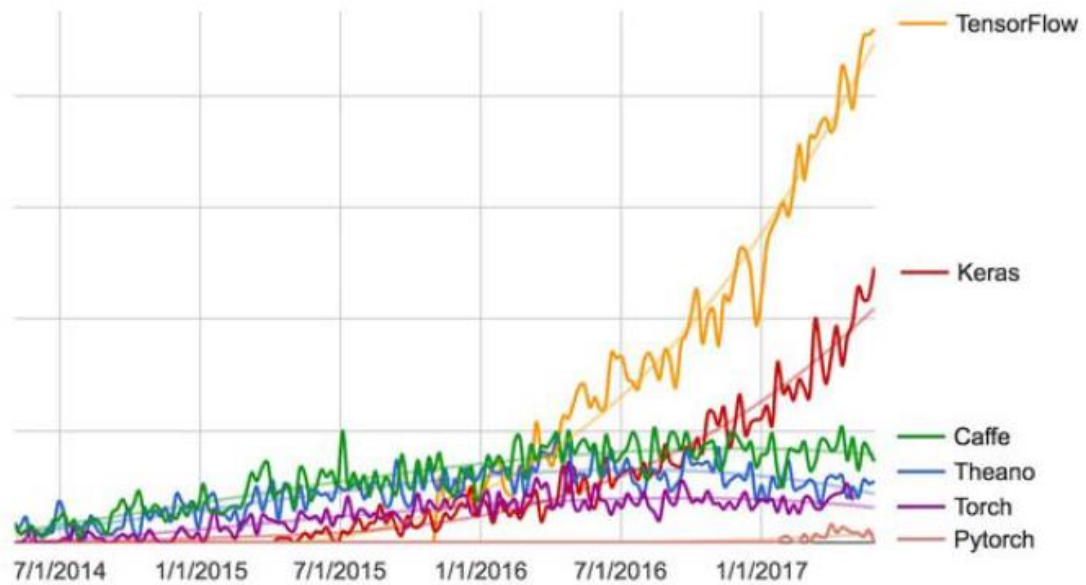


Figura 2.4 Pesquisas relacionadas a diferentes bibliotecas de treinamento de redes neurais. – Chollet 2017

Pensando em futuros desenvolvimentos deste trabalho, toda implementação foi feita utilizando Keras 2.0 sobre a Tensorflow 1.13.1, em Python 3.7.3. É importante ressaltar que a maior parte do código pode ser executado em versões anteriores. Entretanto atualizações podem ter mudado algumas bibliotecas de cálculo tensorial.

2.2 Autoencoder

Autoencoder é uma rede neural que usa uma série de parâmetros de reconhecimento para converter uma entrada em um vetor codificado. E então usa uma série de parâmetros generativos para converter este vetor codificado em uma reconstrução das entradas passadas anteriormente (BENGIO, 2012). Quando se tem uma rede neural sendo treinada para reproduzir a sua entrada na ponta final, pode-se pensar que rede se tornaria apenas uma matriz identidade, porém usando uma camada interna menor, é esperado que a rede aprenda a comprimir as informações de entrada e associar padrões nos dados (BALDI, 2012; BALDI; HORNIK, 1989; JAMES; YOSHUA, 2012).

Esta estrutura é especialmente poderosa dada a sua característica de treinamento não-supervisionado, ou seja, sem que haja uma classificação prévia dos dados utilizados (O'SHEA; KARRA; CLANCY, 2016). Com relação a sua topologia, a sua estrutura de rede neural, quando se tem um autoencoder com um espaço latente de dimensão maior que a entrada do sistema, este é chamado de *overcomplete*, que deve utilizar restrições em seu treinamento para evitar a simples cópia das entradas para as saídas. Muito mais comuns, são os autoencoder em que há uma redução na dimensionalidade das entradas para o espaço latente, sendo estes chamados de *undercomplete*. Assim como as redes neurais, são separados em rasos e profundos pelo número de camadas ocultas, sendo raso o

autoencoder mínimo com apenas três camadas: entradas, espaço latente e reconstrução das entradas (CHARTE et al., 2018).

Há uma maior dificuldade no treinamento de autoencoders não lineares com múltiplas camadas. Se os pesos iniciais forem muito altos tipicamente são atingidos apenas mínimos locais e com pesos iniciais pequenos temos gradientes, e conseqüentemente alteração dos parâmetros durante o treinamento, reduzidos. Diversas melhorias têm sido aplicadas para sanar estas dificuldades ou contornar certas restrições com relação ao tipo de dados utilizado, especialmente na área de visão computacional que é extensamente explorada (LE et al., 2012; LEE et al., 2005; O'SHEA; KARRA; CLANCY, 2016).

Autoencoders rasos, sem mais camadas internas além da dimensão reduzida, podem ser treinados sem pré-treinamento, apesar de que o mesmo vai acelerar o treinamento da rede. Quando o número de parâmetros entre um autoencoder profundo e um raso é o mesmo, podemos afirmar que o *deep* terá menores erros de reconstrução, porém essa vantagem desaparece quando temos um número de parâmetros aumentando. (HOLDEN et al., 2006; LECUN et al., 1998b)

Inferências a partir da redução de espaço são amplamente utilizadas na indústria química muito devido à alta correlação entre suas variáveis medidas (KADLEC; GABRYS; STRANDT, 2009). A qualidade dos dados e a quantidade de informações úteis que contém são fatores primordiais que determinam a qualidade do modelo final (LI et al., 2013).

Trabalhos focados em extensões e novas proposições de metodologias baseadas em PCA e PLS são comuns e bastante utilizados na indústria (BAKIROV; GABRYS; FAY, 2017; KADLEC; GABRYS; STRANDT, 2009; LU et al., 2004). Da mesma forma que a detecção de falhas baseadas em dados históricos de processo abordados nos trabalhos de (L. RUSSELL; CHIANG; BRAATZ, 2000; VENKATASUBRAMANIAN et al., 2003). Entretanto, os próprios autores alertam sobre a limitação destes métodos ao tratar sobre dados que não pertencem à mesma região de operação dos dados de formação. A extrapolação dos modelos empíricos ou semi-empíricos do processo deve sempre ser abordada com cuidado por não se estar trabalhando na região de máxima verossimilhança.

A ampla disponibilidade de dados históricos e os avanços computacionais favorecem o recente avanço de técnicas mais complexas e maior aplicação de ferramentas como as redes neurais na indústria química (FACCHIN, 2005).

Além de uma performance considerável e melhor que métodos atuais em muitos problemas aplicados na indústria, a aplicação do chamado *deep learning* facilita muito a criação de inferências por automatizar uma das partes mais cruciais dos métodos de modelagem: o *feature engineering*, a caracterização do sistema (Chollet, 2017).

Tanto que já são comuns trabalhos de inferência de processos utilizando redes neurais. Agarwal & Budman (2019) usaram redes neurais recorrentes com unidades LSTM para classificar regiões de operação de acordo com sua lucratividade. (CIVLEKOGLU et al., 2007) modelam uma estação de tratamento de efluentes industriais utilizando combinações de redes neurais, na técnica chamada *Neuro-Fuzzy Modeling*. (SHANG et al., 2014) usando unidades RBM em uma rede neural empilhada, para geração de uma inferência em uma unidade de destilação de petróleo. Dentro da área de controle, (GONZAGA et al., 2009) busca a criação de um *soft-sensor* baseado em redes neurais *feedforward* para monitoramento e controle em tempo real de um processo de polimerização de PET.

O autoencoder aparece como uma ferramenta de aprendizado de máquina essencial para caracterização de sistemas complexos. Os trabalhos de (CHOLLET, 2017), (BALDI, 2012), fazem uma melhor explicação de características e detalhes do autoencoder em si. Apesar de amplamente utilizado nas áreas de visão computacional (JARRETT et al., 2009), reconhecimento de fala (JES et al., 2019), codificação de sinais e pré-processamento de dados (CHARTE et al., 2018), a exemplo dos trabalhos de Cireşan, Meier, Gambardella, & Schmidhuber, 2010; J. Deng, Zhang, Eyben, & Schuller, 2014; L. Deng et al., 2010, ele é uma ferramenta pouco utilizada na indústria de processos.

É uma ferramenta de aprendizado não supervisionado utilizada para redução do espaço, agregando a possibilidade de caracterização de não linearidades incluídas no processo. A Figura 2.5 traz uma representação da metodologia desenvolvida neste trabalho. O autoencoder é treinado para a reconstrução das variáveis de entrada e a partir do espaço reduzido é gerada a inferência característica do sistema em estudo.

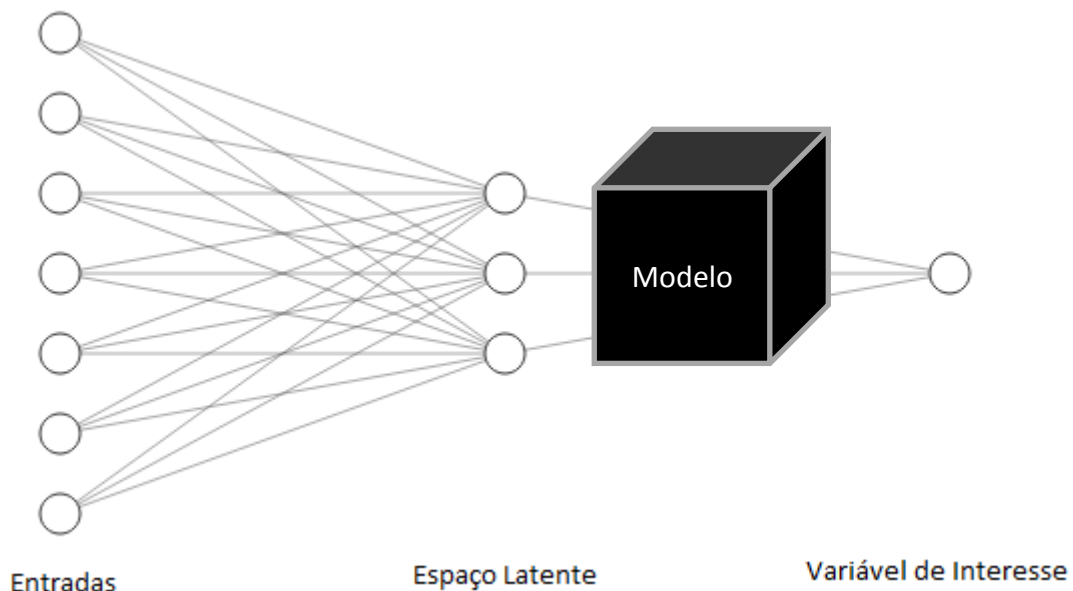


Figura 2.5 Esquema do autoencoder comprimindo as entradas que são passadas a um modelo caixa preta que fará a inferência da variável alvo.

A utilização do espaço latente para geração de novas entradas é alvo do trabalho de (KOLOURI et al., 2018). Já a observação e detecção de falhas utilizando o autoencoder é foco do trabalho de (SAKURADA; YAIRI, 2014), com sinais de satélite, e da tentativa de (GOLDTHORPE; DESMET, 2018). A união da utilização do autoencoder como forma de reduzir o espaço e gerar inferências ao mesmo tempo que monitora os dados em busca de potenciais falhas é uma lacuna até então na literatura. A união destas duas características se torna o maior benefício da utilização do autoencoder para geração de inferências.

2.3 Inferências

As inferências fazem parte da indústria química e estão fortemente conectadas a capacidade de controle e otimização produtiva das plantas. De forma superficial, existe a

distinção entre as inferências provenientes de dados do processo e as inferências provenientes da modelagem fenomenológica do processo. Também conhecidos como modelos fenomenológicos ou 'caixa-branca' (*White-box*), estas inferências a partir de modelos requerem um conhecimento completo acerca dos fenômenos envolvidos no processo. De forma oposta, os modelos caixa-preta (*black-box*), como são conhecidas as inferências a partir de dados, produzem sua estimativa baseada apenas nas observações empíricas presentes nos dados do processo (KADLEC; GABRYS; STRANDT, 2009).

Este trabalho será direcionado à utilização de modelos do tipo caixa-preta, sendo interessante por tanto uma observação sobre o que eles são construídos, os dados à disposição.

2.3.1 Base de dados

A dependência da qualidade da inferência dos modelos do tipo caixa-preta com relação aos dados de processo utilizados é crítica e entre as características a serem observadas em um conjunto de dados, têm-se os cinco fatores discutidos a seguir:

1. Dados faltantes

As amostras podem ter variáveis com valores que não refletem a realidade da medição, são comuns valores retornando zero, infinito ou algum outro valor constante. Devendo-se notar que é possível que tal medidor tenha sido removido, esteja fora dos limites de operação e ainda para aqueles dependentes de ações mecânicas de ativação pode-se ter o desgaste do mesmo. Uma outra causa de dados faltantes em uma base de dados por ser relacionado a transmissão dos dados entre sensores e servidor de armazenamento destes (KADLEC; GABRYS, 2009). A solução pode passar pela substituição destes dados pela interpolação de pontos mais consistentes, ou também pelo descarte das amostras de dados com tal problema.

2. Dados espúrios

Qin et al., (1997) separa os dados espúrios, *outliers*, em duas categorias: os óbvios e os não-óbvios. Os óbvios podem ser considerados dados faltantes, uma vez que são incongruências que violam limitações físicas ou tecnológicas. Já os não-óbvios são mais complexos de serem identificados, uma vez que não violam explicitamente um limite, mas não refletem o estado correto das variáveis.

Abordagens típicas deste problema, buscam detectar os *outliers* através de métodos estatísticos sobre os dados históricos. Pearson (2002) utiliza um algoritmo baseado na média e desvio padrão das variáveis. O identificador Hempel se utiliza da mediana tornando o método mais robusto. Para métodos multivariáveis têm-se o Parametro de Joliffe, o teste T^2 de Hotelling utilizado por Boullosa, Larrabe, Lopez, & Gomez (2017). Mais métodos podem ser encontrados na revisão feita por Hodge & Austin, (2004).

3. Dados enviesados

Os dados podem acabar sendo enviesados principalmente por dois fatores: sensores e processo. Os desvios de conduta de instrumentos afetam a variável medida uma vez que alguns sensores são dependentes de processos mecânicos para produzir suas medições. Por exemplo, o fluxo de um componente pode ser

lentamente diminuído pelo desgaste de uma bomba ou válvula de controle. Tais fatores alteram os dados sem que esta alteração ocorra no processo observado. Já quando a temperatura externa à planta pode variar e influenciar nas medições da planta, ou há uma desativação de catalizadores ou variação pureza de reagentes tem-se uma alteração tanto nas variáveis medidas quanto no processo em si. A distinção e mais importante, a compensação de tal comportamento, requerem um imenso conhecimento do processo e aliado ao fato de que estas mudanças são muito lentas e podem ter mutua influencia torna-se uma tarefa extremamente difícil (KADLEC; GABRYS; STRANDT, 2009).

Na terminologia de aprendizado de máquina, tais efeitos são conhecidos como *drift data*, os tratamentos e algumas soluções são abordados em trabalhos como de GAMA et al. (2004)

4. Dados correlacionados

Tipicamente, os dados industriais disponíveis são altamente correlacionados devido à redundância de algumas variáveis (por exemplo, dois medidores de temperatura vizinhos) para melhor controle do processo. DONG e MCAVOY (1996) caracterizam tal cenário como bases de dados 'ricas em dados e pobres em informação' e demonstram que as variáveis desnecessárias apenas aumentam a complexidade do sistema e não contribuem na modelagem do mesmo.

As formas mais comuns de se trabalhar tal fator são os métodos de redução de espaço como PCA e PLS, usados por exemplo nos trabalhos de (BAKIROV; GABRYS; FAY, 2017; KADLEC; GABRYS; STRANDT, 2009; L. RUSSELL; CHIANG; BRAATZ, 2000). Para casos em que é conhecida uma relação não-linear entre as variáveis, como o trabalho de (SAKURADA; YAIRI, 2014) que se utiliza do *Kernel-PCA*, para redução do espaço sem fazer uma combinação linear entre as variáveis do espaço. Ainda existe toda uma ampla metodologia de seleção de variáveis, como discutido por (RANZAN et al., 2014) para contornar o problema de se ter variáveis correlacionadas.

5. Taxas de amostragem e atraso em medições

Sensores normalmente trabalham em frequências de amostragem diferentes e a sincronização das informações da planta é gerenciada pelo PIMS (*Process Information Management System*) que precisa ser ajustado. Além de algumas variáveis serem dependentes de análises laboratoriais, que levam horas por exemplo, e são críticas no controle do processo, tem-se processos que possuem medidas que fazem mais sentido quando defasadas de um dado tempo de residência ou com sua posição agregada (KADLEC; GABRYS; STRANDT, 2009).

DING e CHEN (2005) fazem uma revisão sobre a frequência de amostragem de medidas em plantas industriais e novamente se salienta o extensivo conhecimento sobre o processo para a compensação deste comportamento nos dados.

Estes cinco cuidados devem ser tomados ao se trabalhar com qualquer base de dados para inferências, tanto para modelos do tipo caixa-preta quanto para modelos caixa-cinza.

2.3.2 Segregação dos dados

Para uma breve explicação, o k -rank pode ser considerado uma junção entre as características das metodologias k -means e y -rank, com separação característica superior ao y -rank e menor custo computacional comparando aos métodos de validação cruzada como k -fold, comumente utilizados (SANTOS et al., 2019).

2.3.3 Modelagem da inferência

Hoskins & Himmelblau (1988) já propuseram utilizar uma rede neural, multicamadas modelos de percepção *feedforward*. Demonstrando a habilidade das redes de descrever modelos e adaptá-los as diferentes distribuições estatísticas das variáveis medidas envolvendo mapeamentos não lineares. Em seu trabalho, também demonstram que as redes aprendem comportamentos baseados no conhecimento de sistemas mesmo sem uma apresentação anterior às regras.

Não optando pela utilização de redes neurais, algoritmos de construção de modelos lineares são familiares na literatura: *Forward Selection*, *Backward Elimination*, *All Subsets Regression*. Esses e alguns outros algoritmos são usados para gerar um modelo que irá prever uma resposta y com base em medidas covariadas x_1, x_2, \dots, x_i . A qualidade desses modelos é muitas vezes definida em termos de exatidão de predição. Além disso, a presença de variáveis redundantes ou desnecessárias pode ocasionar o sobreajuste do modelo (Massaron e Boschetti, 2016).

Regularização é uma maneira de modificar o papel de variáveis em um modelo de regressão para evitar o sobreajuste e para atingir formas mais simples de funções. Para se ter uma ideia mais clara, são apresentadas principais formas de regularização.

- Ridge (*L2 Regularization*)

A ideia por trás da regularização L2 (*Ridge regression*) é reduzir os coeficientes das variáveis que afetam o modelo. Com isso, a contribuição dessas variáveis não influencia muito no resultado final do modelo. Tal resultado é alcançado com uma modificação na função objetivo que calcula os coeficientes do modelo de regressão, impondo uma penalização que irá depender de quão grande são os coeficientes (Massaron e Boschetti, 2016).

- Lasso-Lars

A regularização LASSO (*Least Absolute Shrinkage and Selection Operator*), também conhecida como regularização L1, adiciona à função objetivo uma penalização absoluta dos coeficientes do modelo. Isso irá selecionar apenas as variáveis informativas, levando à zero as não informativas, trazendo mais clareza e utilizando apenas as variáveis realmente necessárias ao modelo. Esta regularização pode ser acoplada ao algoritmo de regressão LARS (*Least Angle Regression*), que é um compromisso entre a rapidez do *Forward Selection* e a cautela do *Forward Stagewise*. Essa junção cria uma solução estável e não tão propensa ao *overfitting* (Massaron e Boschetti, 2016).

Com relação ao tipo de modelo, não há um teorema unificado e abrangente para qualquer tipo de inferência, sendo que normalmente é feita de maneira *ad hoc*, sujeita à preferência e experiência do usuário (BAKIROV; GABRYS; FAY, 2017).

Entretanto após a geração da inferência, devido aos vários desvios que as medições podem sofrer devido às causas citas anteriormente, tem-se um modelo que pode sofrer uma degradação em sua capacidade preditiva. Sendo ele o cerne de um analisador virtual, são necessárias medidas para avaliar e compensar, ou recalibrar, o modelo. Alguns trabalhos buscam a adaptações de médias móveis e técnicas utilizando PCA e PLS (WANG; KRUGER; IRWIN, 2005)(LIN et al., 2007), outros partem para analisadores virtuais mais complexos utilizando modelos baseados em logica *fuzzy* (MACIAS; ANGELOV; ZHOU, 2006), e ainda existe o *metalearning*, onde metodologias que fazem uma composição de métodos (VILALTA; DRISSI, 2002). Mais detalhes podem ser encontrados na revisão feita por Kadlec & Gabrys (2009). Porém, o próprio autor ressalta que apesar da automação dada pelos recentes trabalhos ressalta que o operador do modelo é o mais importante dado seu conhecimento do processo e planta, ao definir parâmetros cruciais destas metodologias.

Ainda que em posse de um bom modelo para a inferência, as falhas de instrumentos ou sinais corrompidos podem afetar e muito as predições geradas. KOURTI; MACGREGOR (1995) propõem procedimentos estatísticos multivariados principalmente baseados em PCA para monitoramento e diagnóstico de processos industriais. (VENKATASUBRAMANIAN et al., 2003) traz uma revisão extensa sobre os métodos existentes para diagnóstico de falhas e diferentes classificações como mostradas na Figura 2.6.

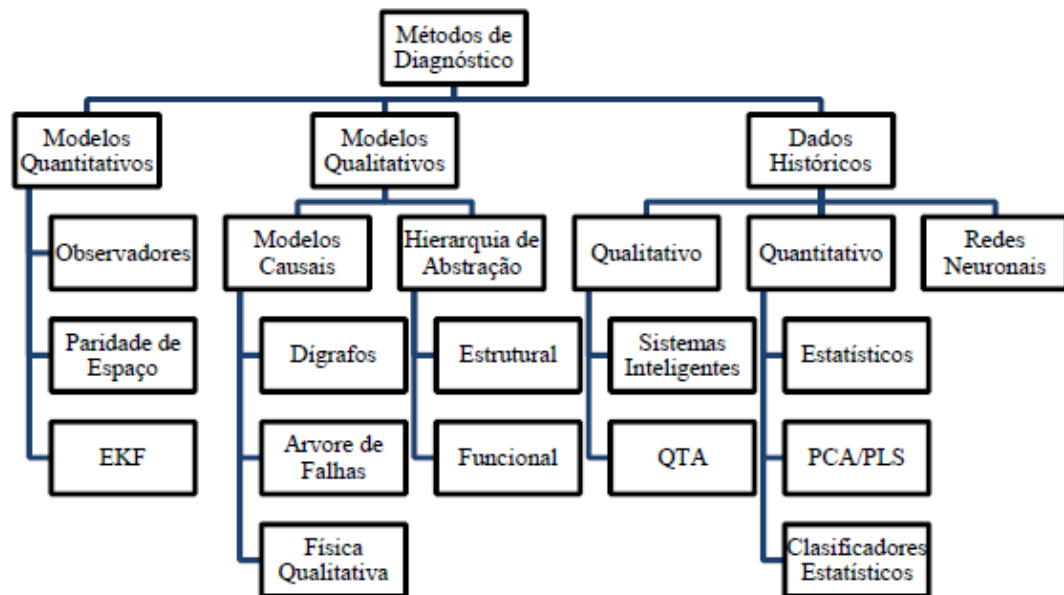


Figura 2.6 Diagrama de classificação entre os diferentes métodos de diagnóstico de falhas.

A terminologia definida por (R.ISERMANN, 1997) descreve quatro níveis diferentes da crescente área de diagnóstico de falhas. São eles, a detecção de quando ocorre a falha, o isolamento do tipo e localização da falha, a identificação do tamanho e comportamento durante a falha e o mais completo diagnóstico, que engloba todas as características anteriores.

Entre as metodologias mais atuais, destaca-se a metodologia de detecção e diagnóstico de falhas baseado em modelos empíricos no subespaço das variáveis de processo –

EMPVSub, foco do trabalho realizado por Bastidas (2018), atingindo todos os níveis de diagnóstico anteriormente comentados.

2.4 Critérios de Avaliação de Modelos

Para comparação entre modelos preditivos uma boa prática é utilizar as mesmas métricas, pois assim faz-se uma análise justa (GREENE, 2014). As métricas utilizadas para comparação dos modelos, tanto do autoencoder treinado quando das inferências geradas a partir dos espaços latentes, serão apresentadas nesta seção.

2.4.1 Erros de reconstrução do autoencoder: RMSE e MSLE

Como critério comparativo são utilizadas as métricas da raiz do erro médio quadrático (*RMSE*, *root mean squared error*) e a erro quadratico médio logarítmico (*MSLE*, *mean squared logarithmic error*) entre os valores de entrada e os valores reconstruídos pelo autoencoder. Caso todas as predições fossem perfeitas, os resíduos teriam valor zero, o coeficiente de determinação seria 1, e tanto o RMSE quanto o MSLE seriam zero por consequência. O coeficiente de determinação faz uma medida da proporção do quanto a variação na variável alvo pode ser explicada pela variação nos regressores, enquanto que o *RMSE* é uma medida do erro de predição. O *MSLE* é da mesma forma, uma medida do erro de reconstrução, mas que trata os erros em valores pequenos da mesma proporcionalmente ao seu tamanho real. Dessa forma penaliza mais o *underfitting*. A forma como calcular estes critérios pode ser vista nas equações a seguir.

$$RMSE = \sqrt{\frac{1}{n} \sum_i (x_i - \hat{x}_i)^2} \quad (2.27)$$

$$MSLE = \frac{1}{n} \sum_i (\log(x_i + 1) - \log(\hat{x}_i + 1))^2 \quad (2.28)$$

onde n representa o número de amostras utilizadas para o cálculo, x_i o valor esperado para a variável e \hat{x}_i o valor reconstruído pelo autoencoder.

2.4.2 Erros de predição da inferência gerada: R² Ajustado, AIC e BIC

Para a geração de inferências, o R² pode ser útil para avaliar o ajuste sobre os dados utilizados na calibração, mas quando o modelo é direcionado à previsão, as informações no âmbito de treinamento podem não ser necessariamente ideias. Sabe-se que, ao adicionar variáveis ao modelo, a variância do erro de previsão pode aumentar, podendo haver uma tendência ao *overfitting*, mesmo que o modelo apresente um melhor ajuste aos dados de treinamento. Com isso, são adotados outros critérios para a avaliação de modelos com o intuito de penalizar aqueles com um maior número de variáveis (GREENE, 2014).

Três critérios serão revisados para avaliação dos modelos: R² ajustado, critério de informação de Akaike (*AIC*) e critério de informação Bayesiano (*BIC*). Os três critérios são dependentes de R², do tamanho da base de dados e do número de parâmetros ajustados. O critério *BIC* penaliza mais fortemente modelos com mais parâmetros. As equações destes critérios podem ser visualizadas subsequentemente. Da mesma forma, é importante ressaltar que o valor do R² ajustado será menor ou igual ao valor do R² e que quanto mais próximo de 1, melhor. Os outros dois critérios podem assumir qualquer valor real e quanto menor esse valor, melhor o modelo (GREENE, 2014).

$$\bar{R}^2 = 1 - \frac{n-1}{n-K} (1 - R^2) = 1 - \frac{n-1}{n-K} \left(\frac{SSE}{\sum_{i=1}^n (y_i - \bar{y})^2} \right) \quad (2.28)$$

$$AIC(K) = s_y^2 (1 - R^2) e^{\frac{2K}{n}} = \log \left(\frac{SSE}{n} \right) + \frac{2K}{n} \quad (2.29)$$

$$BIC(K) = s_y^2 (1 - R^2) n^{\frac{K}{n}} = \log \left(\frac{SSE}{n} \right) + \frac{K \log n}{n} \quad (2.30)$$

onde SSE é a soma dos quadrados dos erros, K é o número de parâmetros e n é o número de amostras.

Capítulo 3 – Metodologia Proposta

Este capítulo apresenta a metodologia proposta para desenvolvimento de inferências utilizando o autoencoder. Toda a metodologia foi implementada em *Python* versão 3.7 e utilizando o pacote *Tensorflow* versão 1.13.1. A sequência de ações a serem tomadas para guiar a produção de inferências a partir de dados estacionários de operação:

1. Coleta de dados e análise dimensional do problema;
2. Normalização dos dados e identificação de *outliers* e dados ausentes;
3. Separação dos dados em grupos de treino, teste e validação;
4. Treinamento do autoencoder e obtenção do espaço latente;
5. Utilização do espaço latente para criação de inferências.

3.1 Análise da Dimensionalidade do Problema

Esta etapa serve apenas como um indicativo da quantidade de variáveis necessárias ao modelo. Não necessariamente será encontrado o número ideal de variáveis, mas já se pode ter uma noção e direcionar o treinamento das redes para um tamanho de espaço latente alvo.

A análise da dimensionalidade do problema busca estimar o número de variáveis que explicam os dados a partir de uma análise dos componentes principais. Os métodos *PCA* e *Kernel-PCA* são úteis neste sentido e podem ser utilizados em situações linearmente separáveis (*PCA* ou *KPCA*) ou situações não linearmente separáveis (*KPCA*) e que se tem uma compreensão da dinâmica do sistema (RHINEHART, 2014).

Exemplificando, dado que o número de principais componentes de um sistema que acumulam 99% da sua variância seja cinco, buscaremos uma arquitetura de autoencoder com 5 neurônios em seu espaço latente.

3.2 Pré-processamento dos Dados

Como bem sabido, a qualidade do modelo é fortemente dependente da qualidade dos dados utilizados. Na etapa de pré-processamento o ideal é averiguar a ocorrência de determinadas situações como congelamento de variáveis, ou valores fisicamente impossíveis, como, por exemplo, vazões negativas. É de praxe remover ou preencher tais valores, através de processos lógicos simples, pois eles não ajudam no desenvolvimento da inferência. Se caso for necessário estimar os valores ausentes, pode-se usar diferentes técnicas de interpolação para estimar os valores faltantes das outras amostras do conjunto de dados, é o mais comum na indústria. Técnicas mais atuais e complexas utilizam inclusive um autoencoder auxiliar para maior assertividade de características que não são correlacionadas e ou binárias/não contínuas como apresentadas no trabalho de Pearson (2002).

A identificação dos estados estacionários se faz necessária para que pontos de transição não sejam interpretados como pontos de operação, levando o aprendizado de máquina generalizar características que não contribuem para criação de inferências sobre o estado da planta (RHINEHART, 2014).

A normalização dos dados deve ser feita levando em consideração as funções de ativação que serão utilizadas. Uma normalização que não está adequada aos limites das funções que os dados serão submetidos, implicará numa perda considerável de desempenho no treinamento, podendo inclusive impedi-lo pela saturação das unidades (NIELSEN, 2015). A normalização Z, que leva a média dos dados a zero e torna seu desvio padrão unitário é amplamente utilizada para treinamento de redes neurais dados recentes avanços em seus otimizadores (CHOLLET, 2017). O que não impede o teste de outras formas de normalização, uma vez que seja consistente. Pode-se buscar uma normalização para um tipo específico de função de ativação ou da base de dados utilizada, por exemplo fazendo um escalonamento utilizando os entre-quartis quando há muita ocorrência de *outliers* (PEDREGOSA et al., 2011).

Para a identificação de *outliers* será utilizado a distribuição T^2 de Hotelling, por se tratar de uma estatística multivariável. Além disso, trata-se de um método bastante difundido na literatura. A remoção destes *outliers* pode ser feita através de um processamento lógico simples. Caso necessário os valores ausentes podem ser estimados, por técnicas de interpolação do conjunto de dados na vizinhança (KADLEC; GABRYS; STRANDT, 2009).

Para segregação dos dados nos grupos de treino, validação e teste, foi utilizada a metodologia *k-rank*, desenvolvida no próprio GIMSCOP com bons resultados para a base de dados utilizada (SANTOS et al., 2019).

3.3 Treinamento do Autoencoder

Feito o pré-tratamento e separação dos dados em grupos, é feito o treinamento do autoencoder. É importante salientar que a rede aqui será treinada para replicar os dados de entrada, ou seja, os pontos de operação, de forma a estabelecer uma estrutura com o autoencoder, capaz de condensar a dinâmica do sistema.

Para tanto é interessante buscar o autoencoder mais simples possível que atenda as necessidades de inferência a serem desenvolvidas. Para tanto a metodologia aqui proposta indica a proposição de um AE, raso quando possível e do tipo *undercomplete* para que haja uma compressão da informação de forma independente de treinamento e de ferramentas auxiliares (*dropout*, inicialização e restrições).

O número de unidades, ou neurônios, na camada interna, que será utilizada para pode ser testado partindo da estimativa de mesmo número de principais componentes de uma prévia análise PCA dos dados a serem utilizados.

Como função objetivo a ser minimizada é recomendada a utilização da clássica equação de mínimos quadrados, ou *MSE* (*mean squared error*), novamente levando em consideração a contiguidade das variáveis e dos dados, e como algoritmo de otimização o ADAM com momentum de Nasterov acoplado, pelo seu desenvolvimento mais recente e ampla capacidade de atingir bons resultados com *backpropagation* (KINGMA; BA, 2015).

Para verificar quais as melhores funções de ativação para compressão e descompressão das informações, foi criado uma lógica para se fazer uma busca exaustiva entre as mais indicadas para tal problema. Tal algoritmo pode ser alterado de forma a fazer buscas mais extensas ou com hiperparâmetros especificados e se encontra no APÊNDICE B.

Tabela 3.1: Hiperparâmetros Utilizados para Treinamento.

Otimizador	Adam com Nasterov Momentum
Função Objetivo	<i>MSE</i>
Função de Ativação	' <i>SELU</i> '; ' <i>RELU</i> '; tanh; sigmoide; linear
Inicialização dos Pesos	<i>Lecun</i> , <i>Glorot e He</i> ; <i>Normal e Uniforme</i>

A verificação do erro de reconstrução, dados os devidos cuidados durante o treinamento, define a capacidade do autoencoder de condensar a dinâmica da planta. Sendo essa a característica principal do método para a utilização do autoencoder como um filtro e também origem de inferências a serem criadas a partir do espaço reduzido, camada central do autoencoder. Tal característica será responsável pela checagem de uma amostra, ou ponto operacional, sendo inferida identificando se é possível utilizá-la para inferência ou se existem anomalias naqueles dados, seja por inconsistência de instrumentação, seja por ser uma condição fora daquelas utilizadas no treinamento da rede neural.

Utilizando o autoencoder treinado, sua estrutura é preservada e seu espaço latente utilizado para construção da inferência da variável desejada.

3.4 Desenvolvimento das Inferências

Para criação da inferência, se utiliza o espaço latente pelo autoencoder criado no passo anterior como um modelo do tipo caixa preta, aqui sendo utilizado uma simples regressão linear. Frisa-se que é possível estabelecer qualquer tipo de relação entre as variáveis alvo e o espaço comprimido, uma vez que o autoencoder é treinado para comprimir o sistema dado.

São muito comuns na indústria as técnicas de PLS e PCR (BAKIROV; GABRYS; FAY, 2017). Podendo ser utilizadas regularizações para analisar se todo espaço latente está sendo utilizado para inferência. Uma vantagem da regularização é que não há a necessidade de manipular o conjunto original dos dados, podendo esta ser usada em inferências online ou modelos preditivos online, sem a intervenção humana (MASSARON; BOSCHETTI, 2016).

Após o desenvolvimento da inferência, o conjunto de dados de teste utilizado para classificar o modelo é duplicado. Ao novo conjunto, é adicionado uma perturbação aleatória de forma a simular a falha de transmissão de um dado ou falha de um instrumento. O algoritmo de adição destes ruídos nos dados está disponível no APÊNDICE D e aleatoriamente seleciona uma determinada porcentagem de pontos de operação, em que a uma variável é aplicado uma função que altera o valor correspondente para 10% a 190% do original.

A metodologia é aplicada aos estudos de caso propostos e explicados no capítulo a seguir.

Capítulo 4 – Estudos de Caso

Com o intuito de testar a utilidade da metodologia proposta, foi escolhido um estudo de caso de uma unidade de separação de propeno. Esta unidade foi modelada com base em uma unidade real em operação, por Schultz (2015) que simulou esta unidade em Aspen Plus para estudos sobre técnicas de *SOC (self-optimizing control)*. Além deste, foram geradas bases de dados utilizando combinações lineares e não lineares de variáveis, apontadas no artigo de (MCCONVILLE, 2008) como relações não-lineares frequentemente encontradas empiricamente em processos relativos à engenharia química.

4.1 Unidade de Separação de Propeno/Propano

A função da unidade possui é a separação dos componentes da corrente de gás liquefeito de petróleo (GLP) e o principal produto a corrente de propeno (C3-) com elevado grau de pureza (99,6%). O processo consiste em uma série de três colunas de destilação que serão respectivamente chamadas de T-01, T-02, T-03. A alimentação é feita na primeira coluna, que separa os compostos pesados, i.e. hidrocarbonetos com quatro ou mais carbonos (C4+), da corrente que sai pelo topo em direção à segunda coluna. Na T-02 é feita a separação dos leves, extraindo pelo topo uma corrente rica em etano (C2). A corrente de fundo que agora consiste em uma mistura rica em propano (C3+) e propeno (C3-) segue para a terceira coluna de destilação, T-03, onde ocorrerá a separação dos compostos, buscando a especificação da corrente de topo, propeno com 99,6% em massa. A última coluna utiliza a corrente de topo como fluido de aquecimento do seu próprio reator após uma etapa de compressão.

A Tabela 4.1 traz a representação dos equipamentos e a Tabela 4.2 mostra os códigos de cada equipamento apresentados no fluxograma simplificado do processo na Figura 5.1.

Tabela 4.1: Lista de representações dos equipamentos da unidade Schultz (2015).




Representação	Descrição
	Limite de bateria – Alimentação da unidade
	Limite de Bateria – Correntes de produtos
	Número para identificação da corrente
AR	Água de resfriamento
CB	Condensado de baixa pressão
DR	Sistema de drenagem
VB	Vapor de baixa pressão

Tabela 4.2: Códigos dos equipamentos da unidade de Schultz (2015)

<i>Código</i>	Descrição
<i>B-01</i>	Bomba de refluxo da coluna T-01
<i>B-02</i>	Bomba de alimentação da coluna T-02
<i>B-03</i>	Bomba de refluxo da coluna T-02
<i>B-04</i>	Bomba de produto de fundo da coluna T-03
<i>C-01</i>	Compressor da corrente de topo da coluna T-03
<i>P-01</i>	Condensador da coluna T-01
<i>P-02</i>	Refervedor da coluna T-01
<i>P-03</i>	Condensador da coluna T-02
<i>P-04</i>	Refervedor da coluna T-02
<i>P-05</i>	Refervedor da coluna T-03
<i>P-06</i>	Condensador da coluna T-03
<i>P-07</i>	Resfriador da corrente de produto especificado
<i>T-01</i>	Coluna de destilação para remoção de C4+
<i>T-02</i>	Coluna de destilação para remoção de etano
<i>T-03</i>	Coluna de destilação para separação propeno/propano
<i>V-01</i>	Vaso de acúmulo de condensado da T-01
<i>V-02</i>	Vaso de acúmulo de produto de topo da T-01
<i>V-03</i>	Vaso de acúmulo de condensado da T-02
<i>V-04</i>	Vaso de acúmulo de condensado da T-03

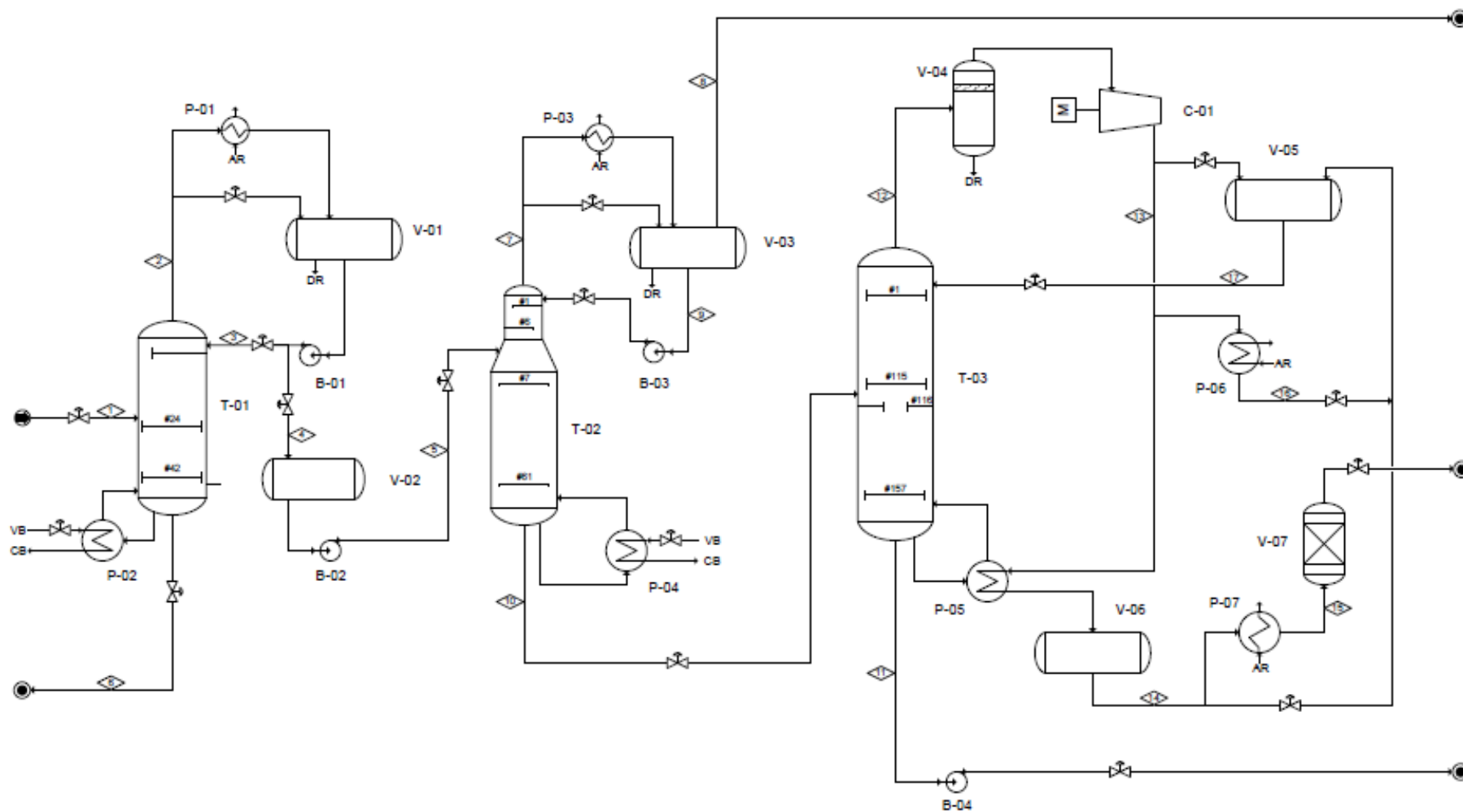


Figura 4.1: Fluxograma simplificado da unidade de separação de propeno extraído de Schultz (2015).

Por se tratar de uma unidade real, o trabalho de Schultz (2015) tem informações e dados construtivos das colunas são omitidos por sigilo industrial. A alimentação da unidade simulada é uma corrente de GLP, com as especificações apresentadas na Tabela 4.3.

Tabela 4.3: Especificação da corrente de alimentação, composta po GLP

Característica (unidade)	Valor
Vazão (kmol/h)	1296,84
Vazão (kg/h)	63000
Temperatura (°C)	66
Pressão (kgf/cm ² g)	17,9

Composto	Valor molar (kmol/h)
Água	1,90
Etano	44,33
Propano	127,75
Propeno	531,20
Isobutano	119,65
Isobuteno	155,35
1-Buteno	83,72
1-3 Butadieno	3,83
Butano	40,82
trans-2-Buteno	103,9
cis-2-Buteno	77,11
Isopentano	3,44
n-Pentano	2,79
Hexano	1,05

4.1.1 Modelagem da Unidade

O modelo estacionário da unidade foi simulado em Aspen Plus versão 7.2 e pode ser visto na Figura 4.2.

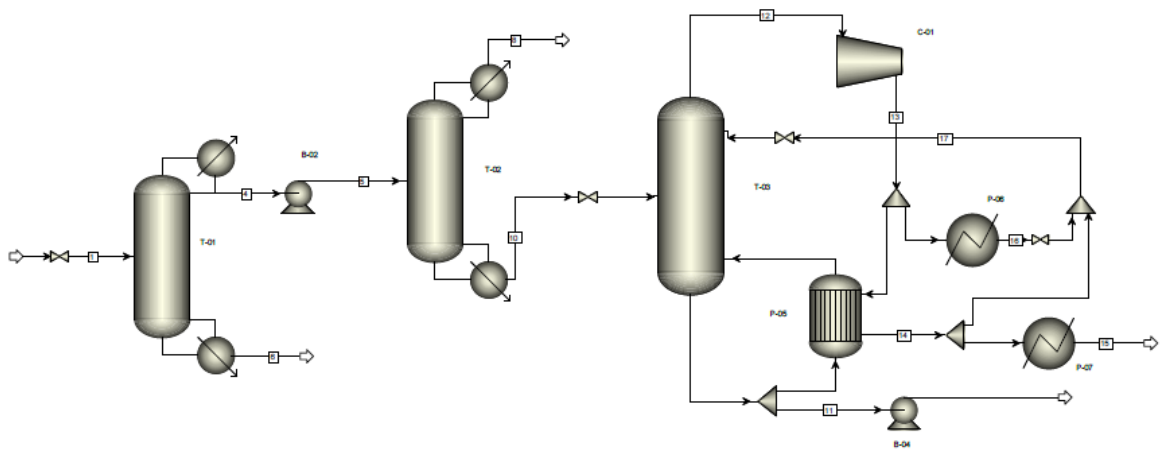


Figura 4.2: Modelo da unidade criado no Aspen Plus

Na modelagem da unidade, as seguintes considerações foram feitas por Schultz (2015):

- Cada coluna de destilação possui dois graus de liberdade estacionários listados abaixo;
- Coluna T-01: Razão de refluxo (RR1) e razão mássica entre a vazão de destilado (corrente 4) e a vazão de entrada (corrente 1), sendo que essa nova variável será chamada de D/F1.
- Coluna T-02: Razão de refluxo (RR2) e a razão mássica entre a vazão de fundo (corrente 10) e a vazão de entrada da coluna (corrente 5), sendo que essa nova variável será chamada de B/F2.
- Coluna T-03: fração da corrente que sai do compressor que será utilizada como fluido de aquecimento do refeedor, que será chamada de FA3. Além disso, será utilizada a fração da corrente que sai do refeedor (corrente 14) que retorna para a coluna como refluxo para a coluna, que será chamada de FR3.
- O distúrbio da unidade consiste na corrente de alimentação com variação na vazão de entrada e na concentração de propeno, sendo que a pressão e a temperatura são controladas;
- A pressão de topo de cada coluna é mantida constante;
- A temperatura de entrada de cada coluna é mantida controlada;
- Os vasos do processo não foram simulados, visto que não influenciam no resultado da simulação estacionária;
- Foi utilizado o modelo termodinâmico de Peng-Robinson para calcular as propriedades físico-químicas das correntes, devido as correntes serem compostas por hidrocarbonetos;

- Foram desconsideradas as perdas de carga das tubulações do processo;
- A alimentação de cada coluna possui pressão constante, controlada por uma válvula, sendo que esta foi modelada de forma a fornecer uma pressão de saída especificada;
- Os trocadores de calor, com exceção do P-05, foram modelados apenas para o cálculo da troca térmica necessária, desconsiderando os limites mecânicos dos equipamentos;
- O trocador P-05 foi modelado como um casco e tubo com coeficiente global de transferência de calor constante no valor de $932 \text{ kcal}/(\text{h} \cdot \text{m}^2 \cdot ^\circ\text{C})$ e área total de 2168 m^2 ;
- O compressor foi considerado isentrópico, calculando a energia requerida para manter uma pressão de descarga especificada.

As variáveis calculadas e o diagrama de entradas e saídas do modelo de cada coluna são apresentados na Figura 5.3.

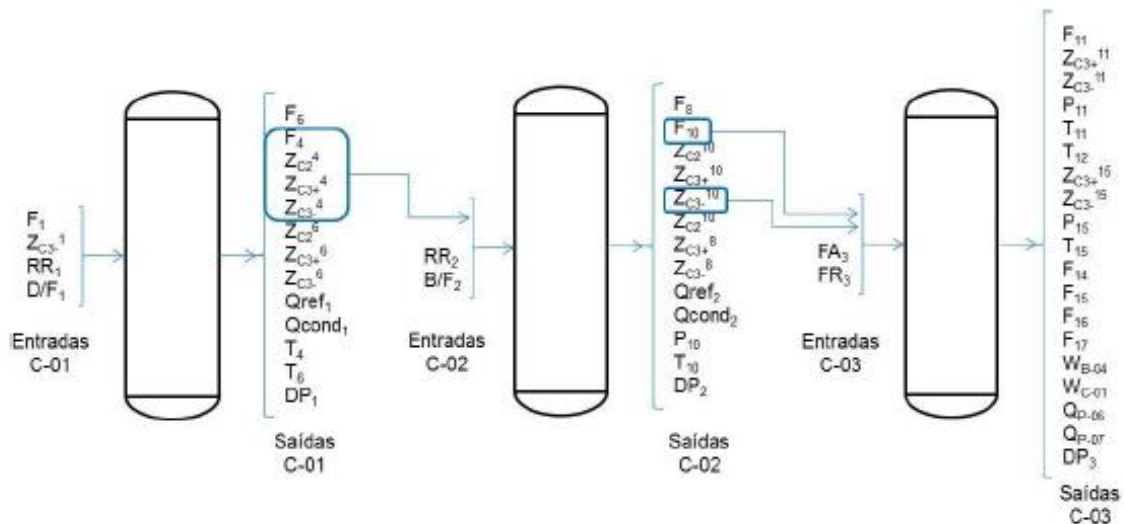


Figura 4.3: Modelo de entradas e saídas do processo

Para facilitar a representação das variáveis do processo, por haver um grande número dessas, foi criada uma notação para representá-las, conforme descrito na Tabela 4.4. Já a Tabela 4.5 descreve as principais correntes do processo, de modo a facilitar a compreensão.

Tabela 4.4: Notação das variáveis do processo.

Notação	Descrição
DP_x	Diferença de pressão entre o topo e o fundo da coluna T-0X
F_x	Vazão mássica da corrente x
F_{m_x}	Vazão molar da corrente x
P_x	Pressão da corrente x
Q_{cond_x}	Calor trocado no condensador da coluna T-0X
Q_{ref_x}	Calor trocado no refeedor da coluna T-0X
Q_x	Calor trocado no trocador P-0X
RR_x	Razão de refluxo da coluna T-0X
T_x	Temperatura da corrente x
W_x	Trabalho realizado no equipamento x
Z_x^y	Fração mássica do componente x na corrente y

Tabela 4.5: Principais correntes da unidade.

Variável	Corrente
F_1	Vazão de alimentação da unidade
F_4	Vazão de topo da coluna T-01
F_6	Vazão de fundo da coluna T-01
F_8	Vazão de topo da coluna T-02
F_{10}	Vazão de fundo da coluna T-02
F_{11}	Vazão de fundo da coluna T-03
F_{14}	Vazão de saída do refeedor da coluna T-03
F_{15}	Vazão de propeno produzido na unidade
F_{16}	Vazão de saída do condensador da coluna T-03
F_{17}	Vazão de refluxo da coluna T-03

SCHUTZ (2015) gerou os dados utilizados aqui como estudo de caso a partir de uma análise de sensibilidade com base nos resultados das simulações do *Aspen*. Para isso o modelo foi dividido em três partes, cada uma representando uma coluna. Cada coluna foi simulada separadamente para um conjunto de dados de entrada, uniformemente espaçados, de forma a se obterem dados com validade em toda a região de operação.

Para a coluna T-01 foi realizada uma variação nos distúrbios da unidade e nos graus de liberdade da coluna (RR_1 e D/F_1). Como distúrbios foram considerados variações na vazão mássica de alimentação (F_1) e na fração mássica de propeno ($Z_{C_3}^{-1}$), sendo que para a alteração da fração molar do propeno foi mantida a mesma proporção da especificação nominal da corrente para os demais componentes. Os valores nominais das variáveis, seus limites são apresentados na Tabela 4.6.

Tabela 4.6: Dados de operação da coluna T-01 gerados por SHULTZ

Varável	Valor nominal	Limites
F_1	63000	59850-66150
Z_{C3-}	0,355	0,319-0,390
RR_1	1,98	1,00-4,00
D/F_1	0,459	0,250-0,650

Total de Pontos: 9000

Os limites de variação da composição de entrada e vazão de alimentação da T-02 foram obtidos de acordo com a saída calculada da coluna T-01. Os valores nominais dessas variáveis, os limites utilizados e o total de pontos são apresentados na Tabela 4.7.

Tabela 4.7: Dados de operação da coluna T-02 gerados por Schultz (2015)

Variável	Valor nominal	Limites
F_5	30064	11656 – 51630
Z_{C2}^5	0,0460	0,0171 – 0,109
Z_{C3-}^5	0,765	0,449 – 0,941
Z_{C3+}^5	0,188	0,0575 – 0,320
RR_2	12,23	8,00 – 15,00
B/F_2	0,896	0,600 – 0,910

Total de Pontos: 414720

Para a coluna T-03, foram utilizadas como entrada do modelo da coluna, a vazão mássica de entrada da coluna (F_{10}), a fração mássica de propeno na entrada (Z_{C3-}^{10}) e os graus de liberdade citados anteriormente, FA_3 e FR_3 , sendo que os limites de variação da composição de entrada e vazão de alimentação da coluna T-03 foram também obtidos a partir dos resultados calculados sobre a coluna T-02. Os valores nominais, os limites e o número de pontos utilizados são apresentados na Tabela 4.8.

Tabela 4.8: Região utilizada para a coluna T-03

Variável	Valor nominal	Limites
F ₁₀	26458	15874 – 37041
Z _{C3} ⁻¹⁰	0,798	0,594 – 0,953
FA ₃	0,873	0,600 – 0,890
FR ₃	0,792	0,642 – 0,942

Total de Pontos: 1764

4.2 Bases de Dados Geradas Artificialmente

Para uma análise mais sistemática e com uma base de dados com comportamento e características não-lineares conhecidas, foram produzidas equações com seis variáveis e de relação pré-estabelecida entre elas.

Foram gerados aleatoriamente 1001 amostras de 4 variáveis diferentes, que serão aqui chamadas de 'x1', 'x2', 'x4' e 'x5'. A partir destes vetores, foram construídos 4 conjuntos de dados, com variáveis ditas de entrada 'x3' e 'x6' e resultante 'y' diferentes, com o intuito de verificar a capacidade da ferramenta, os conjuntos serão chamados de 'LL', 'NL', 'LN' e 'NN' e serão todos identificados e terão as relações entre variáveis demonstradas a seguir.

4.2.1 Bases de Dados LL

A base de dados LL tem tal denominação pela relação linear entre as variáveis de entrada e a saída. Ou seja, 'x3' e 'x6' são combinações lineares das variáveis geradas e 'y' é uma combinação linear destas últimas. A obtenção de 'x3', 'x6' e 'y' é dada pelas equações a seguir.

$$x_3 = 3x_1 - 2x_2 \quad (4.1)$$

$$x_6 = 2x_5 - \frac{1}{2}x_4 \quad (4.2)$$

$$y = x_3 + x_6 \quad (4.3)$$

Feito isto temos uma base de dados tanto com relações lineares entre suas variáveis de entrada como em sua resultante.

4.2.2 Bases de Dados NL

A base de dados NL é construída de forma que haja não-linearidade entre as variáveis de entrada. Ou seja, 'x3' e 'x6' são combinações não-lineares das variáveis geradas e 'y' segue como uma combinação linear destas últimas. Relações apresentadas nas equações a seguir:

$$x_3 = x_1^2 + \frac{3}{10} \log_{10} x_2 \quad (4.4)$$

$$x_6 = x_5 + \left(\frac{1}{2}\right)^{(1+\ln x_4)} \quad (4.5)$$

$$y = 2x_3 + 3x_6 \quad (4.6)$$

4.2.3 Bases de Dados LN

A base de dados LN tem tal denominação pela relação linear entre as variáveis de entrada e a saída sendo resultado de uma combinação não-linear de 'x3' e 'x6'. Ou seja, 'x3' e 'x6' são novamente combinações lineares das variáveis geradas, mas a resultante 'y' construída de forma não-linear como pode ser observada na Equação 5.2.3.3.

$$x_3 = 3x_1 - 2x_2 \quad (4.7)$$

$$x_6 = 2x_1 - \frac{1}{2}x_2 \quad (4.8)$$

$$y = 2^{1+\ln x_3} + 3x_6^2 + \log_{10} x_6 \quad (4.9)$$

4.2.4 Bases de Dados NN

A base de dados NN, segue a mesma lógica e com isto, representa a base de dados construída com relações não-lineares entre suas entradas. Nestes dados, 'x3' e 'x6' são combinações não-lineares de 'x1' e 'x2' e 'x4' e 'x5'. E 'y' é uma construção não-linear destas. Tais relações são apresentadas nas equações abaixo.

$$x_3 = x_1^2 + \frac{3}{10} \log_{10}(x_2)^2 \quad (4.10)$$

$$x_6 = x_5 - \left(\frac{1}{2}\right)^{1+\ln x_4} \quad (4.11)$$

$$y = 2^{(1+\ln x_3)} + 3x_6^2 + 2 \log_{10} x_6 \quad (4.12)$$

Feito isto têm-se agora quatro bases de dados, a todas foi adicionado um ruído branco de cerca de 1% do tamanho das variáveis geradas. O código foi escrito para Python, utilizando as bibliotecas *Numpy* e *Pandas* e pode ser reproduzido para obtenção de bases de dados idênticas excluindo o ruído adicionado e se encontra no APÊNDICE A.

Capítulo 5 – Resultados e Discussão

Neste capítulo será mostrada a aplicação da metodologia proposta nesta dissertação para os estudos de caso apresentados no capítulo anterior.

De maneira análoga a uma unidade industrial, as variáveis de entrada de cada modelo serão utilizadas em conjunto com as variáveis facilmente estimadas, como por exemplo: DP_x (diferença de pressão entre o topo e o fundo da respectiva coluna), RR_x (razão de refluxo da respectiva coluna), Q_{cond_x} (calor trocado no condensador da respectiva coluna), Q_{ref_x} (calor trocado no refeedor da respectiva coluna), W_x (trabalho realizado no equipamento x).

As vazões mássicas e razões entre vazões mássicas também serão utilizadas se necessário, pois essas variáveis possuem alta correlação com as concentrações das correntes, e podem ser medidas.

5.1 Coluna T-01

Para a coluna T-01, a variável chave para ser estimada é a concentração dos pesados na corrente de topo, tida como impureza dessa corrente que seguirá para a coluna T-02. A inferência desta variável é de grande importância para o controle do processo, visto que essa impureza deve se manter no mínimo, considerando as condições termodinâmicas e econômicas da planta. Os valores da média, desvio padrão, mínimo e máximo, quartis dessa concentração, dos dados disponíveis estão dispostos na Tabela 5.1.

Tabela 5.1 Descrição da variável de saída Z_{C4+}^4 (concentração dos pesados no topo da coluna T-01).

Variável	Nº de Amostras	Média	Desvio Padrão	Mínimo	25%	50%	75%	Máximo
Z_{C4+}^4	871	0,1038	0,1520	4,5e-4	7,1e-4	8,7e-4	0,170	0,5255

5.1.1 Pré-Processamento dos Dados da Coluna T-01

O desenvolvimento da inferência para a coluna T-01 se inicia então com o pré-processamento dos dados. Como os dados são oriundos de uma simulação estática, a etapa de detecção de estados estacionários não se faz necessária. Porém a detecção de *outliers* é necessária, uma vez que os dados foram gerados a partir de um modelo ajustado a dados da simulação em *Aspen Plus*. Assim, existe a possibilidade de haver inconsistências.

Sabe-se que são 871 amostras de 10 variáveis, apresentadas na Tabela 5.2, representando a operação da coluna.

Tabela 5.2: Variáveis disponíveis, da coluna T-01, que podem ser utilizadas como entrada para o modelo.

Variável	Descrição
F_1	Vazão mássica de entrada na coluna T-01
D/F_1	Razão mássica entre a vazão de destilado (corrente 4) e a vazão de entrada (corrente 1)
RR_1	Razão de refluxo da coluna T-01
Q_{ref_1}	Calor trocado no refeedor da coluna T-01
Q_{cond_1}	Calor trocado no condensador da coluna T-01
F_4	Vazão mássica de topo da coluna T-01
F_6	Vazão mássica de fundo da coluna T-01
T_4	Temperatura no topo da coluna T-01
T_6	Temperatura no fundo da coluna T-01
DP_1	Diferença de pressão na coluna T-01

A metodologia se inicia com a normalização desses dados, atribuindo média zero e desvio padrão unitário. Em seguida, com os dados normalizados, aplica-se PCA para uma visualização das amostras e já se pode usar a matriz de covariâncias para estimar o vetor T^2 de Hotelling. A variância explicada em cada componente principal, bem como a variância acumulada nos componentes pode ser visualizada na Figura 5.1.

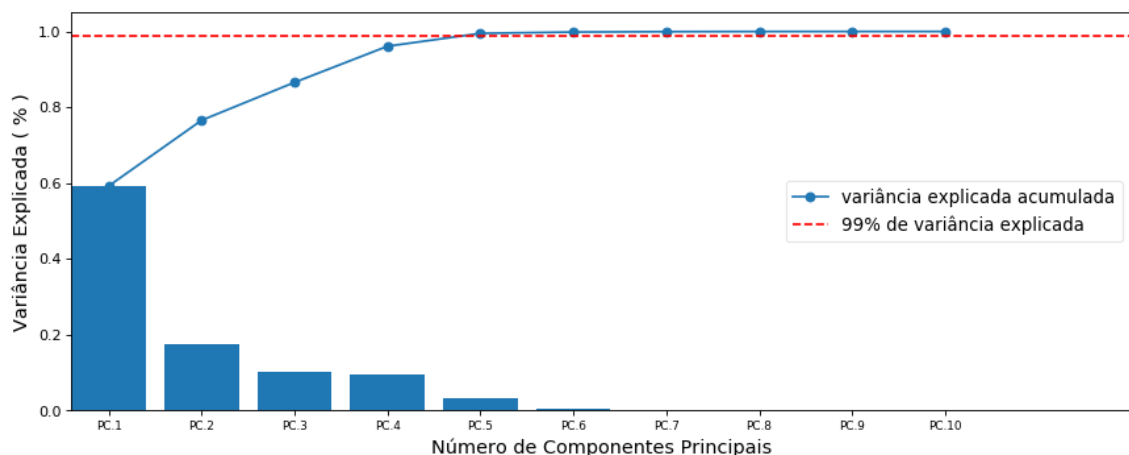


Figura 5.1 Variância explicada acumulada e individual para as variáveis de entrada da coluna T-01.

É possível ver que 95% da variância do sistema pode ser explicada diretamente pelos quatro principais componentes e com a adição do quinto se tem 99% da variância do sistema.

Os *outliers* são identificados através do gráfico de controle T^2 de Hotelling. O cálculo do limite T_{α}^2 foi obtido utilizando uma significância de $\alpha = 99\%$, para a distribuição de Fischer com 871 amostras e 10 variáveis. O resultado pode ser visualizado no gráfico da Figura 5.2.

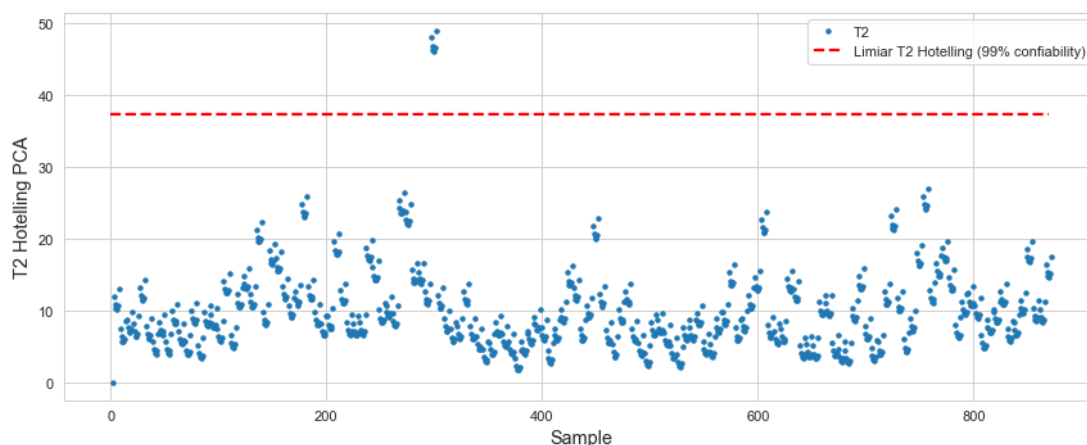


Figura 5.2: Gráfico do T^2 versus as amostras da coluna T-01.

O método selecionou seis pontos bem acima do limite estabelecido pelo método, que foram removidos da base de dados. Desta forma, os dados estão prontos para seguir adiante na metodologia e fazer a seleção dos conjuntos que serão utilizados para calibração e teste. Com a remoção destes seis pontos, utilizando novamente o gráfico de controle T^2 de Hotelling, agora com 865 amostras, o limite que era de 38 passou a ser 25, sendo que mesmo com esta redução não se têm pontos a serem removidos como pode ser visto na Figura 5.3.

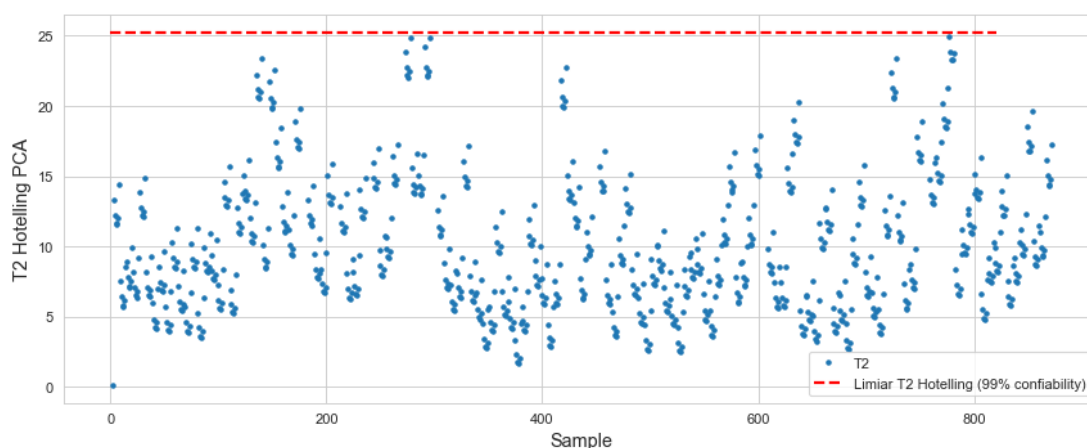


Figura 5.3 Gráfico de controle T^2 estatístico sem amostras acima do limite

A segregação dos dados nos conjuntos de treinamento, validação e teste se dará pelo método *k-rank*, sendo que 60% ficam para treino 20% para validação e 20% para testar o autoencoder em dados não utilizados para treinamento.

5.1.2 Treinando o Autoencoder para a Coluna T-01

Com os dados normalizados e separados, pode-se partir para o treinamento do autoencoder.

Partindo de uma configuração mais simples, com poucos neurônios, para caso necessário adicionar mais complexidade. Foram treinadas redes rasas com 5 unidades na camada interna.

Os autoencoders com menores erros de reconstrução são apresentados na Tabela 5.3, sendo que o primeiro é utilizado a partir daqui.

Tabela 5.3: Melhores Rede Rasa Treinadas

Unidades da Camada Interna	Função de Ativação do Encoder	Função de Ativação do Decoder	RMSE	MSLE
5	linear	linear	0,072143	0,000726
5	RELU	linear	0,072993	0,000801
5	tanh	linear	0,075927	0,000831
5	sigmoide	linear	0,076922	0,000851
5	SELU	linear	0,079069	0,000938

As Figura 5.4 e Figura 5.5 auxiliam na visualização da capacidade de reconstrução do autoencoder trazendo a comparação entre os dados de entrada e os dados reconstruídos pela rede neural. Nelas se tem a projeção dos dados não utilizados no treinamento, grupo de dados de teste, contra o resultado de saída do autoencoder, os dados reconstruídos pela ferramenta.

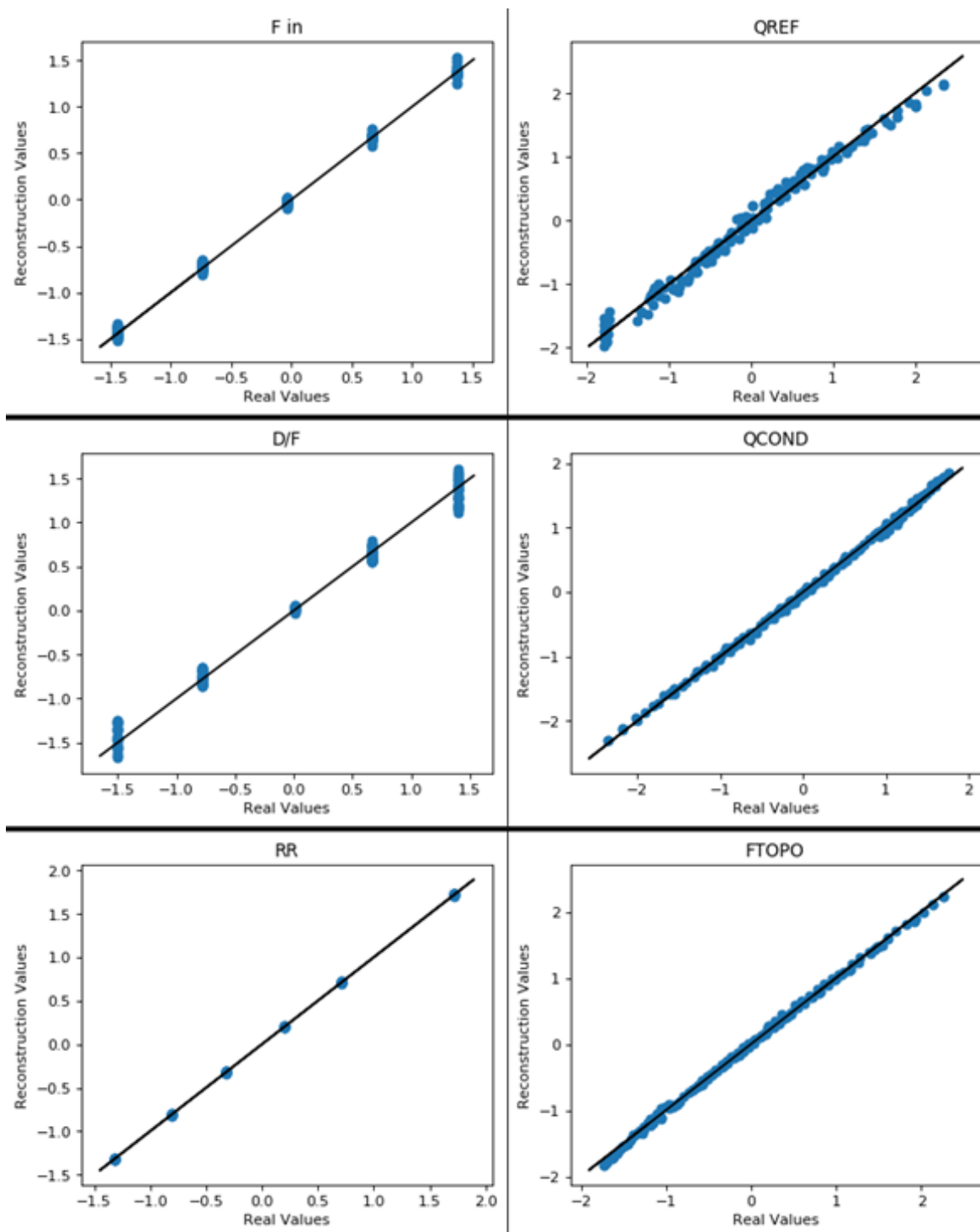


Figura 5.4 Comparativo entre pontos de operação reconstruídos pelo autoencoder e dados de entrada, separados por variável. – coluna T-01

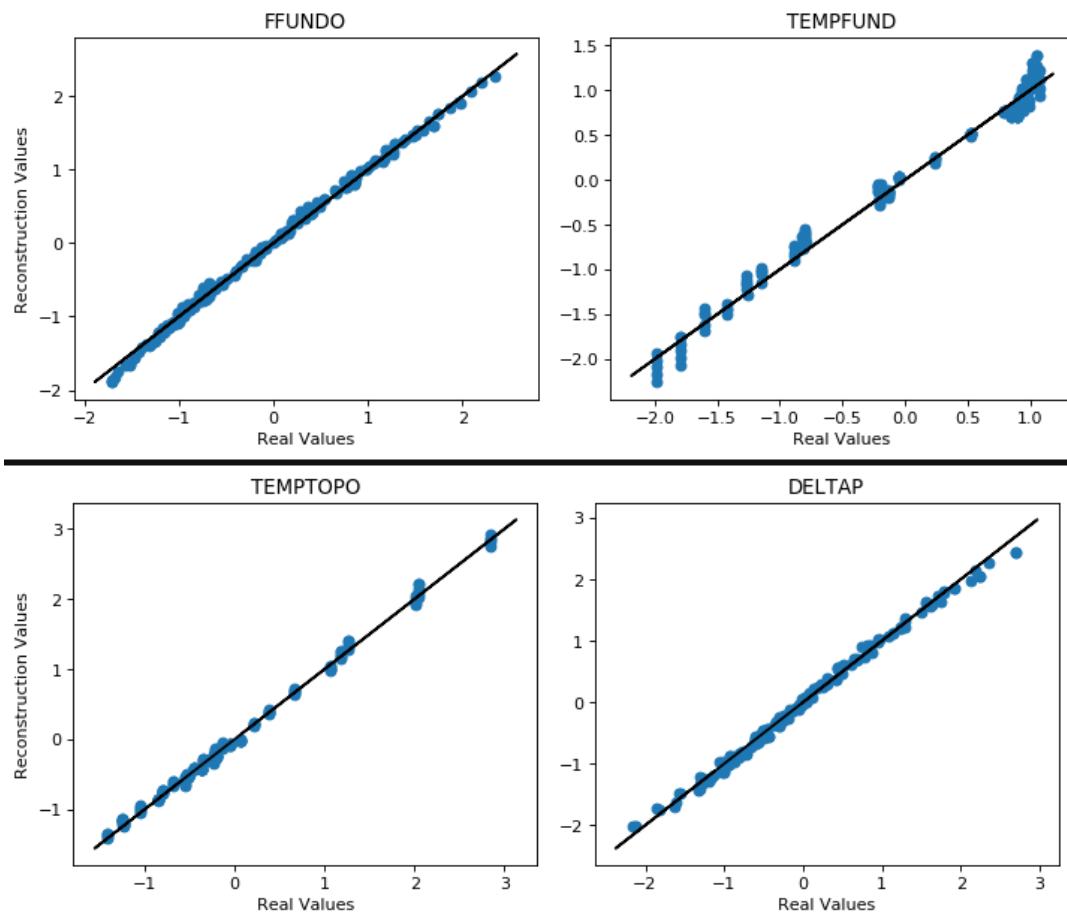


Figura 5.5 Comparativo entre pontos de operação reconstruídos pelo autoencoder e dados de entrada, separados por variável – coluna T-01

Outra forma de visualização do erro de reconstrução pode ser o cálculo da raiz quadrada da média do erro quadrático entre variáveis de um ponto de operação. No caso da coluna T-01 têm-se 10 variáveis que utilizando novamente os dados reservados para teste, produzem o diagrama apresentado na Figura 5.6, que traz demarcada o maior RMSE de reconstrução, o ponto de operação que foi pior reconstruído pelo autoencoder.

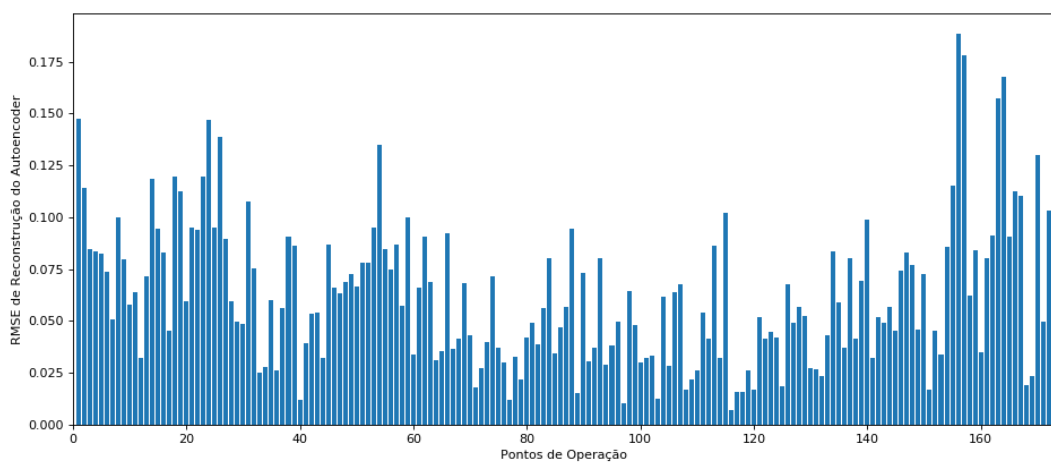


Figura 5.6 Erro de Reconstrução dos dados de teste pelo autoencoder – coluna T-01

5.1.3 Inferindo a Concentração dos componentes pesados (C4+) na Corrente de Topo da Coluna T-01

Da separação dos dados feita com o k -rank, ficaram 173 amostras separadas para teste, estes dados contendo as 10 variáveis de entrada e a variável de saída $y = Z_{c4+}^4$ são utilizados no desenvolvimento de inferências a partir do espaço latente.

Os dados, todas as dez variáveis, passam pela primeira metade do autoencoder, sendo reduzidas a cinco variáveis que serão utilizadas como entrada para nossas inferências. Os dados de entrada passam por metade do autoencoder e as saídas do espaço latente são utilizadas como variáveis para criação da inferência.

São utilizadas as restrições de Ridge e Lasso-Lars para comparativo com as inferências produzidas a partir dos vetores de principais componentes. Entretanto, não houve significativa alteração entre os parâmetros de uma e outra como se pode observar na Tabela 5.4:

Tabela 5.4: Resultados dos critérios de avaliação para o conjunto de teste – inferência dos pesados no topo da coluna T-01.

Método	Quantidade de Variáveis Seleccionadas	R ²	RMSE	MSLE	BIC	AIC
Sem restrição	5	0,9775	0,0226	4,70e-4	-1286,64	16,92
LASSOLARS	5	0,9775	0,0226	4,70e-4	-1286,64	16,92
Ridge	5	0,9774	0,0226	4,71e-4	-1286,01	16,92

Os parâmetros resultantes da regressão sem restrição utilizando o espaço latente para inferir Z_{c4+}^4 no topo da coluna T-01 são apresentados na Tabela 5.5. Junto aos coeficientes determinados temos o desvio padrão e probabilidade de rejeição da hipótese nula.

Para comparação é feita a inferência a partir dos principais componentes. Do gráfico da variância explicada versus os componentes principais da figura 6.1, sabe-se que 94% da variância é explicada através de 4 PCs, e 99% através de 5 PCs. Ou seja, a análise dos componentes principais informa que, embora hajam 10 variáveis, algumas informações são redundantes e correlacionadas.

A partir dos vetores de principais componentes foi feita a regressão linear para estimação da fração mássica de pesados na saída da primeira coluna de destilação, Z_{c4+}^4 . São utilizados também os métodos *Ridge*, e *LASSOLARS*. As métricas, AIC e BIC auxiliam na comparação entre os modelos, penalizando modelos com mais variáveis. Para os dados alocados no conjunto de teste, os resultados estão na Tabela 5.6.

Tabela 5.5: Parâmetros do modelo linear ajustado – inferência dos pesados no topo da coluna T-01.

Parâmetro	Coeficiente	Erros	P(h ₀)
Constante	0,1061	0,002	<0,001
N1	0,1413	0,002	<0,001
N2	0,0713	0,003	<0,001
N3	-0,0789	0,003	<0,001
N4	0,0850	0,002	<0,001
N5	0,0317	0,002	<0,001

Tabela 5.6: Resultados dos critérios de avaliação para os modelos a partir de principais componentes – inferência dos pesados no topo da coluna T-01.

Restrição	Quantidade de Variáveis Seleccionadas	R ²	RMSE	MSLE	BIC	AIC
-	5	0,9775	0,0225	4,69e-4	-1287,23	16,93
LASSOLARS	4	0,9734	0,0245	5,35e-4	-1262,98	14,59
Ridge	5	0,9775	0,0226	4,70e-4	-1286,82	16,93

Os resultados similares aos obtidos através da redução de espaço do autoencoder eram esperados, uma vez que o autoencoder que melhor conseguiu generalizar os dados de treinamento possui funções lineares para a codificação e decodificação das entradas.

Já na Tabela 5.7 são apresentados os parâmetros resultantes da regressão ajustada com a restrição Lasso-Lars, com seus desvios padrão e probabilidade de rejeição da hipótese nula, que reduziu a regressão para quatro variáveis.

Tabela 5.7: Coeficientes da regressão linear utilizando Lasso-Lars – inferência dos pesados no topo da coluna T-01.

Método	Coeficientes	Erros	P(h ₀)
Constante	0,1035	0,003	<0,001
PC1	0,0441	0,001	<0,001
PC2	-0,0046	0,002	0,017
PC3	0	0,003	1
PC4	-0,0522	0,003	<0,001

PC5		0,1381		0,004		<0,001
-----	--	--------	--	-------	--	--------

Para um comparativo visual, mesmo que as inferências tenham tido resultados similares, a Figura 5.7 apresenta os valores reais em função dos valores preditos pelos modelos. Esses gráficos estão dispostos nas figuras a seguir, e a diagonal que corta o gráfico representa a reta em que o valor predito seria idêntico ao valor real, ou seja, quanto mais afastado dessa reta, pior será a predição do modelo.

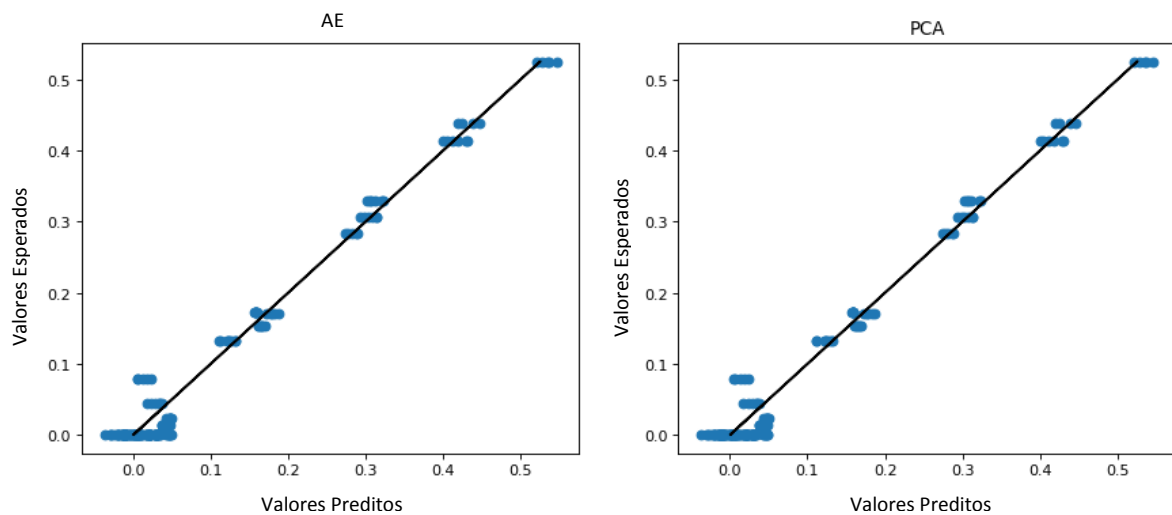


Figura 5.7 Inferências utilizando o espaço latente e componentes principais – inferência dos pesos no topo da coluna T-01.

É notório também que na região onde os dados são muito próximos de zero, o modelo obteve os piores resultados, e, nessa região, os erros relativos são amplificados.

Outra forma de visualização pode é utilizada na Figura 5.8 onde se têm as entradas distribuídas ao longo do eixo. A escolha por tal forma de apresentação dos dados também se deve à sua similaridade com os gráficos de tendência de variáveis de processo utilizados por operadores industriais.

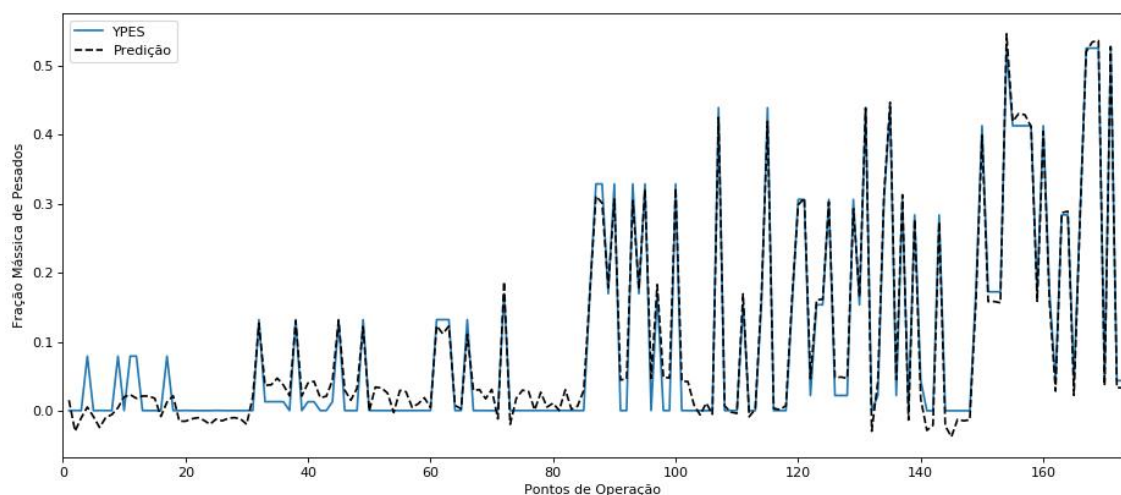


Figura 5.8 Inferências e valores esperados ao longo dos pontos de operação

5.1.4 Efeito do ruído na inferência gerada a partir do espaço latente

A utilidade do modelo se degrada à medida que os fatores citados na seção 2.3 são aplicados, de forma que os dados que alimentam a inferência podem levar a valores com erros de grandeza maior que a esperada.

Para visualização das consequências da utilização de dados com falhas nas inferências, na base de dados de teste uma porcentagem determinada de amostras foi selecionada, sendo novamente escolhida aleatoriamente 1 das 10 variáveis possíveis em que é adicionado um ruído, gerando um dado com valor de 10% ou 190% do valor original.

Utilizando a primeira parte do autoencoder para reduzir a dimensionalidade, o espaço latente é utilizado para inferência da variável desejada, neste caso da coluna T-01, a fração de pesados (Z_{C4+}^4) no topo, determinando a quantidade de impurezas que passam indesejavelmente para a segunda coluna. O comparativo direto pode ser visualizado na Figura 5.9, que apresenta as inferências a partir do encodado e sinaliza os dados alterados posicionando suas marcações no valor esperado.

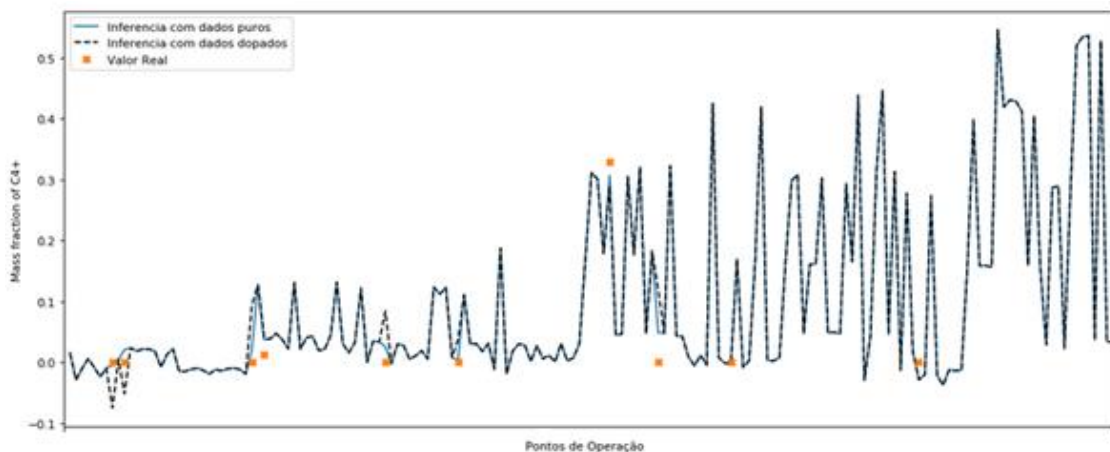


Figura 5.9 Inferências geradas a partir do encodado com dados puros e dados dopados

Essa nova base de dados pode ser reconstruída pelo autoencoder, que foi treinado para generalizar e reconstruir os pontos de operação conhecidos e de operação em estado estacionário. É esperado que as falhas não possam ser reconstruídas e com isso o erro do autoencoder aumente consideravelmente. Tal característica é a chave para a identificação de entradas que estão fora da região de treinamento da inferência.

A Figura 5.10 une as duas informações mostrando no gráfico superior, os valores esperados da variável a ser inferida, em linha contínua azul, e as inferências geradas com a base de dados com falhas, em linha tracejada preta, sinalizando com marcações os pontos alterados. Na parte inferior se têm os erros de reconstrução do autoencoder para cada amostra, com a linha limite indicativa do maior erro de reconstrução dos dados utilizados para treinamento do autoencoder.

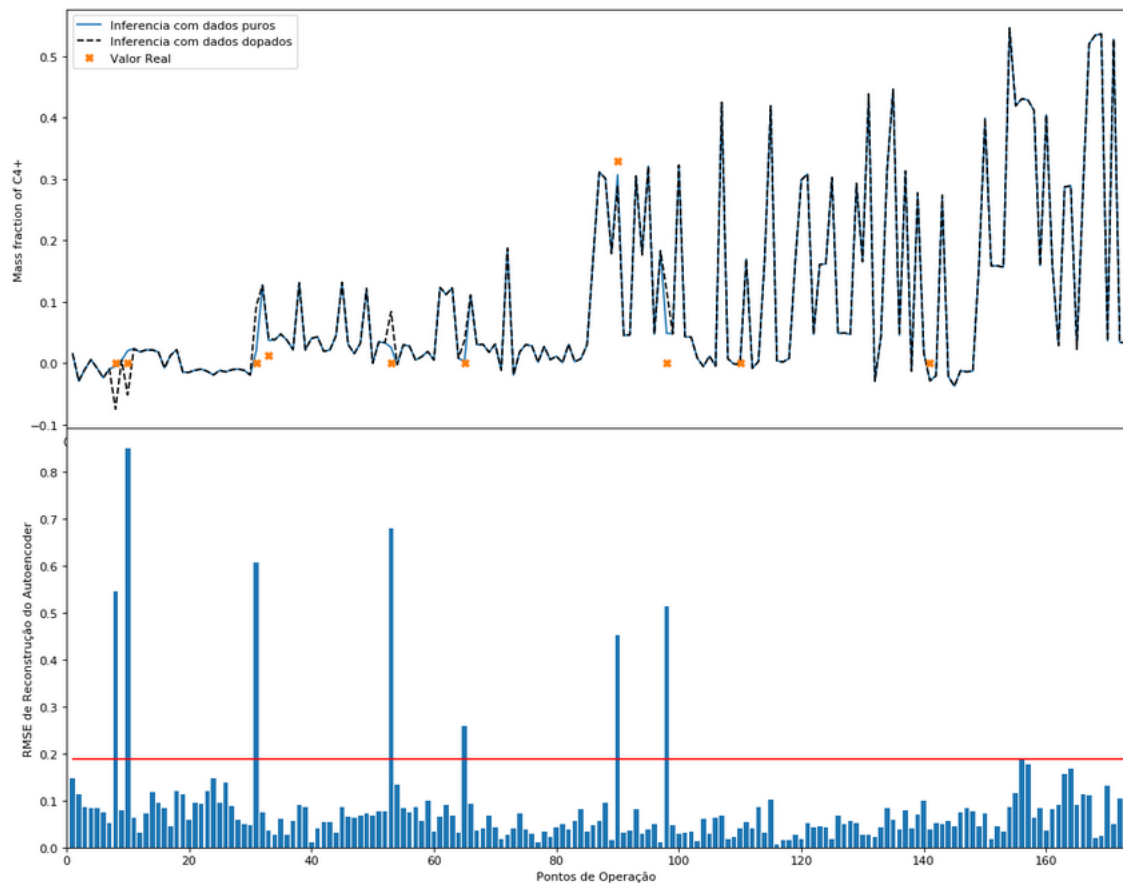


Figura 5.10 Diagrama de inferências e erro de reconstrução dos pontos de operação pelo autoencoder

Pode-se notar claramente que nas amostras em que há distúrbios/falhas no ponto de operação utilizado para predição da fração de pesados, o resultado da reconstrução destes pelo autoencoder ultrapassa o limite estabelecido anteriormente. Podendo assim ser identificado o ponto de operação como um ponto não conhecido pelo autoencoder, uma condição de operação nova, onde não se tem certeza de que o modelo utilizado para inferência da variável terá uma predição com erro condizente com esperado. Ou seja, o erro de reconstrução do autoencoder acusa dados que possivelmente levariam à uma inferência ruim da respectiva variável.

Os resultados obtidos utilizando a redução de espaço feita pelo autoencoder e através de PCA são similares em relação à qualidade das inferências obtidas para a fração de pesados saindo pelo topo da coluna T-01, entretanto o autoencoder possui uma ferramenta complementar de checagem dos dados que se usa em paralelo com o modelo gerado.

5.2 Coluna T-02

Na coluna T-02 a variável de maior interesse para se inferir é a saída de propeno pelo topo da coluna, representando seu desperdício. Portanto, a tarefa aqui é extrair a informação dessa concentração a partir das variáveis de processo disponíveis. Os valores da média, desvio padrão, mínimo e máximo, 1º, 2º e 3º quartis da concentração de propeno no topo da coluna T-02, para os dados disponíveis estão dispostos na Tabela 5.8. Os dados desta coluna, apesar de mais numerosos em relação às outras, já estavam pré-tratados,

escalonados com média unitária, portanto as frações molares abordadas nesta base de dados não obedecem aos limites entre 0 e 1 como usualmente são apresentados.

Tabela 5.8: Descrição da variável de saída Z_{C3-}^8 – (fração mássica de propeno no topo da coluna T-02).

Var	Nº de Amostras	Média	Desvio Padrão	Mínimo	25%	50%	75%	Máximo
Z_{C3-}^8	407038	1	0,2756	0,002	0,8434	1,0549	1,2054	1,5214

5.2.1 Pré-Processamento dos Dados da Coluna T-02

Para a coluna T-02, as variáveis disponíveis que podem ser utilizadas como variáveis de entrada estão dispostas na Tabela 5.9. Já aqui, pode-se concluir que a matriz de entrada terá dimensão de 407038 amostras por 11 variáveis de entrada.

Tabela 5.9: Variáveis da coluna T-02 que podem ser utilizadas como entrada para o modelo.

Variável	Descrição
Z_{C3-}^5	Concentração de propeno na corrente de entrada da coluna T-02
Z_{C3+}^5	Concentração de propano na corrente de entrada da coluna T-02
Z_{C2}^5	Concentração de etano na corrente de entrada da coluna T-02
RR_2	Razão de refluxo da coluna T-02
B/F_2	Razão mássica entre a vazão de fundo (corrente 10) e a vazão de entrada da coluna T-02 (corrente 5)
F_8	Vazão de topo da coluna T-02
F_{10}	Vazão de fundo da coluna T-02
P_{10}	Pressão da corrente de fundo da coluna T-02
T_{10}	Temperatura da corrente de fundo da coluna T-02
Q_{cond_2}	Calor trocado no condensador da coluna T-02
Q_{ref_2}	Calor trocado no refeedor da coluna T-02
DP_2	Diferença de pressão na coluna T-02

Como os dados são simulados e podem possuir erros de simulação, *outliers* devem ser identificados e removidos. A metodologia inicia-se com a normalização dos dados, atribuindo média zero e desvio padrão unitário. Em seguida, com os dados normalizados, aplica-se PCA para uma visualização das amostras e já se pode usar a matriz de covariâncias para estimar o vetor T^2 de Hotelling. A variância explicada em cada componente principal, bem como a variância acumulada nos componentes pode ser visualizada na Figura 5.11.

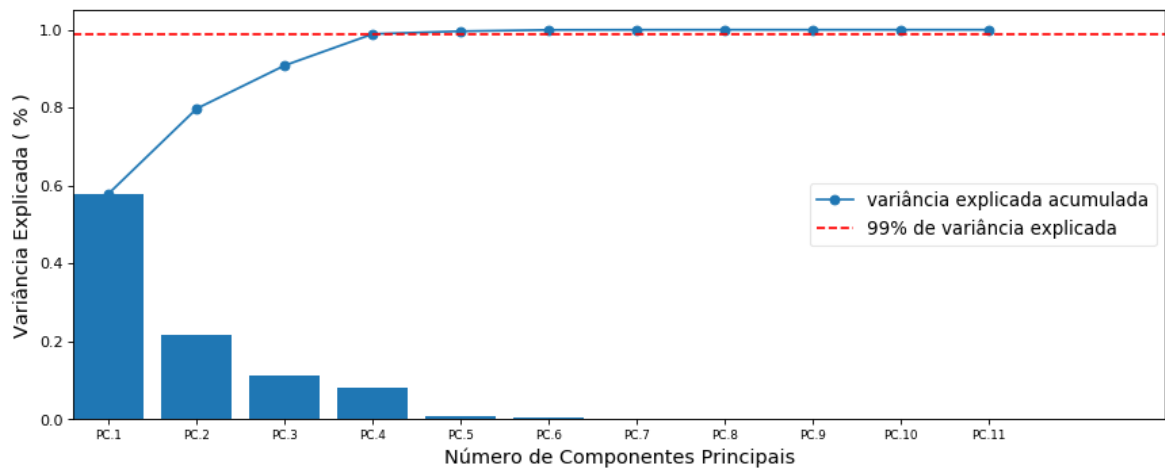


Figura 5.11 Variância explicada acumulada e individual para as variáveis de entrada da coluna T-02.

A variância acumulada dos dados com as cinco principais componentes atinge 99,6%. Foram identificados 4280 *outliers* através do gráfico de controle T^2 de Hotelling. O cálculo do limite T_{α}^2 foi obtido utilizando uma significância de $\alpha = 99\%$ para a distribuição de Fischer. O resultado pode ser visualizado no gráfico da Figura 5.12.

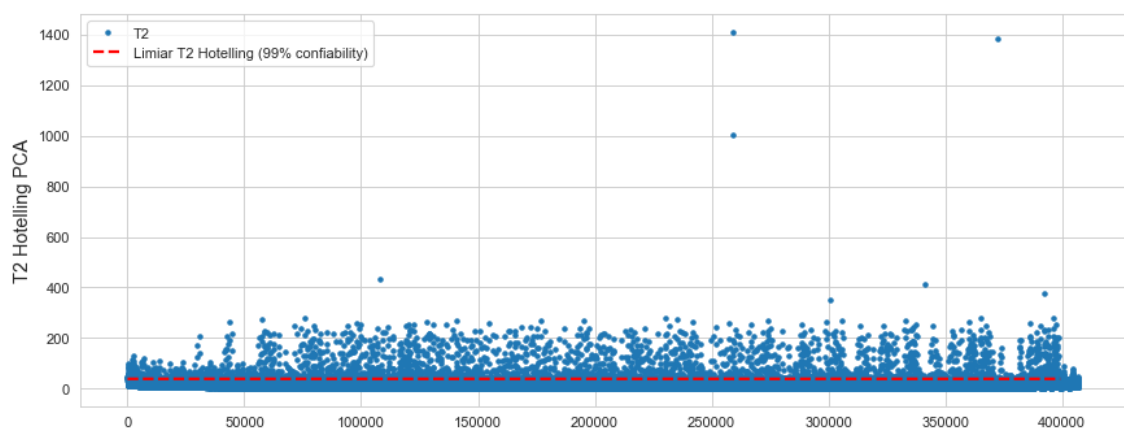


Figura 5.12: Gráfico do T^2 versus as amostras da coluna T-02.

A segregação dos dados nos conjuntos de treinamento, validação e teste é feita através do método *k-rank*, sendo que 60% ficam para treino 20% para validação e 20% para testar o autoencoder em dados não utilizados para treinamento.

5.2.2 Treinando o Autoencoder para a Coluna T-02

Com os dados normalizados e separados, pode-se partir para o treinamento do autoencoder. Foram treinadas, testando os hiperparâmetros apontados no capítulo 2, redes rasas com 5 unidades na camada interna.

Os autoencoders com menores erros de reconstrução são apresentados na Tabela 5.10, sendo que o primeiro é utilizado para compressão do espaço e geração dos dados para inferência da variável alvo do problema.

Tabela 5.10: Melhores Redes Rasas Treinadas – coluna T-02

Unidades da Camada Interna	Função de Ativação do Encoder	Função de Ativação do Decoder	RMSE	MSLE
5	linear	linear	0,061016	5,28e-4
5	SELU	linear	0,061025	5,27e-4
5	RELU	linear	0,061089	5,21e-4
5	tanh	linear	0,062038	5,54e-4
5	sigmoide	linear	0,062092	5,50e-4

Para a coluna T-02 utilizando os dados reservados para teste para cálculo do RMSE de reconstrução, produzem o diagrama apresentado na Figura 5.13, que traz demarcada o maior erro de reconstrução na linha em vermelho.

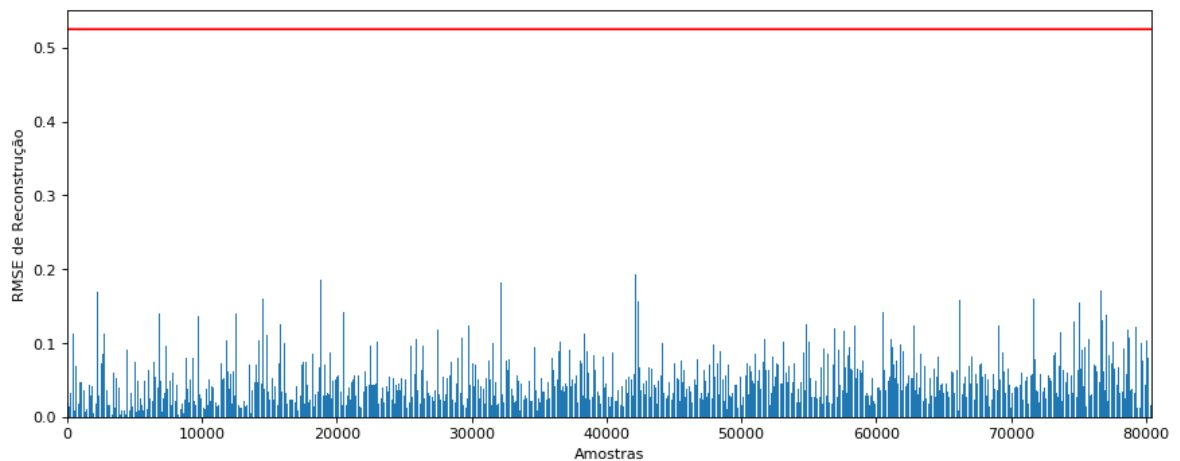


Figura 5.13 Erro de Reconstrução dos dados de teste pelo Autoencoder – coluna T-02

5.2.3 Inferindo a Concentração de propano (C3+) na Corrente de Topo da Coluna T-02

Da separação dos dados feita com o *k-rank*, o conjunto de dados de teste tem 80410 amostras separadas para teste, estes dados contendo as 11 variáveis de entrada e a variável de saída $y = Z_{c3-}^8$ são utilizados no desenvolvimento de inferências a partir do espaço encodado.

Os dados passam pela parte de compressão do autoencoder treinado como entrada para nossas inferências. Os resultados destas inferências geradas podem ser observados na Tabela 5.11.

Tabela 5.11: Resultados dos critérios de avaliação para os modelos utilizando o conjunto de dados de teste – coluna T-02

	Restrição	Quantidade de Variáveis Seleccionadas	R ²	RMSE	MSLE	BIC	AIC
Autoencoder	-	5	0,9945	0,01073	3,214e-5	-729101	7,55
	Lasso-Lars	5	0,9945	0,01073	3,214e-5	-729101	7,55
	Ridge	5	0,9945	0,01073	3,214e-5	-729101	7,55
PCA	-	5	0,9945	0,01077	3,228e-5	-728630	5,53
	Lasso-Lars	5	0,9945	0,01077	3,228e-5	-728630	5,53
	Ridge	5	0,9945	0,01077	3,228e-5	-728630	5,53

Os resultados similares aos obtidos através da redução de espaço do autoencoder era, da mesma forma, esperada uma vez que o autoencoder linear foi o de menor erro de reconstrução.

As inferências foram muito similares e pode se fazer um comparativo visual na Figura 5.14 que apresenta os valores reais em função dos valores preditos pelos modelos.

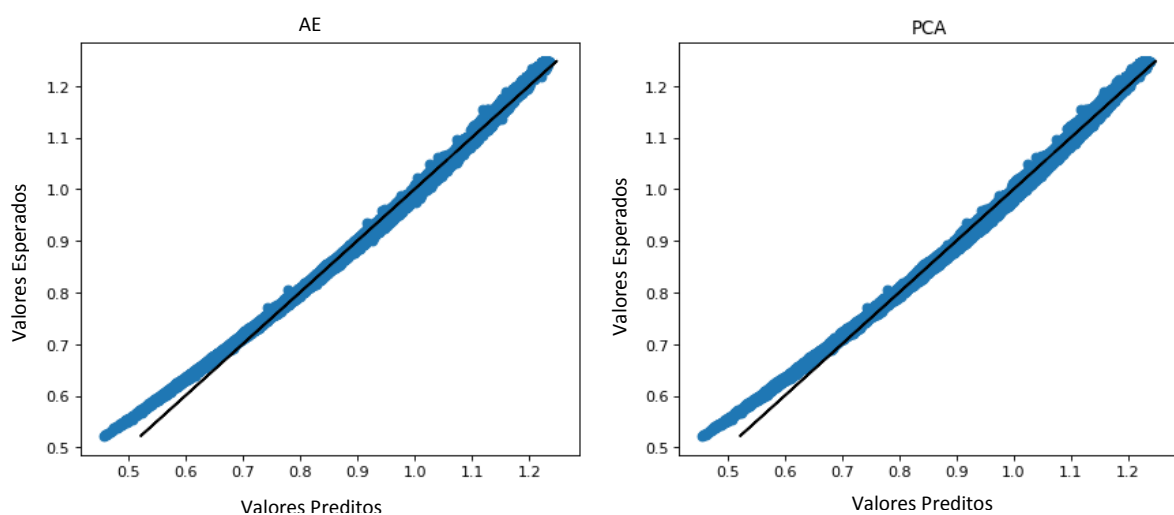


Figura 5.14 Inferências utilizando o espaço encodado e principais componentes – inferência de propeno no topo da coluna T-02

Como são muito numerosas as amostras do conjunto de teste para a base de dados da coluna T-02, uma visualização possível é a apresentada na Figura 5.15, onde se tem um recorte onde aparecem apenas uma parte dos dados, mas assim podemos visualizá-los melhor.

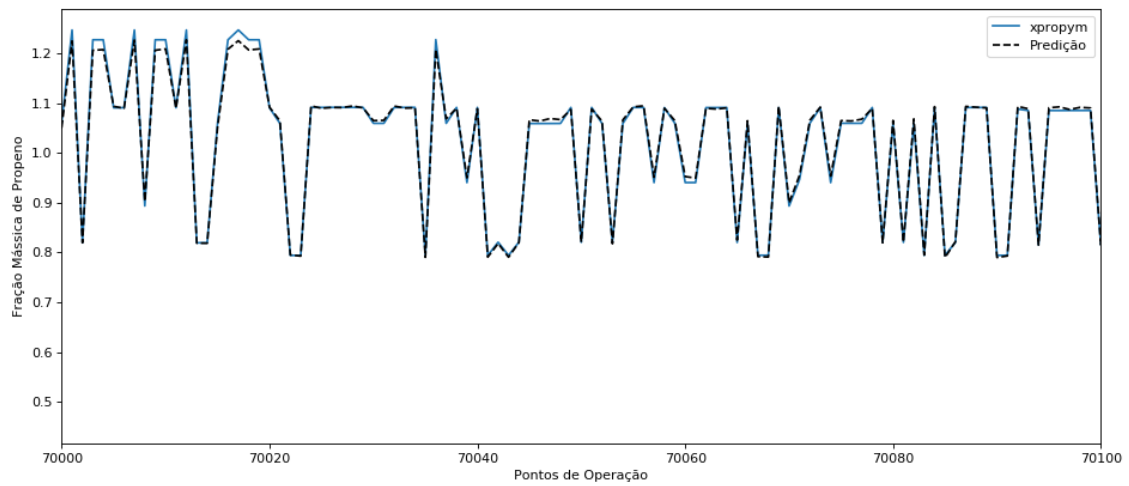


Figura 5.15 Inferências e valores esperados ao longo dos pontos de operação - coluna T-02

5.2.4 Efeito do ruído na inferência gerada a partir do encodado

Para visualização das consequências da utilização de dados com falhas nas inferências, 5% dos pontos de operação tiveram seus valores corrompidos. A Figura 5.16 apresenta um recorte dos dados utilizados.

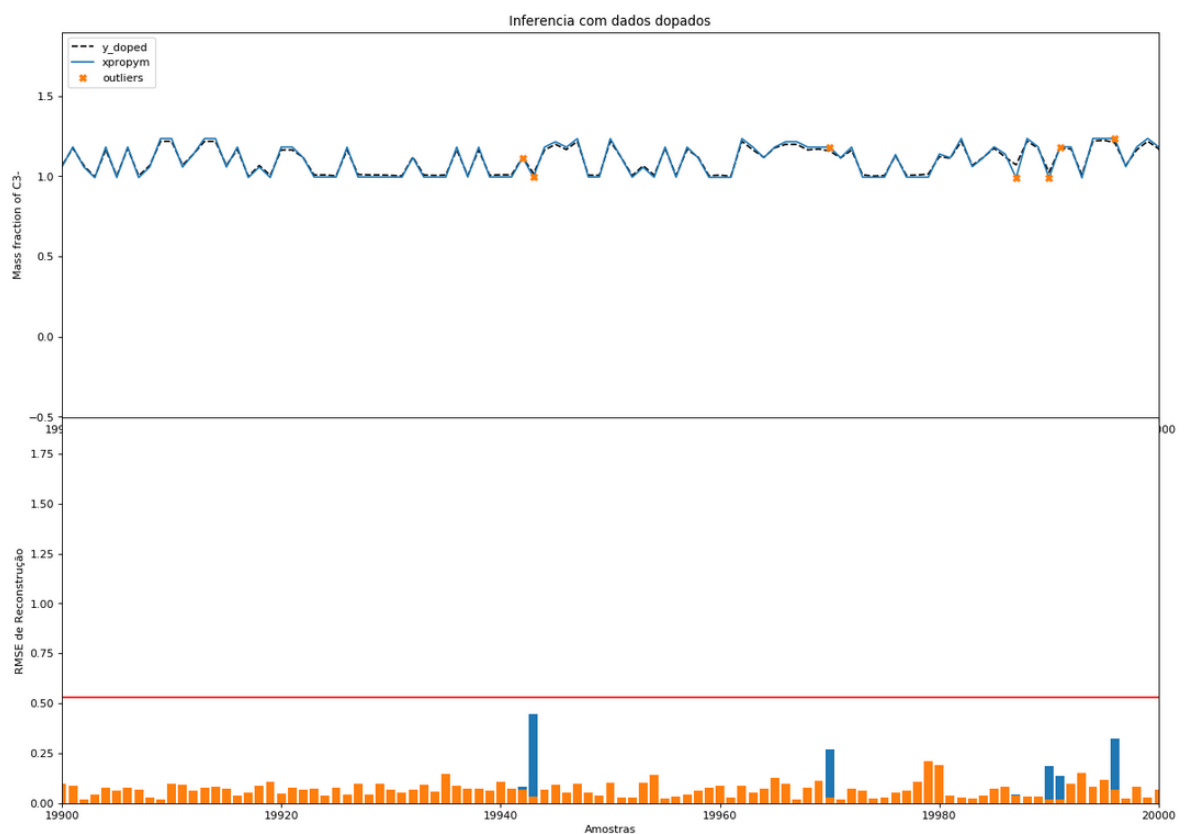


Figura 5.16 Diagrama de inferências e erro de reconstrução dos pontos de operação pelo autoencoder

Os resultados obtidos utilizando a redução de espaço feita pelo autoencoder e através de PCA são similares em relação à qualidade das inferências obtidas para os dados da coluna T-02. Entretanto, não há uma clara relação aqui entre a inferência gerada e o erro de reconstrução do autoencoder.

5.3 Coluna T-03

Na coluna T-03, é importante controlar a especificação da concentração de propeno no topo da coluna. Para isso, faz-se necessário inferir a concentração de propano, componente que torna impura a corrente de propeno. Portanto, a tarefa agora é extrair a informação dessa concentração a partir das variáveis de processo disponíveis. Os valores da média, desvio padrão, mínimo e máximo, 1º, 2º e 3º quartis da concentração de propano no topo da coluna T-03, para os dados disponíveis estão dispostos na Tabela 5.12.

Tabela 5.12: Descrição da variável de saída Z_{C3+}^{15} (fração mássica de propano no topo da coluna T-03).

Variável	Nº de Amostras	Média	Desvio Padrão	Mínimo	25%	50%	75%	Máximo
Z_{C4+}^4	1571	0,1194	0,1161	1,31e-6	3,19e-3	0,0759	0,2131	0,3593

5.3.1 Pré-Processamento dos Dados da Coluna T-03

Para a coluna T-03, as variáveis disponíveis que podem ser utilizadas como variáveis de entrada estão dispostas na Tabela 5.13. A matriz de entrada terá dimensão 1764x14, que são 1764 amostras representadas pelas linhas da matriz versus 19 colunas que são as variáveis de entrada.

Tabela 5.13: Variáveis, da coluna T-03, que podem ser utilizadas como entrada para o modelo.

Variável	Descrição
Z_{C3-}^{10}	Concentração de propeno na corrente de entrada da coluna T-03
FR ₃	Fração da corrente que sai do refeedor (corrente 14) que retorna para a coluna como refluxo para a coluna
FA ₃	Fração da corrente que sai do compressor que será utilizada como fluido de aquecimento do refeedor
F ₁₁	Vazão de fundo da coluna T-03
P ₁₁	Pressão da corrente de fundo da coluna T-03
F ₁₅	Vazão de topo da coluna T-03
T ₁₅	Temperatura da corrente de topo da coluna T-03
F ₁₄	Vazão que sai do refeedor (corrente 14) e retorna para a coluna como refluxo
F ₁₄	Vazão que sai do condensador (corrente 16) e retorna para a coluna como refluxo
W _{C-01}	Energia requerida pelo compressor (C-01)
W _{B-04}	Energia requerida pela bomba (B-04)
Q _{P06}	Energia requerida pelo condensador P-06
Q _{P07}	Energia requerida pelo trocador P-07

T_{TOPO}	Temperatura no topo da coluna T-03
T_{FUNDO}	Temperatura no fundo da coluna T-03
P_{FUNDO}	Pressão no fundo da coluna T-03
V_{REF}	Fração vaporizada que sai do refeedor

A metodologia se inicia com a normalização desses dados, atribuindo média zero e desvio padrão unitário. Em seguida, com os dados normalizados, aplica-se PCA para uma visualização das amostras e já se pode usar a matriz de covariâncias para estimar o vetor T^2 de Hotelling. A variância explicada em cada componente principal, bem como a variância acumulada nos componentes pode ser visualizada na Figura 5.17.

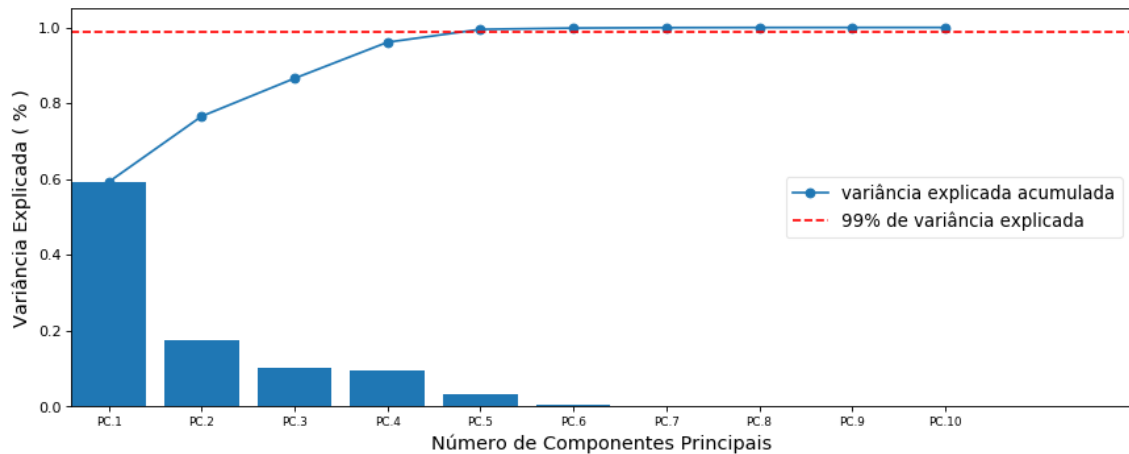


Figura 5.17 Variância explicada acumulada e individual para as variáveis de entrada da coluna T-03.

É possível ver que 95% da variância do sistema pode ser explicada diretamente pelos quatro principais componentes e com a adição do quinto se tem 99% da variância do sistema.

Os *outliers* são identificados através do gráfico de controle T^2 de Hotelling. O cálculo do limite T_{α}^2 foi obtido utilizando uma significância de $\alpha = 99\%$, para a distribuição de Fischer com 1571 amostras e 19 variáveis. O resultado pode ser visualizado no gráfico da Figura 5.18.

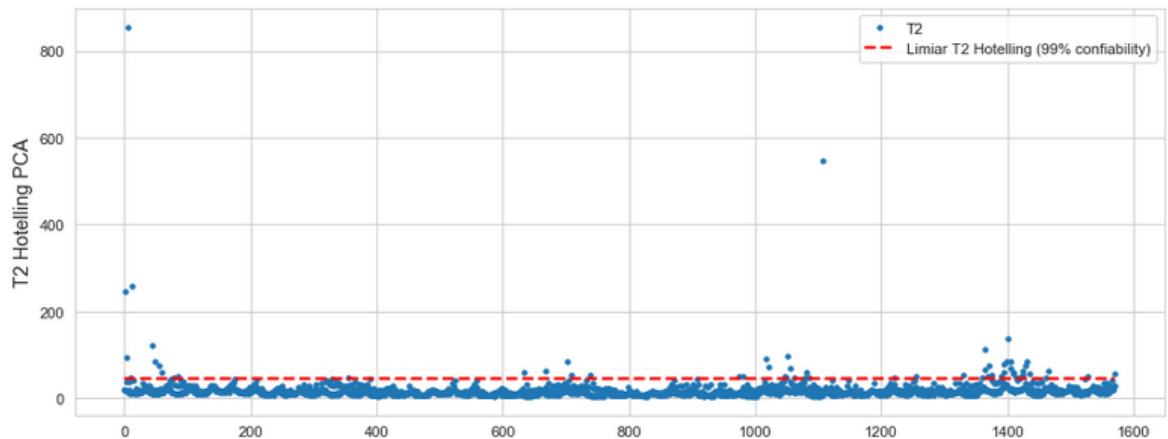


Figura 5.18: Gráfico do T^2 versus as amostras da coluna T-03.

O método selecionou 64 pontos acima do limite estabelecido pelo método, que foram removidos da base de dados. Desta forma, os dados estão prontos para seguir adiante na metodologia proposta e fazer a seleção dos conjuntos que serão utilizados para calibração e teste.

A segregação dos dados nos conjuntos de treinamento, validação e teste é feita através do método k -rank, sendo que 60% ficam para treino 20% para validação e 20% para testar o autoencoder em dados não utilizados para treinamento.

5.3.2 Treinando o Autoencoder para a Coluna T-03

Com os dados normalizados e separados, pode-se partir para o treinamento do autoencoder. Foram treinadas, testando os hiperparâmetros apontados na metodologia, redes rasas com 7 unidades na camada interna.

Os autoencoders com menores erros de reconstrução são apresentados na Tabela 5.14, sendo que o primeiro é utilizado para compressão do espaço e geração dos dados para inferência da variável alvo do problema.

Tabela 5.14: Melhores Redes Rasas Treinadas – coluna T-03

Unidades da Camada Interna	Função de Ativação do Encoder	Função de Ativação do Decoder	RMSE	MSLE
7	linear	linear	0,09173	0,001661
7	RELU	linear	0,09187	0,001659
7	tanh	linear	0,09222	0,001643
7	SELU	linear	0,09225	0,001681
7	sigmoide	linear	0,09242	0,001658

Para a coluna T-03 utilizando os dados reservados para teste para cálculo do RMSE de reconstrução, produzem o diagrama apresentado na Figura 5.19, que traz demarcada o maior erro de reconstrução na linha em vermelho.

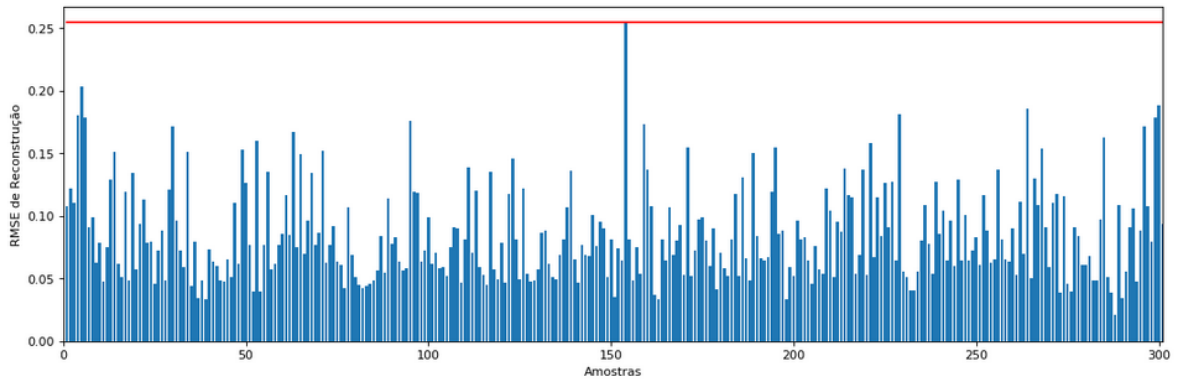


Figura 5.19 Erro de Reconstrução dos dados de teste pelo Autoencoder

5.3.3 Inferindo a Concentração de propano (C3+) na Corrente de Topo da Coluna T-03

Da separação dos dados feita com o k -rank, o conjunto de dados de teste tem 301 amostras separadas para teste, estes dados, contendo as 10 variáveis de entrada e a variável de saída $y = Z_{C3+}^{15}$, são utilizados no desenvolvimento de inferências a partir do espaço encodado.

Os dados, todas as dezenove variáveis, passam pela primeira metade do autoencoder, sendo reduzidas a treinamento como entrada para nossas inferências. E são utilizadas as restrições de Ridge e Lasso-Lars para comparativo com as inferências produzidas a partir dos vetores de principais componentes. Entretanto, não houve significativa alteração entre os parâmetros de uma e outra como podemos observar na Tabela 5.15:

Tabela 5.15: Resultados dos critérios de avaliação para os modelos utilizando o conjunto de dados de teste – coluna T-03

	Restrição	Quantidade de Variáveis Seleccionadas	R ²	RMSE	MSLE	BIC	AIC
Autoencoder	-	7	0,9954	0,0083	5,584e-5	-2848,67	23,82
	Lasso-Lars	7	0,9954	0,0083	5,584e-5	-2848,67	23,82
	Ridge	7	0,9954	0,0083	5,664e-5	-2845,03	23,79
PCA	-	7	0,9958	0,0079	5,094e-5	-2876,60	24,00
	Lasso-Lars	7	0,9958	0,0079	5,094e-5	-2876,60	24,00
	Ridge	7	0,9958	0,0079	5,093e-5	-2877,04	24,00

Os resultados similares aos obtidos através da redução de espaço do autoencoder eram, da mesma forma, esperados uma vez que o autoencoder que melhor conseguiu generalizar os dados de treinamento possui funções lineares para a codificação e decodificação das entradas.

Para um comparativo visual, mesmo que as inferências tenham tido resultados similares, a Figura 5.20 apresenta os valores reais em função dos valores preditos pelos modelos. Esses gráficos estão dispostos nas figuras a seguir, e a diagonal que corta o gráfico representa a reta em que o valor predito seria idêntico ao valor real, ou seja, quanto mais afastado dessa reta, pior será a predição do modelo.

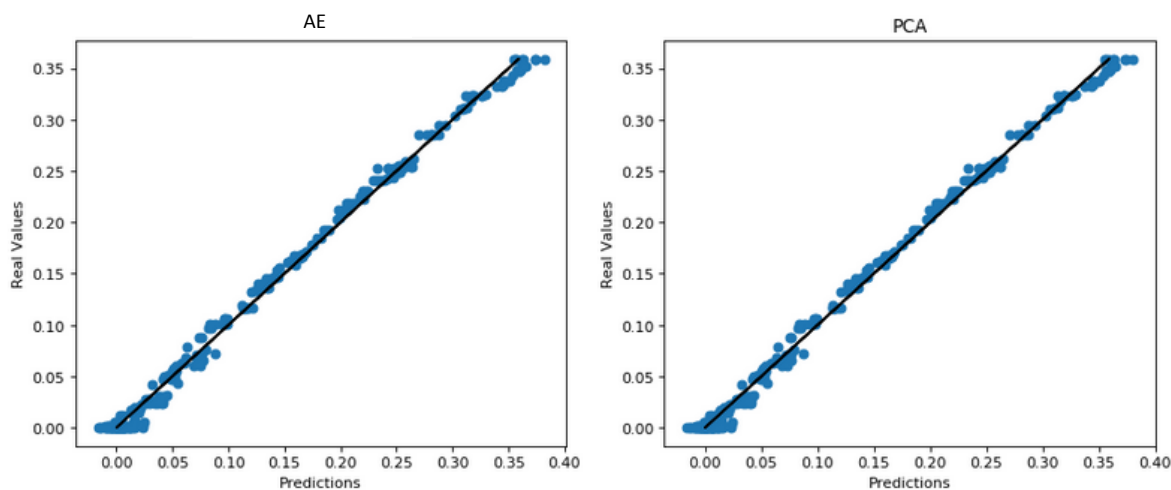


Figura 5.20 Inferências utilizando o espaço encodado e principais componentes – inferência de propano no topo da coluna T-03.

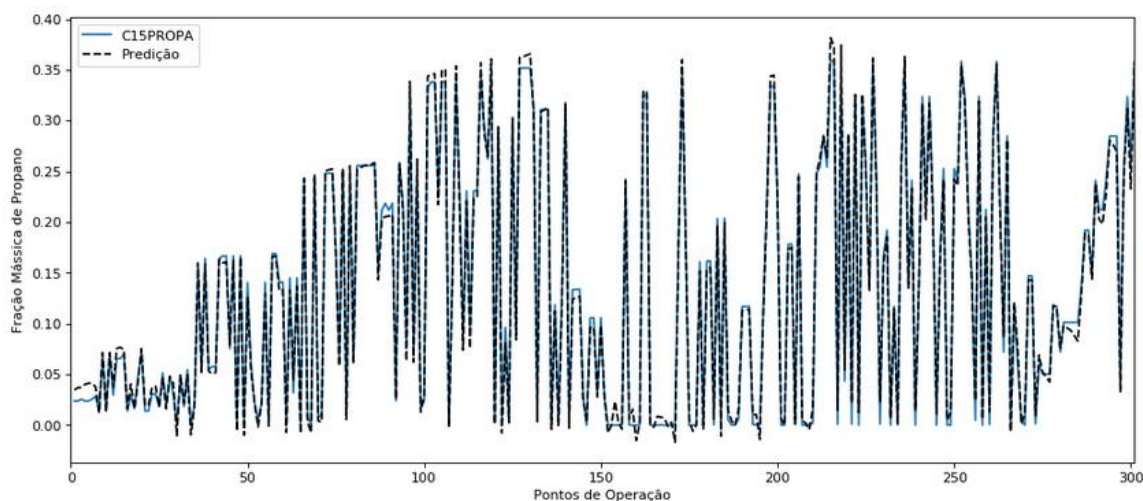


Figura 5.21 Inferências e valores esperados ao longo dos pontos de operação – coluna T-03

5.3.4 Efeito do ruído na inferência gerada a partir do encodado

Para visualização das consequências da utilização de dados com falhas nas inferências, 18 pontos de operação tiveram seus valores corrompidos e com o auxílio da Figura 5.22

podemos observar a correlação entre o erro de reconstrução do autoencoder e a inferência obtida através da regressão utilizando o espaço comprimido.

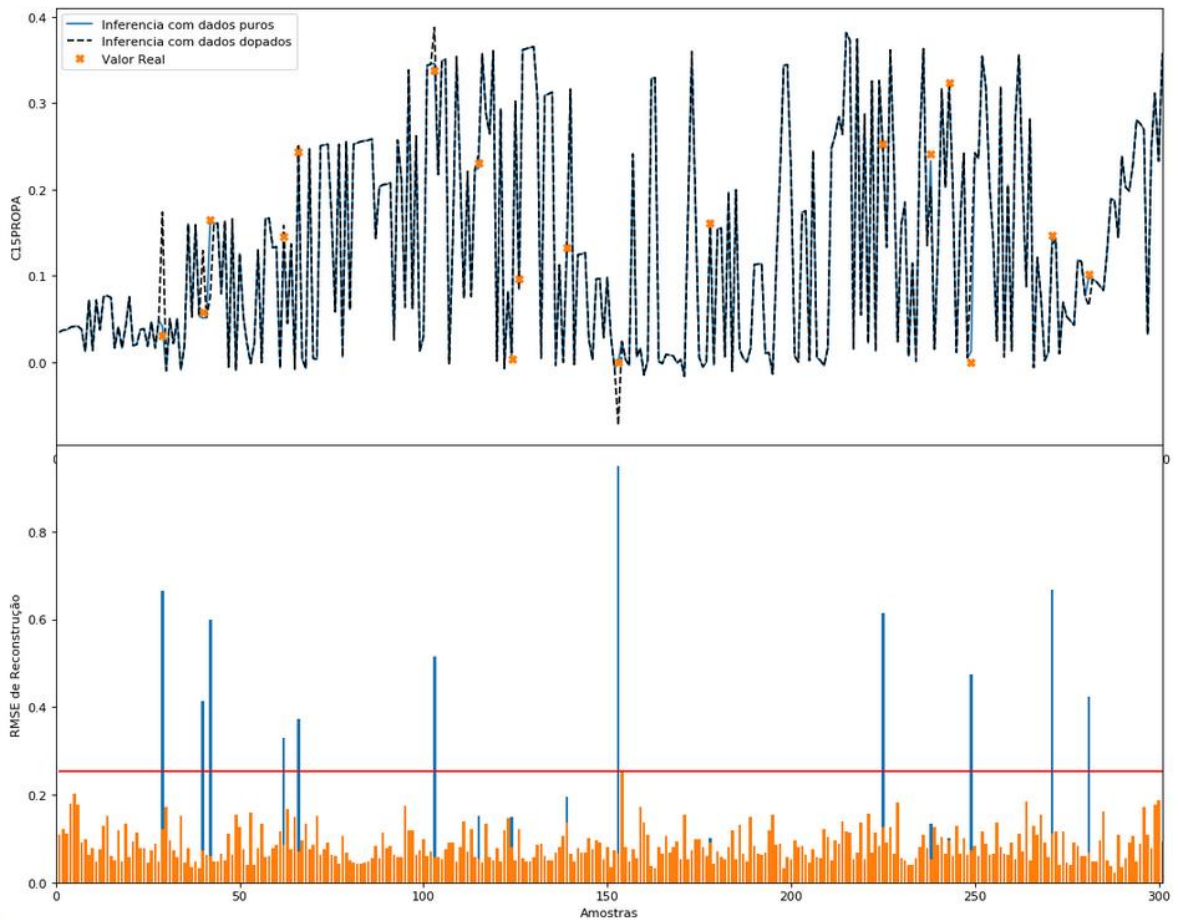


Figura 5.22 Diagrama de inferências e erro de reconstrução dos pontos de operação pelo autoencoder

Os resultados obtidos utilizando a redução de espaço feita pelo autoencoder e através de PCA são similares em relação à qualidade das inferências obtidas para os dados da coluna T-03. Apesar disso, não há uma clara relação entre os erros de reconstrução e os erros de inferência para os dados em estudo. Em pontos em que há um erro de reconstrução pelo autoencoder maior que o limite estabelecido não se pode afirmar que o modelo terá uma boa predição da variável inferida, no caso, da fração mássica de propano no topo da coluna T-03.

5.4 Base de dados artificiais

As bases de dados geradas como estudo de caso possuem relação entre variáveis conhecidas e já expostas no capítulo anterior. Os resultados serão sumariamente expostos e discutidos, separando-os um a um.

5.4.1 Base de dados LL

O conjunto de dados LL é composto então por 1001 amostras de seis variáveis (x_1 , x_2 , x_3 , x_4 , x_5 e x_6) em que há uma dependência linear entre suas variáveis e sua combinação linear tem como resultante a variável alvo y . A variável alvo é descrita na Tabela 5.16 e dado que ela apresenta valores menores que zero a métrica de comparação do erro logarítmico médio (MSLE).

Tabela 5.16: Descrição da variável de saída y

Variável	Nº de Amostras	Média	Desvio Padrão	Mínimo	25%	50%	75%	Máximo
y	1001	1,2137	1,1480	-2,0006	0,3651	1,1988	2,0726	4,3363

O conjunto com relações lineares entre variáveis e variável alvo sendo combinação linear das demais variáveis, na análise de dimensionalidade, apresentou conforme a Figura 5.23 a possibilidade de redução, utilizando PCA, de 6 variáveis para 4.

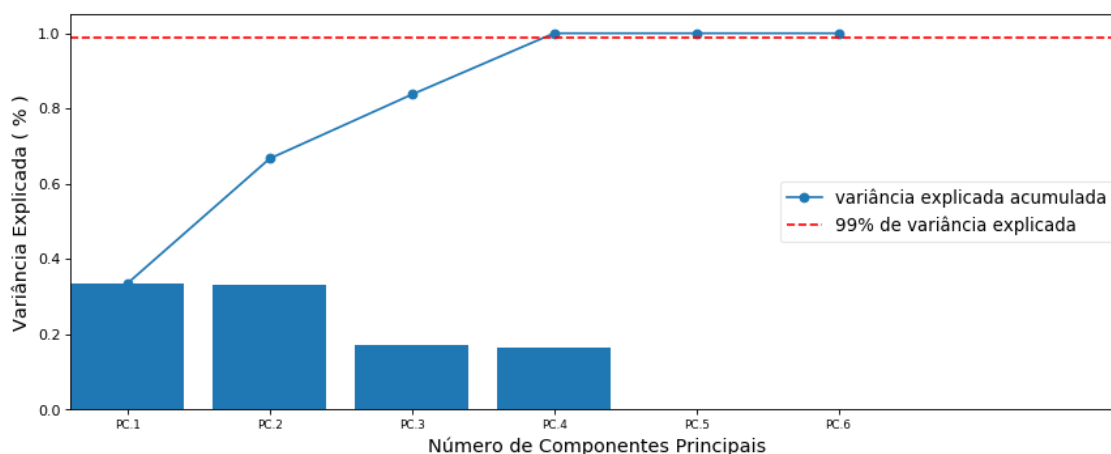


Figura 5.23 Variância explicada por cada principal componente do PCA.

Dentre as redes treinadas a de melhor capacidade de reconstrução dos dados de entrada e utilizada para geração de inferências foi a que utilizou a função de ativação RELU na compressão da informação e descomprimiu o espaço latente com uma função linear. Os melhores autoencoders e suas métricas são dispostos na Tabela 5.17 e em seguida, na Tabela 5.18 são mostrados os resultados das regressões lineares.

Tabela 5.17: Melhores Rede Rasa Treinadas

Unidades da Camada Interna	Função de Ativação do Encoder	Função de Ativação do Decoder	RMSE	MSLE
4	RELU	linear	0,0056	0,6E-5
4	Linear	linear	0,0070	1,1E-5
4	Tanh	linear	0,0176	3,6E-5
4	SELU	linear	0,0200	4,8E-5

4	linear	SELU	0,0957	22,0E-5
---	--------	------	--------	---------

Tabela 5.18: Resultados dos critérios de avaliação para os modelos utilizando o conjunto de dados de teste.

	Restrição	Quantidade de Variáveis Seleccionadas	R ²	RMSE	BIC	AIC
Autoencoder	-	4	0,9999	0,0069	-1971,93	19,38
	LASSOLARS	4	0,9999	0,0069	-1971,93	19,39
	Ridge	4	0,9999	0,0086	-1882,78	18,50
PCA	-	4	0,9999	0,0069	-1969,12	19,36
	LASSOLARS	2	0,9979	0,0508	-1181,38	7,37
	Ridge	4	0,9999	0,0069	-1968,83	19,36

A diferença entre as previsões feitas pela regressão a partir do encodado e a partir dos principais componentes pode ser visualizada na Figura 5.24. A primeira sendo a regressão sem restrições a partir do espaço comprimido pelo autoencoder e a segunda a regressão utilizando os principais componentes e a restrição Lasso-Lars que reduziu o número de parâmetros significativos do modelo, obtendo assim um AIC menor, ao custo de um RMSE consideravelmente maior.

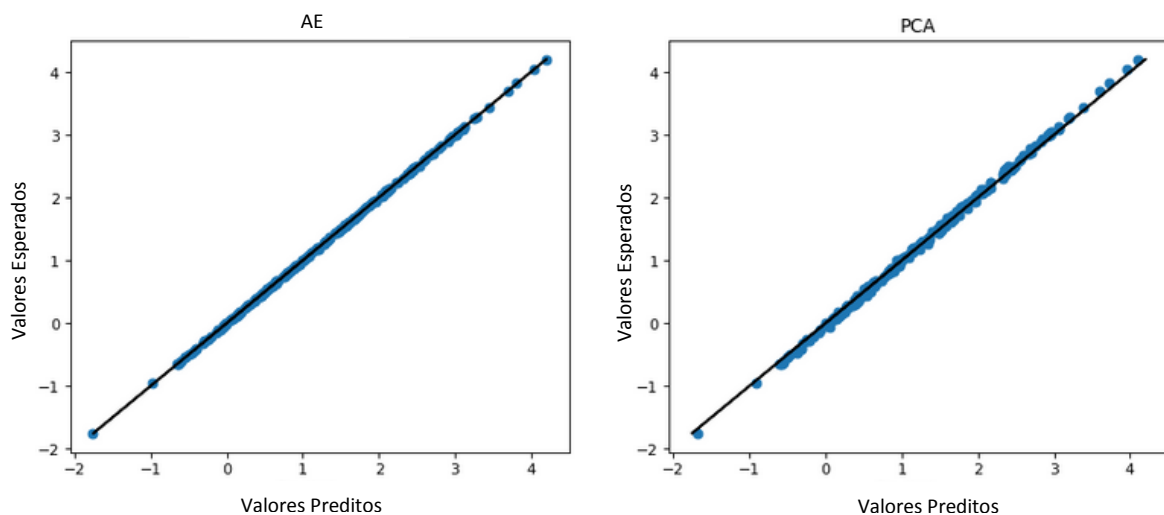


Figura 5.24 Inferências de y a partir dos espaços reduzidos por PCA e pelo autoencoder

Apesar da utilização da função de ativação RELU na codificação dos dados de entrada do autoencoder, os resultados das inferências tanto utilizando a redução do espaço por PCA quanto pelo autoencoder treinado são similares. É interessante apontar que a

restrição Lasso-Lars não zerou coeficientes do modelo gerado a partir do encodado e zerou dois, eliminando o terceiro e quarto principais componentes, do modelo obtido utilizando o PCA. Conforme exposto na seção 5.3, a variável alvo (y) destas bases de dados é função direta de apenas duas, x_3 e x_6 , das seis variáveis que compõem os dados de entrada.

O conjunto de dados de teste foi dopado com 12 pontos pelo algoritmo de adição de ruído e o diagrama conjunto de inferências e erro de reconstrução do autoencoder demonstra através da Figura 5.25, que todos ficaram com um erro acima do valor estabelecido como limite

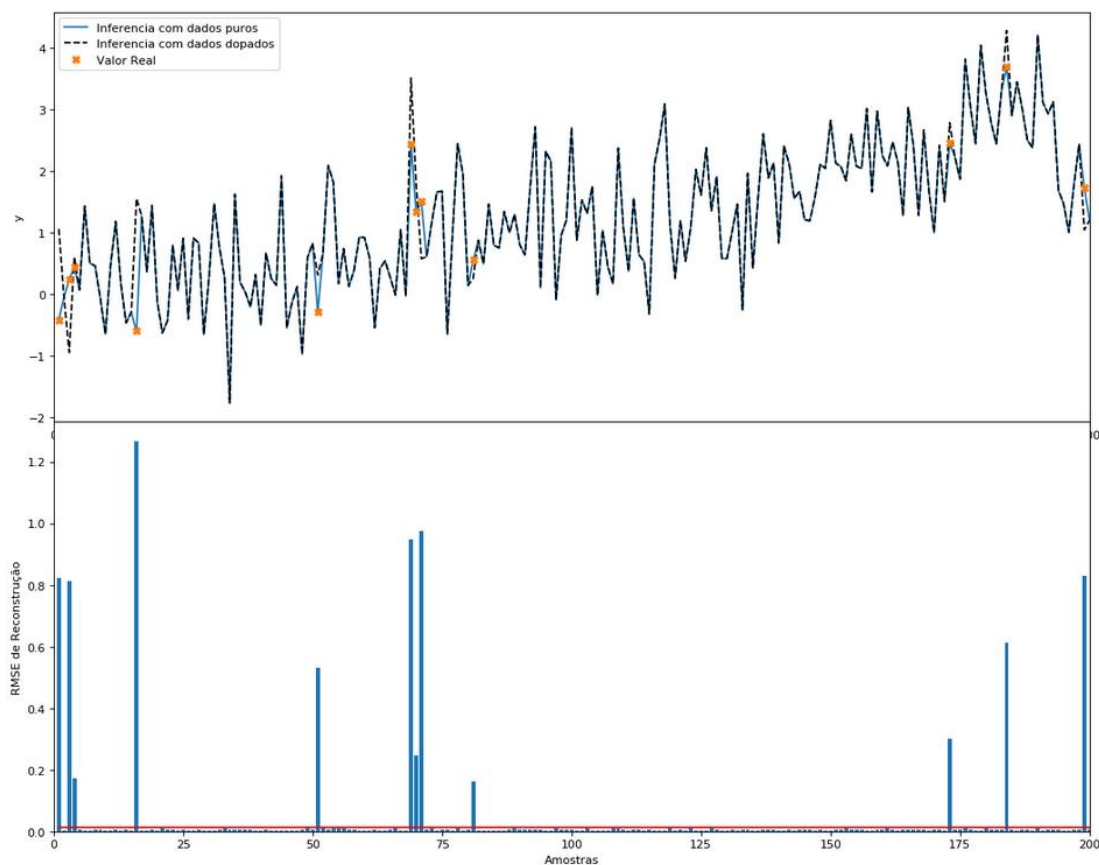


Figura 5.25 Diagrama de inferências e erro de reconstrução dos pontos de operação

5.4.2 Base de dados NL

O conjunto de dados NL, composto por 1001 amostras de seis variáveis (x_1 , x_2 , x_3 , x_4 , x_5 e x_6) em que há uma dependência não-linear entre suas variáveis e a combinação linear entre x_3 e x_6 tem como resultante a variável alvo y . A variável alvo é descrita na Tabela 5.19 e dado que ela apresenta valores menores que zero a métrica de comparação do erro logarítmico médio (MSLE) não será apresentada.

Tabela 5.19: Descrição da variável de saída y

Variável	Nº de Amostras	Média	Desvio Padrão	Mínimo	25%	50%	75%	Máximo
y	1001	2,7353	1,1181	-0,1934	1,9317	2,7342	3,5254	6,0051

O conjunto com relações lineares entre variáveis e variável alvo sendo combinação linear das demais variáveis, na análise de dimensionalidade, apresentou conforme a Figura 5.26 a possibilidade de redução, utilizando PCA, de 6 variáveis para 4.

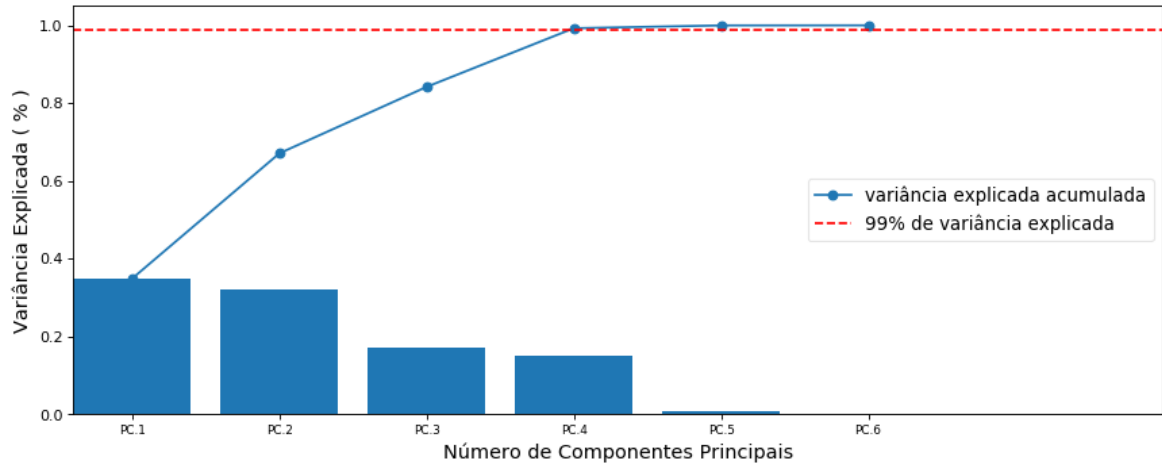


Figura 5.26 Variância Explicadas por cada principal componente do PCA

Por tanto, foram treinadas autoencoders com quatro neurônios no espaço latente. Entre as redes treinadas a de melhor capacidade de reconstrução dos dados de entrada e utilizada para geração de inferências foi a que utilizou a função de ativação RELU na compressão da informação e descomprimiu o espaço latente com uma função linear. Os melhores autoencoder e suas métricas são dispostas na Tabela 5.20.

Tabela 5.20: Melhores Rede Rasa Treinadas

Unidades da Camada Interna	Função de Ativação do Encoder	Função de Ativação do Decoder	RMSE	MSLE
4	linear	linear	0,09262	0,001360
4	RELU	linear	0,09332	0,001197
4	SELU	linear	0,09922	0,001716
4	tanh	linear	0,09945	0,001354
4	SELU	SELU	0,12806	0,001576

O melhor autoencoder capaz de reduzir as seis variáveis de entrada para duas, foi a versão análoga ao PCA, o autoencoder com funções de ativação lineares. Seguindo a metodologia proposta, na Tabela 5.21 temos o comparativo entre as inferências obtidas através do PCA e do espaço reduzido.

Tabela 5.21: Resultados dos critérios de avaliação para os modelos utilizando o conjunto de dados de teste.

	Restrição	Quantidade de Variáveis Seleccionadas	R ²	RMSE	BIC	AIC
Autoencoder	-	4	0,9913	0,1003	-889,09	8,67
	LASSOLARS	3	0,9890	0,1004	-894,12	6,67
	Ridge	4	0,9913	0,1003	-889,19	8,67
PCA	-	4	0,9908	0,1032	-878,00	8,56
	LASSOLARS	2	0,9890	0,1127	-853,76	4,20
	Ridge	4	0,9908	0,1032	-877,86	8,55

A diferença entre as predições feitas pela regressão a partir do espaço latente e a partir dos principais componentes pode ser visualizada na Figura 5.27. A primeira é a regressão sem restrições a partir do espaço latente e a segunda é a regressão utilizando os quatro principais componentes com restrição Lasso-Lars que reduziu o número de parâmetros significativos do modelo.

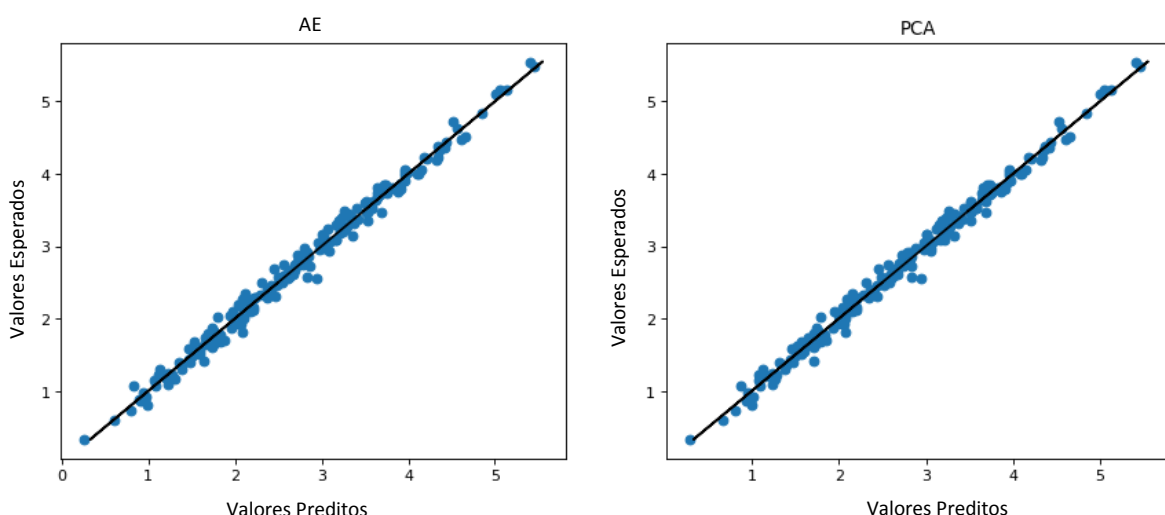


Figura 5.27 Comparação visual entre inferências a partir do espaço latente e dos principais componentes

O resultado segue o mesmo padrão dos anteriores, inferências similares e autoencoder linear com menor erro de reconstrução, e mesmo quando se têm não linearidades na relação entre as variáveis do sistema. Neste caso, a restrição Lasso-Lars removeu apenas uma variável do modelo a partir do espaço latente e duas daquele proveniente dos principais componentes. Com relação conhecida entre as variáveis, é sabido que apenas

duas variáveis (x_3 e x_6) são necessárias para inferência de y . Com isso, foram treinadas outras redes com espaço latente de dois neurônios sendo as melhores apresentadas na Tabela 5.22.

Tabela 5.22: Melhores redes com 2 neurônios no espaço latente

Unidades da Camada Interna	Função de Ativação do Encoder	Função de Ativação do Decoder	RMSE	MSLE
2	linear	linear	0,6000	0,0624
2	SELU	linear	0,6005	0,06135
2	tanh	linear	0,6012	0,06237
2	RELU	linear	0,6021	0,06291
2	RELU	SELU	0,6044	0,06194

Construindo as inferências, utilizando-se os dois principais componentes, que agregam 67% da variância dos dados, e além do espaço latente do melhor autoencoder da Tabela 5.22 utilizando o único autoencoder que não possui funções de ativação lineares obtêm-se os resultados apresentados na Tabela 5.23.

Tabela 5.23: Resultados dos critérios de avaliação para os modelos utilizando o conjunto de dados de teste.

Redução do Espaço	Quantidade de Variáveis Seleccionadas	R ²	RMSE	BIC	AIC
PCA	2	0,9900	0,1072	-871,41	4,38
AE linear-linear	2	0,9904	0,1053	-878,67	4,46
AE RELU-SELU	2	0,9933	0,0873	-952,68	5,21

Apesar da melhora no AIC, não há uma melhora expressiva nos outros parâmetros e mesmo utilizando o autoencoder não linear, que possui o menor RMSE. Tem-se, entretanto, um ponto negativo que pode ser percebido comparando as Figura 5.28 e Figura 5.29. A primeira apresenta o diagrama, do espaço latente com 4 unidades, com inferência e erro de reconstrução dos dados do conjunto de teste que foi dopado e teve 12 das 180 amostras corrompidas. A segunda é referente ao autoencoder com 2 neurônios no espaço latente. Há uma clara vantagem no aumento do tamanho do espaço latente. Com mais neurônios é possível fazer uma reprodução mais fiel das entradas e desta forma filtrar melhor os pontos que estão ou não dentro do espaço do modelo de inferência.

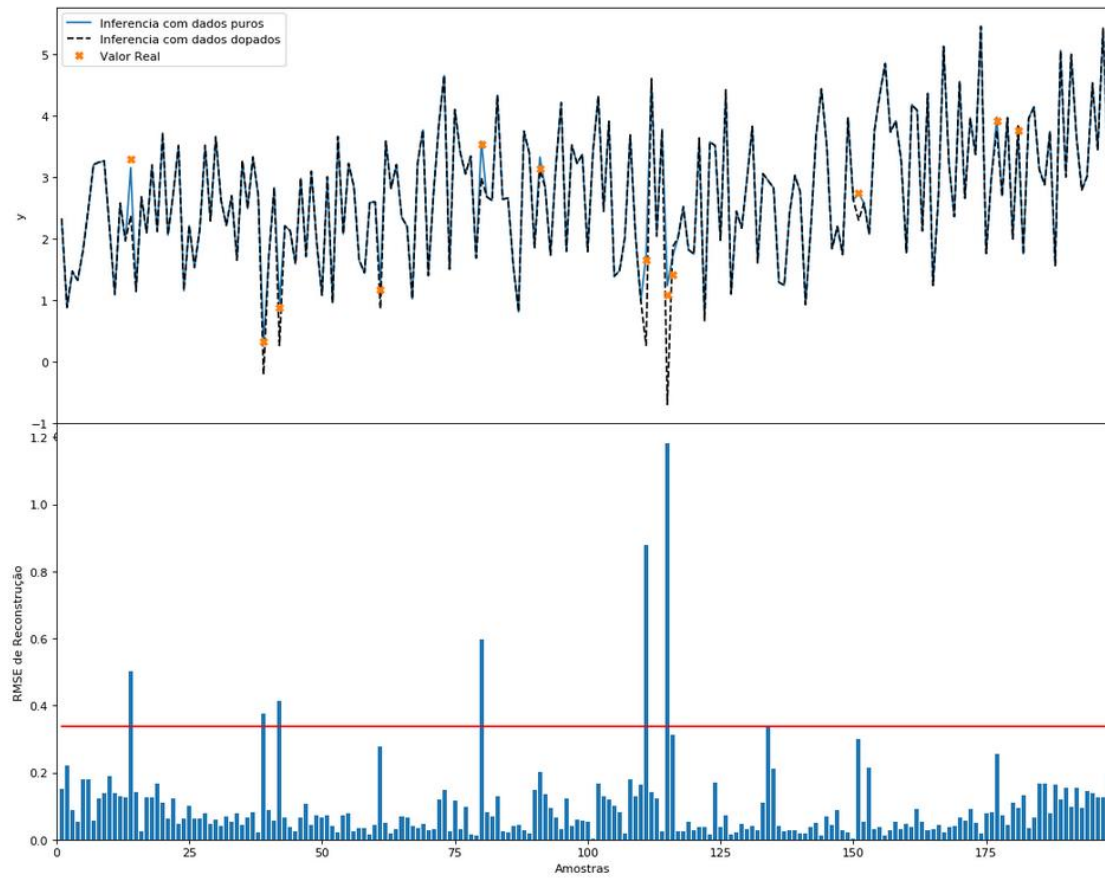


Figura 5.28 Diagrama de inferências e erro de reconstrução do autoencoder com 4 neurônios no espaço latente.

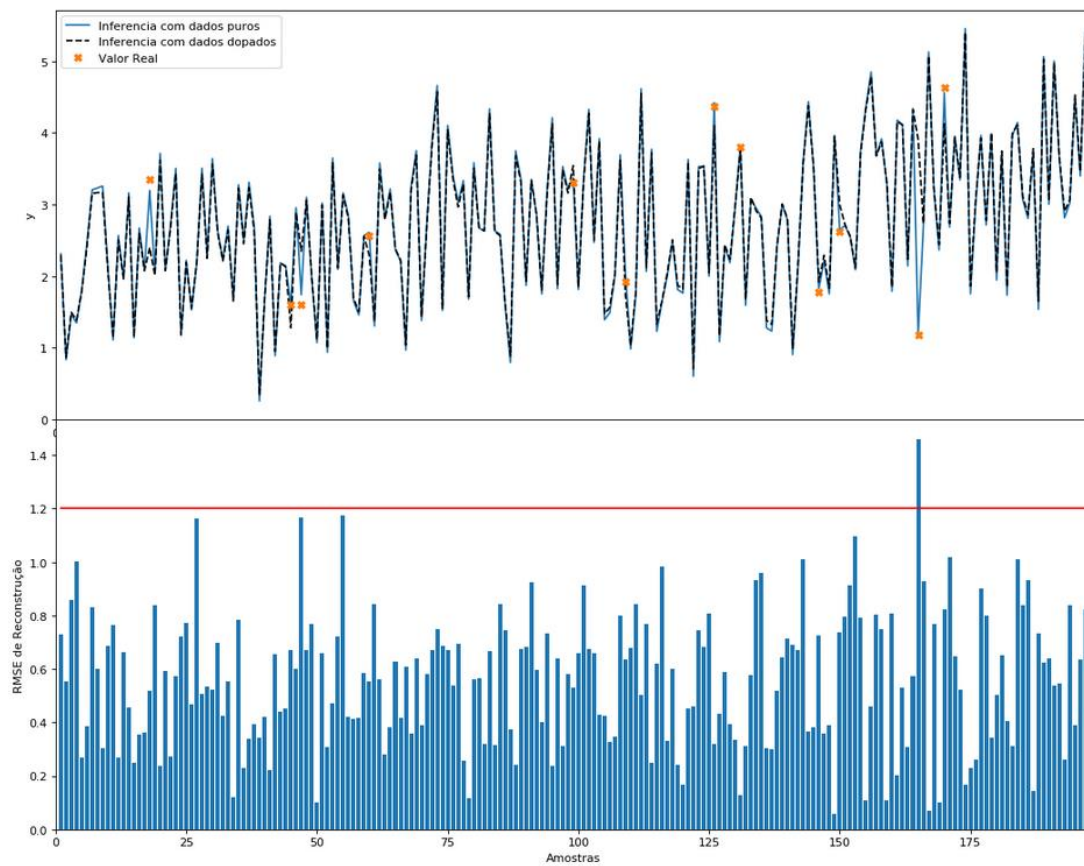


Figura 5.29 Diagrama de inferências e erro de reconstrução com 2 neurônios no espaço latente.

5.4.3 Base de dados LN

O conjunto de dados LN, composto por 1001 amostras de seis variáveis (x_1 , x_2 , x_3 , x_4 , x_5 e x_6) em que há uma dependência linear entre suas variáveis e a relação não-linear entre x_3 e x_6 gera a variável alvo y . A variável alvo é descrita na TABELA e dado que ela apresenta valores menores que zero a métrica de comparação do erro logarítmico médio (MSLE).

Tabela 5.24: Descrição da variável de saída y

Variável	Nº de Amostras	Média	Desvio Padrão	Mínimo	25%	50%	75%	Máximo
y	1001	2,7353	1,1181	-0,1934	1,9317	2,7342	3,5254	6,0051

O conjunto apresentou conforme a Figura 5.30 a possibilidade de redução, utilizando PCA, de 6 variáveis para 4, optando pela retenção de 99% da variância dos dados.

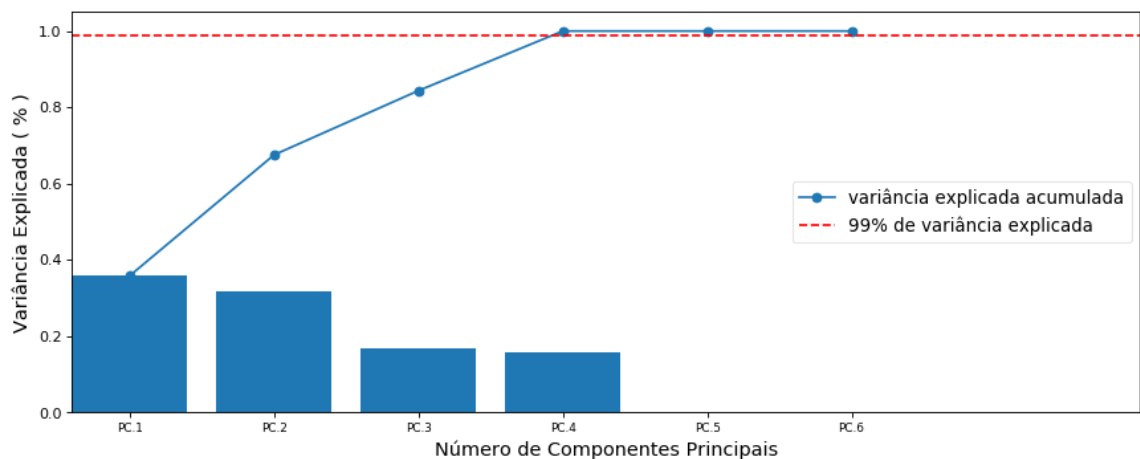


Figura 5.30 Variância explicada por cada principal componente do PCA

Foram treinados autoencoders com quatro neurônios na camada interna. Entre as redes treinadas a de melhor capacidade de reconstrução dos dados de entrada e utilizada para geração de inferências foi o autoencoder com função de ativação linear para codificação e decodificação. Os melhores autoencoder e suas métricas são dispostos na Tabela 5.25 e em seguida, na Tabela 5.26, são demonstrados os resultados das regressões lineares.

Tabela 5.25 Melhores Rede Rasas Treinadas.

Unidades da Camada Interna	Função de Ativação do Encoder	Função de Ativação do Decoder	RMSE	MSLE
4	linear	linear	0,00621	0,7E-5
4	RELU	linear	0,00765	0,9E-5

4	tanh	linear	0,02018	5,4E-5
4	SELU	linear	0,02562	9,4E-5
4	SELU	SELU	0,09303	31,1E-5

Tal resultado é esperado, principalmente por esta base de dados ser similar a base de dados LL, uma vez que até aqui não foram utilizados os dados de saída, a variável y . Os espaços reduzidos são ajustados aos dados do conjunto de treino e os resultados destes modelos com os dados de teste são apresentados na Tabela 5.26.

Tabela 5.26 Resultados dos critérios de avaliação para os modelos utilizando o conjunto de dados de teste.

	Restrição	Quantidade de Variáveis Seleccionadas	R ²	RMSE	BIC	AIC
Autoencoder	-	4	0,9385	1,3125	130,22	-1,63
	LASSOLARS	4	0,9385	1,3125	130,22	-1,63
	Ridge	4	0,9385	1,3136	130,55	-1,64
PCA	-	4	0,9385	1,3125	130,20	-1,63
	LASSOLARS	3	0,9380	1,3180	126,57	-3,65
	Ridge	3	0,9384	1,3133	130,45	-1,64

A grande similaridade entre as predições feitas pela regressão a partir do espaço latente e a partir dos principais componentes pode ser visualizada na Figura 5.31. A primeira é a regressão sem restrições a partir do espaço espaço latente e a segunda é a regressão utilizando os quatro principais componentes com restrição Lasso-Lars que reduziu o número de parâmetros significativos do modelo.

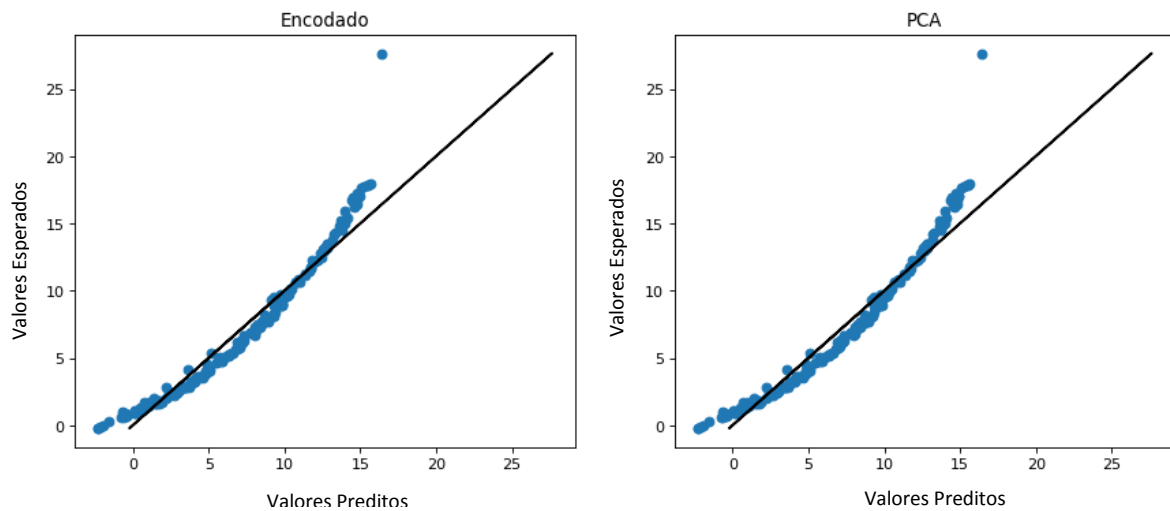


Figura 5.31 Comparação visual entre inferências a partir do espaço latente e dos componentes principais.

Os resultados são bastante similares e devido a relação não-linear entre as variáveis x e a variável alvo y era esperado que regressões lineares não fossem suficientes para ajustar tal comportamento. Neste caso, a restrição Lasso-Lars removeu apenas uma variável do modelo a partir do PCA e nenhuma do baseado no espaço latente.

A Figura 5.32 mostra novamente a capacidade do autoencoder de identificar todos os 12 pontos em que foi adicionado o ruído. Sendo que o erro, RMSE, máximo de reconstrução dos dados de treino era aproximadamente 0,01 enquanto alguns dos erros das amostras com falha chegam a valores cem vezes maior.

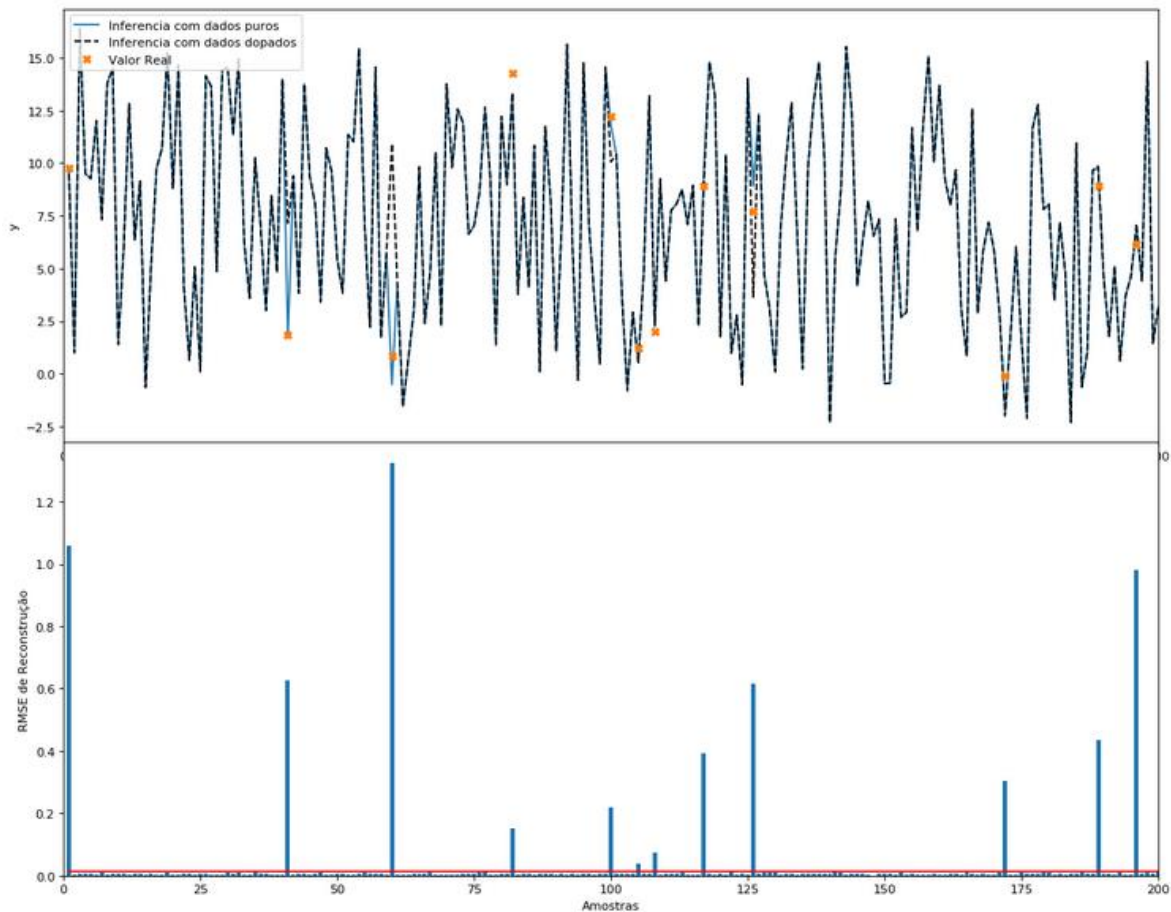


Figura 5.32: Diagrama de inferências e erro de reconstrução do autoencoder – dados LN

5.4.1 Base de dados NN

O conjunto de dados NN, é composto por 1001 amostras de seis variáveis (x_1 , x_2 , x_3 , x_4 , x_5 e x_6) em que há uma relação não-linear entre suas variáveis e a relação não-linear entre x_3 e x_6 gerando a variável alvo y . A variável é descrita na Tabela 5.27.

Tabela 5.27: Descrição da variável de saída y

Variável	Nº de Amostras	Média	Desvio Padrão	Mínimo	25%	50%	75%	Máximo
y	1001	53,53	943,55	0,7429	4,6779	7,1673	11,38	29658,6

Optando pela retenção de 99% da variância dos dados, como tem sido feito, o método PCA precisa das seis variáveis, como mostrado na Figura 5.33. Com 4 componentes tem-se 92% da variância do sistema e com 5 já se tem pouco mais de 98%. Neste caso serão usados apenas os cinco principais componentes para haver uma redução do espaço.

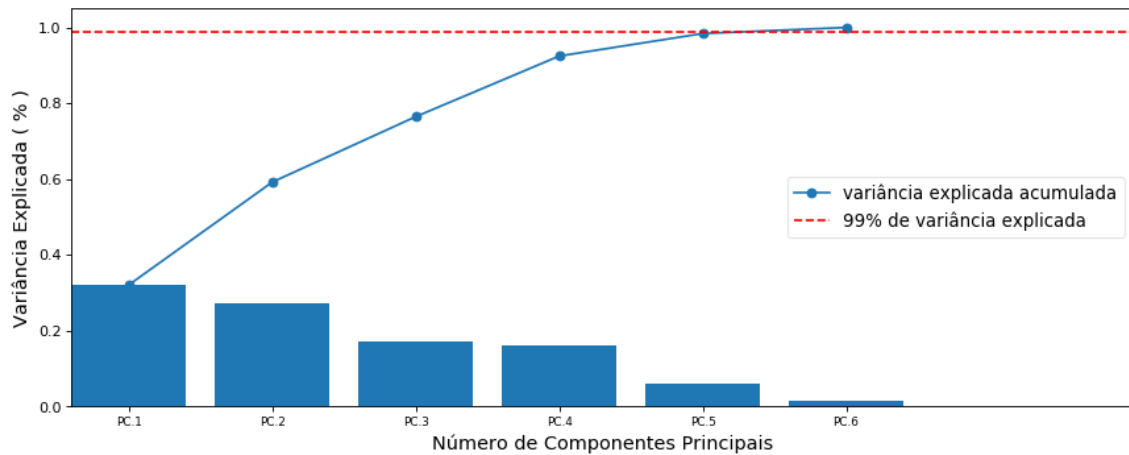


Figura 5.33: Variância explicada por cada principal componente do PCA.

Foram treinados autoencoders com cinco neurônios na camada interna. Entre as redes treinadas a de melhor capacidade de reconstrução dos dados de entrada e utilizada para geração de inferências foi o autoencoder com funções de ativação SELU para compressão e descompressão. Os melhores autoencoder e suas métricas são dispostos na Tabela 5.28.

Tabela 5.28: Melhores Redes Rasas Treinadas com 5 neurônios na camada interna

Unidades da Camada Interna	Função de Ativação do Encoder	Função de Ativação do Decoder	RMSE	MSLE
5	SELU	SELU	0,1199	0,002286
5	RELU	SELU	0,1201	0,001733
5	RELU	linear	0,1208	0,002063
5	linear	SELU	0,1213	0,001712
5	linear	linear	0,1245	0,002610

O resultado do comparativo das redes era esperado devido à grande não-linearidade entre as variáveis de entrada. Entretanto, se comparado ao erro de reconstrução dos outros conjuntos de dados, principalmente o LL e o LN, verifica-se que a capacidade de generalização dos dados para este autoencoder é inferior. Os espaços reduzidos são ajustados aos dados do conjunto de treino e os resultados destes modelos com os dados de teste são apresentados na Tabela 5.29, mostram que mesmo com funções de ativação potencialmente não lineares, não há uma melhor inferência para os dados utilizados.

Tabela 5.29: Resultados dos critérios de avaliação para os modelos utilizando o conjunto de dados de teste.

	Restrição	Quantidade de Variáveis Seleccionadas	R ²	RMSE	BIC	AIC
Aut oen	-	5	0,8885	22,02	1237,67	-10,86

	LASSOLARS	3	0,8726	23,55	1253,34	-15,13
	Ridge	5	0,8879	22,08	1238,70	-10,87
PCA	-	5	0,8867	22,21	1240,94	-10,90
	LASSOLARS	5	0,8867	22,21	1240,94	-10,90
	Ridge	5	0,8857	22,30	1242,58	-10,91

A grande similaridade entre as predições feitas pela regressão a partir do espaço latente e a partir dos principais componentes pode ser visualizada na Figura 5.34. A primeira é a regressão a partir do espaço espaço latente e a segunda é a regressão utilizando os cinco principais componentes, ambas utilizando a restrição Lasso-Lars.

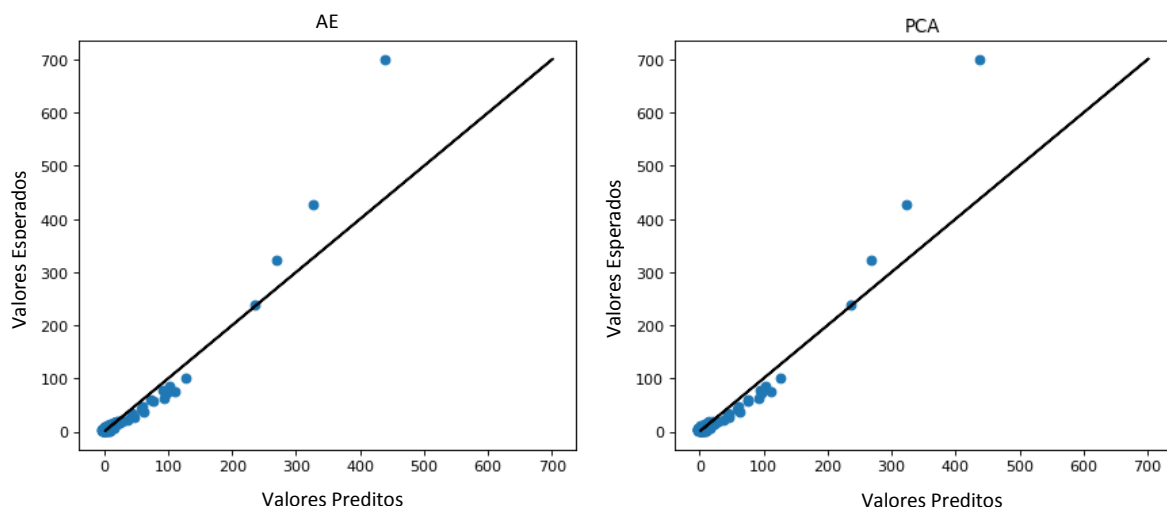


Figura 5.34: Comparação visual entre inferências a partir do espaço latente e dos principais componentes

Os resultados são bastante similares e devido a relação não-linear entre as variáveis x e a variável alvo y era esperado que regressões lineares não fossem suficientes para ajustar tal comportamento. Pode se fazer um comparativo entre os dois modelos e seus parâmetros através da Tabela 5.30, onde são mostrados os coeficientes dos parâmetros, seus erros e a probabilidade de rejeição daquele parâmetro.

Tabela 5.30 Coeficientes das regressões lineares utilizando Lasso-Lars

PCA				Autoencoder			
	Coeficientes	Erros	P(h_0)		Coeficientes	Erros	P(h_0)
Cte	17,9677	1,590	<0,001	Cte	-6,5002	3,435	0,06
PC1	-1,6358	1,253	0,193	N1	0	2,191	1
PC2	26,4397	1,208	<0,001	N2	27,6516	1,295	<0,001
PC3	1,5374	1,485	0,302	N3	0	3,008	1

PC4	-4,1085	1,557	0,009	N4	16,9316	3,008	<0,001
PC5	43,23307	2,362	<0,001	N5	20,7849	2,003	<0,001

Como já indicado pelos conjuntos de dados anteriores, e a Figura 5.35 demonstra, o autoencoder não foi capaz de identificar todos os 12 pontos em que foi adicionado dados espúrios. Na figura, as barras em laranja demonstram os valores do erro de reconstrução sem os dados espúrios adicionados.

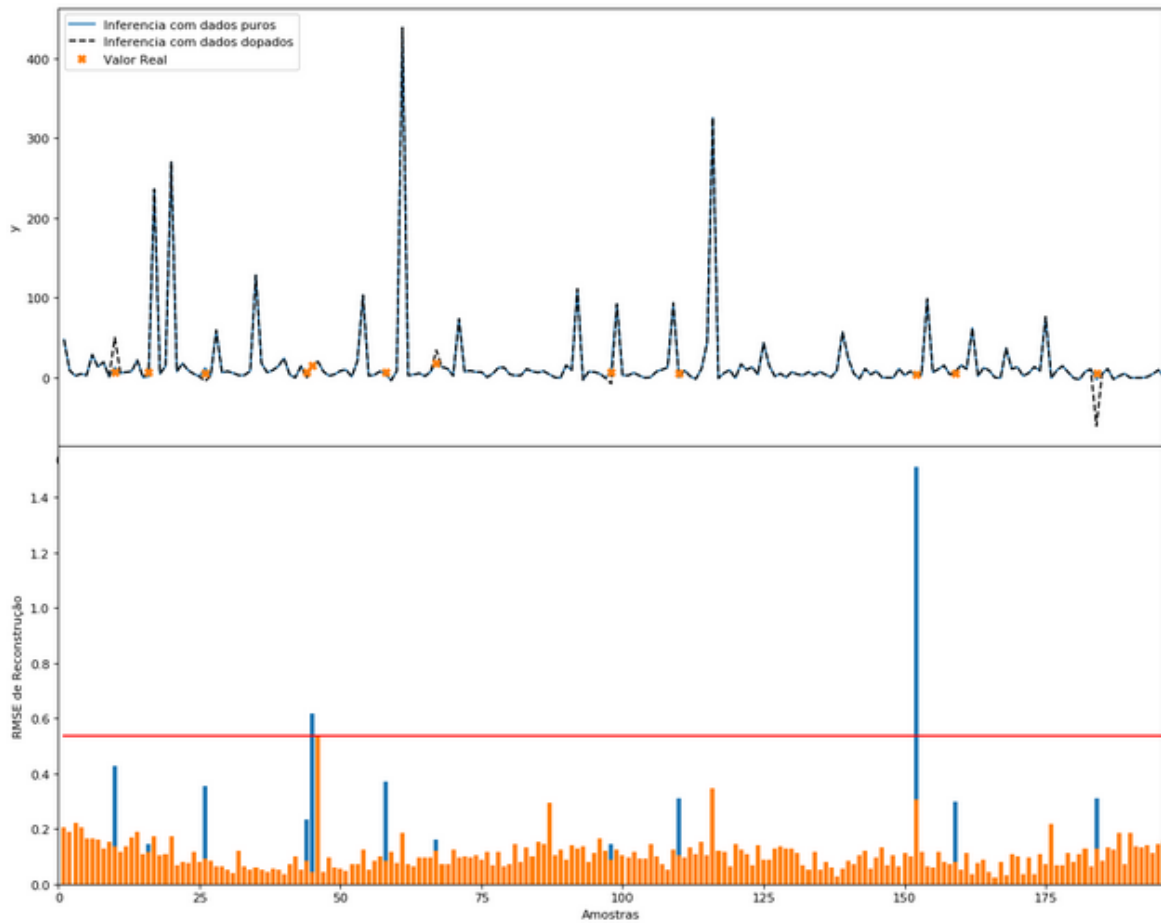


Figura 5.35: Diagrama de inferências e erro de reconstrução do autoencoder – dados NN

5.5 Conclusões

Baseado na metodologia proposta, a utilização do autoencoder, principalmente o constituído de funções de ativação lineares tem no mínimo capacidade similar de generalização dos sistemas aplicados. Sua característica de detecção de falhas é uma ferramenta de alto interesse para monitoramento de processos na indústria química, mas precisa ser melhor estudada.

Dentre os métodos de treinamento das redes neurais, foram testados uma ampla gama de hiperparâmetros associados ao ajuste dos autoencoders, mas via de regra, para os casos em que foi aplicada metodologia, não houve generalização não-linear satisfatória, mesmo em casos com relação conhecida não-linear entre variáveis.

Quanto aos métodos utilizados para se construir os modelos, não houve um método que se saiu melhor em todas as situações. Portanto, não se pode defender a utilização de apenas um método, mas sim a comparação deles e a validação do melhor método. Há, porém, uma clara necessidade em se aumentar a complexidade dos modelos de inferência uma vez que não se obteve, nem era o objetivo, resultados expressivos e que teriam utilidade real, em se tratando dos casos das colunas simuladas. A expansão de bases destas regressões lineares feitas poderia obter melhores resultados, por exemplo.

Durante o desenvolvimento do trabalho, os resultados obtidos com a base de dados oriunda da simulação das colunas não estavam demonstrando uma superioridade da metodologia utilizando autoencoder, principalmente com relação à captura de não linearidades. Com isso, foram geradas as bases de dados artificiais que teriam não linearidades conhecidas a serem generalizadas durante o treinamento da rede neural. Entretanto, mesmo com esses dados, os resultados que foram obtidos não se demonstraram melhores que aqueles obtidos através de uma metodologia muito menos complexa como esperado.

Outro problema encontrado, que deve ser melhor explorado, é a automatização e otimização dos hiperparâmetros, que têm elevado custo computacional e neste trabalho foi feito de forma manual e sem uma busca realmente exaustiva. O aumento da profundidade da rede do autoencoder, adicionando mais camadas internas, não influenciou na qualidade das inferências geradas a partir do seu espaço latente.

Há uma frustração em relação ao potencial que era esperado do autoencoder e o resultado apresentado neste trabalho. Porém este trabalho tem valor por iniciar e propor uma aplicação diferente para o autencoder.

Capítulo 6 – Considerações Finais

A indústria hoje compreende a necessidade por uma produção mais segura, mais limpa e mais eficiente. Consequentemente os sistemas avançados de monitoramento e controle vêm ganhando destaque nos processos industriais. No entanto, algumas variáveis ainda enfrentam problemas quando se falta em medições confiáveis e rápidas. Os analisadores em linha se apresentam como uma alternativa, mas ainda os tempos de análise e amostragem não são apropriados para o controle direto, além disso são equipamentos caros, que exigem manutenção especializada e suas informações não pode são confiáveis. As medições laboratoriais também passam pelo problema do tempo de amostragem. Com elas, não se consegue controlar automaticamente os processos.

Diante das dificuldades em se monitorar e mensurar variáveis relacionadas com a qualidade dos produtos, o presente trabalho propôs uma sistemática para o desenvolvimento de inferências, com a finalidade de estimar variáveis de difícil medição, seja para monitoramento e controle do processo.

Compreende-se que, ao se trabalhar com dados e com métodos de regressão, não há um método suficientemente bom para todos os possíveis casos, especialmente quando se trata de situações não lineares. Para isso, a metodologia foi construída de modo que é feita uma comparação entre os modelos propostos mesmo que não seja o principal objetivo do trabalho.

Tal como os métodos de regressão, é válida a utilização de várias métricas de avaliação para que se possa validar com mais clareza a qualidade dos modelos. Essa necessidade pode ser vista, por exemplo, na análise do coeficiente de determinação R^2 dos modelos construídos nos estudos de caso. Esse critério não é suficiente para determinar qual dos modelos generaliza melhor os dados. Sendo, por tanto, interessante se utilizar de várias métricas em conjunto.

Por fim, a principal conclusão é que apesar do enorme potencial que o *autoencoder* assume no campo de analisadores virtuais. Este trabalho ainda não conseguiu atestar e justificar sua ampla utilização. Os resultados apresentados foram equivalentes aos obtidos através da metodologia PCA, só que a um custo computacional significativamente superior. Há uma necessidade de melhorar a capacidade de generalização dos dados. Tal qual a área de tratamento de imagens teve com a implementação das camadas convolucionais, as redes neurais precisam de uma especialização para obter melhores resultados e saírem da categoria caixa-preta para o que chamamos de *feature engineering*.

6.1 Sugestões para trabalhos futuros

A seguir são listadas algumas sugestões para trabalhos futuros:

- Buscar alternativas de aprendizado não supervisionado com autoencoder de arquiteturas diferentes – GANs e Autoencoders do tipo *Overcomplete*;
- Especializar o autoencoder com relação à sua função objetivo – o erro entre entradas e saídas pode ser capturado de maneira mais representativa e conclusiva;
- Aplicar a metodologia a dados online de processo – utilização de redes que se atualizam em janelas de tempo operacional – Autoencoder Bayesiano;
- Tornar o autoencoder uma rede escassa, identificando a relação entre as variáveis observadas e a inferência – técnicas novas de *dropout* buscam otimizar a busca dos parâmetros das redes sem perder tanto a capacidade de generalização dos dados.

Referências

ABADI, M. et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. [s. l.], 2016. Disponível em: <<http://arxiv.org/abs/1603.04467>>

AGARWAL, P.; BUDMAN, H. Classification of Profit-Based Operating Regions for the Tennessee Eastman Process using Deep Learning Methods. **IFAC-PapersOnLine**, [s. l.], v. 52, n. 1, p. 556–561, 2019. Disponível em: <<https://linkinghub.elsevier.com/retrieve/pii/S2405896319302071>>

ALMOTIRI, J.; ELLEITHY, K.; ELLEITHY, A. Comparison of Autoencoder and Principal Component Analysis Followed by Neural Network for E-Learning Using Handwritten Recognition. **2017 IEEE Long Island Systems, Applications and Technology Conference, LISAT 2017**, [s. l.], n. May, 2017.

BAKIROV, R.; GABRYS, B.; FAY, D. Multiple adaptive mechanisms for data-driven soft sensors. **Computers and Chemical Engineering**, [s. l.], v. 96, p. 42–54, 2017. Disponível em: <<http://dx.doi.org/10.1016/j.compchemeng.2016.08.017>>

BALDI, P. Autoencoders, Unsupervised Learning, and Deep Architectures. **JMLR: Workshop and Conference Proceedings**, [s. l.], n. 27, p. 37–50, 2012. Disponível em: <<http://proceedings.mlr.press/v27/baldi12a/baldi12a.pdf>>

BALDI, P.; HORNIK, K. Neural networks and principal component analysis: Learning from examples without local minima. **Neural Networks**, [s. l.], v. 2, n. 1, p. 53–58, 1989.

BASTIDAS, M. E. H. Detecção E Diagnóstico De Falhas Baseado Em Modelos Empíricos No Subespaço Das Variáveis De Processo (Empvsub). [s. l.], 2018.

BENGIO, Y. Practical recommendations for gradient-based training of deep architectures. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, [s. l.], v. 7700 LECTU, p. 437–478, 2012.

BERGSTRA, J. et al. Hyperopt: A Python library for model selection and hyperparameter optimization. **Computational Science and Discovery**, [s. l.], v. 8, n. 1, 2015.

- BERGSTRA, J.; YAMINS, D.; COX, D. D. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. **12th PYTHON IN SCIENCE CONF. (SCIPY 2013)**, [s. l.], n. Scipy, p. 13–20, 2013. Disponível em: <<http://hyperopt.github.io/hyperopt/%5Cnhttps://github.com/jaberg/hyperopt%5Cnhttp://www.youtube.com/watch?v=Mp1xnPfe4PY>>
- BOLLAPRAGADA, R. et al. A Progressive Batching L-BFGS Method for Machine Learning. [s. l.], 2018. Disponível em: <<http://arxiv.org/abs/1802.05374>>
- BOULLOSA, D. et al. Monitoring through T2 Hotelling of cylinder lubrication process of marine diesel engine. **Applied Thermal Engineering**, [s. l.], v. 110, p. 32–38, 2017. Disponível em: <<http://dx.doi.org/10.1016/j.applthermaleng.2016.08.062>>
- CHARTE, D. et al. A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines. **Information Fusion**, [s. l.], v. 44, n. December 2017, p. 78–96, 2018. Disponível em: <<https://doi.org/10.1016/j.inffus.2017.12.007>>
- CHOLLET, F. **Deep Learning With Python**. [s.l: s.n.]. v. 1
- CHOLLET, F. **Deep Learning with Python**. 1st. ed. Greenwich, CT, USA: Manning Publications Co., 2017.
- CIREŞAN, D. C. et al. Deep, big, simple neural nets for handwritten digit recognition. **Neural Computation**, [s. l.], v. 22, n. 12, p. 3207–3220, 2010.
- CIVILEKOGU, G. et al. Modeling carbon and nitrogen removal in an industrial wastewater treatment plant using an adaptive network-based fuzzy inference system. **Clean - Soil, Air, Water**, [s. l.], v. 35, n. 6, p. 617–625, 2007.
- COLLOBERT, R.; KAVUKCUOGLU, K.; FARABET, C. Torch7: A matlab-like environment for machine learning. **BigLearn, NIPS Workshop**, [s. l.], p. 1–6, 2011. Disponível em: <http://infoscience.epfl.ch/record/192376/files/Collobert_NIPSWORKSHOP_2011.pdf>
- CYBENKO, G. Approximation by Superpositions of a Sigmoidal Function. **Math. Control Signals Systems**, [s. l.], v. 2, p. 303–314, 1989.
- DAUPHIN, Y. et al. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. [s. l.], 2014. Disponível em: <<https://arxiv.org/abs/1406.2572>>. Acesso em: 26 jul. 2019.
- DEAN, J. et al. Large Scale Distributed Deep Networks. In: PEREIRA, F. et al. (Eds.). **Advances in Neural Information Processing Systems 25**. [s.l.] : Curran Associates, Inc., 2012. p. 1223–1231.
- DENG, J. et al. Autoencoder-based unsupervised domain adaptation for speech emotion recognition. **IEEE Signal Processing Letters**, [s. l.], v. 21, n. 9, p. 1068–1072, 2014.
- DENG, L. et al. Binary Coding of Speech Spectrograms Using a Deep Auto-encoder.

Interspeech, [s. l.], n. September, p. 1692–1695, 2010.

DING, F.; CHEN, T. Modeling and identification of multirate systems. **Zidonghua Xuebao/Acta Automatica Sinica**, [s. l.], v. 31, n. 1, p. 105–122, 2005.

DONG, D.; MCAVOY, T. J. Nonlinear principal component analysis - Based on principal curves and neural networks. **Computers and Chemical Engineering**, [s. l.], v. 20, n. 1, p. 65–78, 1996.

DONG, X.; CHEN, S.; PAN, S. J. Learning to Prune Deep Neural Networks via Layer-wise Optimal Brain Surgeon. [s. l.], n. Nips, 2017. Disponível em: <<http://arxiv.org/abs/1705.07565>>

DOZAT, T. Incorporating Nesterov Momentum into Adam. **ICLR Workshop**, [s. l.], n. 1, p. 2013–2016, 2016.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. **Journal of Machine Learning Research**, [s. l.], v. 12, p. 2121–2159, 2011.

FACCHIN, S. **Técnicas de Análise Multivariável aplicadas ao Desenvolvimento de Analisadores Virtuais**. 2005. UFRGS, [s. l.], 2005.

GAMA, J. et al. Learning with Drift Detection. **Advances in Artificial Intelligence**, [s. l.], p. 286–295, 2004. Disponível em: <http://link.springer.com/10.1007/978-3-540-28645-5_29>

GODFREY, L. B.; GASHLER, M. S. A Continuum among Logarithmic, Linear, and Exponential Functions, and Its Potential to Improve Generalization in Neural Networks. [s. l.], p. 481–486, 2015.

GOLDTHORPE, P.; DESMET, A. Denoising autoencoder anomaly detection for correlated data. **European Conference of the Prognostics and Health Management Society**, [s. l.], p. 1–6, 2018.

GONZAGA, J. C. B. et al. ANN-based soft-sensor for real-time process monitoring and control of an industrial polymerization process. **Computers and Chemical Engineering**, [s. l.], v. 33, n. 1, p. 43–49, 2009.

GRAVES, A. Generating Sequences With Recurrent Neural Networks. [s. l.], 2013. Disponível em: <<https://arxiv.org/abs/1308.0850>>. Acesso em: 26 jul. 2019.

GREENE, W. H. **Econometric Analysis**. Seventh ed. [s.l: s.n.].

HE, K. et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. **Proceedings of the IEEE International Conference on Computer Vision**, [s. l.], v. 2015 Inter, p. 1026–1034, 2015.

HINTON, G. E. et al. Improving neural networks by preventing co-adaptation of feature detectors. [s. l.], p. 1–18, 2012. Disponível em: <<http://arxiv.org/abs/1207.0580>>

HINTON, G. E.; SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural

networks. **Science**, [s. l.], v. 313, n. 5786, p. 504–507, 2006. Disponível em: <<http://www.ncbi.nlm.nih.gov/sites/entrez?db=pubmed&uid=16873662&cmd=showdetailview&indexed=google>>

HODGE, V. J.; AUSTIN, J. A Survey of Outlier Detection Methodologies. **Artificial Intelligence Review**, [s. l.], v. 22, n. 2, p. 85–126, 2004. Disponível em: <<https://doi.org/10.1007/s10462-004-4304-y>>

HOLDEN, A. J. et al. Data with Neural Networks - hinton. [s. l.], v. 313, n. July, p. 504–507, 2006.

HONG, C. et al. Multimodal Deep Autoencoder for Human Pose Recovery. **IEEE Transactions on Image Processing**, [s. l.], v. 24, n. 12, p. 5659–5670, 2015.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer Feedforward Networks are Universal Approximators. **Neural Networks**, [s. l.], v. 2, p. 359–366, 1989.

HOSKINS, J. C.; HIMMELBLAU, D. M. Artificial neural network models of knowledge representation in chemical engineering. **Computers and Chemical Engineering**, [s. l.], v. 12, n. 9–10, p. 881–890, 1988.

IOFFE, S.; SZEGEDY, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. [s. l.], 2015. Disponível em: <<https://arxiv.org/pdf/1502.03167.pdf>>

JAMES, B.; YOSHUA, B. Random Search for Hyper-Parameter Optimization. **Journal of Machine Learning Research**, [s. l.], v. 13, n. 1, p. 281–305, 2012. Disponível em: <<https://dl.acm.org/citation.cfm?id=2188395>>

JARRETT, K. et al. What is the best multi-stage architecture for object recognition? In: 2009 IEEE 12TH INTERNATIONAL CONFERENCE ON COMPUTER VISION (ICCV) 2009, Los Alamitos, CA, USA. **Anais...** Los Alamitos, CA, USA: IEEE Computer Society, 2009. Disponível em: <<https://doi.ieeecomputersociety.org/10.1109/ICCV.2009.5459469>>

JES, F. et al. Deep Convolutional Autoencoders vs PCA in a Highly-Unbalanced Parkinson's Disease Dataset : A DaTSCAN Study. **Springer Nature**, [s. l.], p. 47–56, 2019.

JIA, Y. et al. Caffe: Convolutional Architecture for Fast Feature Embedding. [s. l.], 2014. Disponível em: <<http://arxiv.org/abs/1408.5093>>

KADLEC, P.; GABRYS, B. **Soft sensors: Where are we and what are the current and future challenges?** [s.l.] : IFAC, 2009. v. 2 Disponível em: <<http://dx.doi.org/10.3182/20090921-3-TR-3005.00098>>

KADLEC, P.; GABRYS, B.; STRANDT, S. Data-driven Soft Sensors in the process industry. **Computers and Chemical Engineering**, [s. l.], v. 33, n. 4, p. 795–814, 2009.

KADLEC, P.; GRBIĆ, R.; GABRYS, B. Review of adaptation mechanisms for data-driven soft sensors. **Computers and Chemical Engineering**, [s. l.], v. 35, n. 1, p. 1–24, 2011.

KETKAR, N. **Introduction to Keras**. [s.l.: s.n.].

KINGMA, D.; BA, J. ADAM: A Method for Stochastic Optimization. **Proc. of the 3rd International Conference for Learning Representations**, [s. l.], p. 1–15, 2015. Disponível em: <<http://arxiv.org/abs/1412.6980>>

KLAMBAUER, G. et al. Self-Normalizing Neural Networks. **Advances in Neural Information Processing Systems**, [s. l.], p. 971–980, 2017. Disponível em: <<http://arxiv.org/abs/1706.02515>>

KOLOURI, S. et al. Sliced-Wasserstein Autoencoder: An Embarrassingly Simple Generative Model. [s. l.], n. Algorithm 1, p. 1–25, 2018. Disponível em: <<http://arxiv.org/abs/1804.01947>>

KOURTI, T.; MACGREGOR, J. F. J. F. Process analysis, monitoring and diagnosis, using multivariate projection methods. **Chemometrics and intelligent laboratory systems**, [s. l.], v. 28, p. 3–21, 1995. Disponível em: <<http://www.sciencedirect.com/science/article/pii/0169743995800369%5Cnhttp://www.sciencedirect.com/science/article/B6TFP-46X12TV-18/2/fdb028b26900bd0887453a76d8c4d4b8>>

L. RUSSELL, E.; CHIANG, L.; BRAATZ, R. **Data-Driven Methods for Fault Detection and Diagnosis in Chemical Processes**. [s.l.: s.n.].

LE, Q. V. et al. Building high-level features using large scale unsupervised learning. In: PROCEEDINGS OF THE 29TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING 2012, **Anais...** [s.l.: s.n.] Disponível em: <<http://arxiv.org/abs/1112.6209>>. Acesso em: 26 jul. 2019.

LECUN, Y. et al. Gradient-Based Learning Applied to Document Recognition. In: PROCEEDINGS OF THE IEEE (86) 11 1998a, **Anais...** : IEEE, 1998. Disponível em: <<http://ieeexplore.ieee.org/document/726791/#full-text-section>>

LECUN, Y. et al. Efficient BackProp. In: **Neural Networks: tricks of the trade**. [s.l.] : Springer, 1998. b. p. 44.

LEE, K. Il et al. Application of artificial neural networks to the analysis of two-dimensional fluorescence spectra in recombinant E coli fermentation processes. **Journal of Chemical Technology and Biotechnology**, [s. l.], v. 80, n. 9, p. 1036–1045, 2005.

LI, M. et al. Prediction of gas solubility in polymers by back propagation artificial neural network based on self-adaptive particle swarm optimization algorithm and chaos theory. **Fluid Phase Equilibria**, [s. l.], v. 356, p. 11–17, 2013. Disponível em: <<http://dx.doi.org/10.1016/j.fluid.2013.07.017>>

LIN, B. et al. A systematic approach for soft sensor development. **Computers and Chemical Engineering**, [s. l.], v. 31, n. 5–6, p. 419–425, 2007.

LIU, W. et al. A survey of deep neural network architectures and their applications. **Neurocomputing**, [s. l.], v. 234, n. October 2016, p. 11–26, 2017. Disponível em: <<http://dx.doi.org/10.1016/j.neucom.2016.12.038>>

LU, N. et al. PCA-based modeling and on-line monitoring strategy for uneven-length batch processes. **Industrial & Engineering Chemistry Research**, [s. l.], v. 43, p. 3343–3352, 2004.

MACIAS, J. J.; ANGELOV, P.; ZHOU, X. A method for predicting quality of the crude oil distillation. **Proceedings of the 2006 International Symposium on Evolving Fuzzy Systems, EFS'06**, [s. l.], v. 00, p. 214–220, 2006.

MANNING-DAHAN, T. PCA and Autoencoders. [s. l.], 2017. Disponível em: <http://tylermd.com/pdf/pca_ae.pdf>

MASSARON, L.; BOSCHETTI, A. **Regression Analysis with Python**. [s.l.] : Packt Publishing Ltd, 2016.

MCCONVILLE, F. X. Functions for easier curve fitting. **Chemical Engineering**, [s. l.], v. 115, n. 12, p. 48–51, 2008.

MISHKIN, D.; SERGIEVSKIY, N.; MATAS, J. Systematic evaluation of CNN advances on the ImageNet. [s. l.], 2016. Disponível em: <<http://arxiv.org/abs/1606.02228><http://dx.doi.org/10.1016/j.cviu.2017.05.007>>

NAIR, V.; HINTON, G. E. Rectified Linear Units Improve Restricted Boltzmann Machines. **Proceedings of the 27th international conference on Machine learning**, [s. l.], p. 8, 2010. Disponível em: <<https://www.cs.toronto.edu/~hinton/absps/reluICML.pdf>>

NIELSEN, M. A. **Neural Networks and Deep Learning**. [s.l.] : Determination Press, 2015. Disponível em: <neuralnetworksanddeeplearning.com>

NIU, F. et al. HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. [s. l.], 2011. Disponível em: <<https://arxiv.org/abs/1106.5730>>. Acesso em: 26 jul. 2019.

NOEL, C.; OSINDERO, S. Dogwild!—Distributed Hogwild for CPU & GPU. [s. l.], p. 1–6, 2014. Disponível em: <<http://web.stanford.edu/~rezab/nips2014workshop/submits/dogwild.pdf>>

O'SHEA, T. J.; KARRA, K.; CLANCY, T. C. Learning to Communicate: Channel Auto-encoders, Domain Specific Regularizers, and Attention. [s. l.], 2016. Disponível em: <<https://arxiv.org/abs/1608.06409>>. Acesso em: 26 jul. 2019.

PEARSON, R. K. Outliers in process modeling and identification. **IEEE Transactions on Control Systems Technology**, [s. l.], v. 10, n. 1, p. 55–63, 2002.

PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. **J. Mach. Learn. Res.**, [s. l.], v. 12, p. 2825–2830, 2011. Disponível em: <<http://dl.acm.org/citation.cfm?id=1953048.2078195>>

QIN, S. J.; YUE, H.; DUNIA, R. Self-Validating Inferential Sensors with Application to Air Emission Monitoring. **Industrial and Engineering Chemistry Research**, [s. l.], v. 36, n. 5, p. 1675–1685, 1997.

R.ISERMANN. Supervision FDD Methods - An Introduction. **Methods**, [s. l.], v. 5, n. 5, p. 639–652, 1997.

RADIUK, P. M. Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets. **Information Technology and Management Science**, [s. l.], v. 20, n. 1, p. 20–24, 2018.

RANZAN, C. et al. Wheat flour characterization using NIR and spectral filter based on ant colony optimization. **Chemometrics and Intelligent Laboratory Systems**, [s. l.], v. 132, p. 133–140, 2014. Disponível em: <<http://dx.doi.org/10.1016/j.chemolab.2014.01.012>>

REDDI, S. J.; KALE, S.; KUMAR, S. On the Convergence of Adam and Beyond. [s. l.], 2019. Disponível em: <<https://arxiv.org/abs/1904.09237>>. Acesso em: 26 jul. 2019.

RHINEHART, R. R. Automated steady and transient state identification in noisy processes. **2013 American Control Conference**, [s. l.], p. 4477–4493, 2014.

RUDER, S. An overview of gradient descent optimization algorithms. [s. l.], 2016. Disponível em: <<https://arxiv.org/abs/1609.04747>>. Acesso em: 26 jul. 2019.

SAKURADA, M.; YAIRI, T. Anomaly detection using autoencoders with nonlinear dimensionality reduction. **Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis**, [s. l.], p. 4, 2014.

SANTOS, P. V. J. L. et al. K-RANK: An Evolution of Y-Rank for Multiple Solutions Problem. **Brazilian Journal of Chemical Engineering**, [s. l.], v. 36, p. 409–419, 2019. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0104-66322019000100409&nrm=iso>

SARLE, W. S. Artificial Neural Networks and Statistical Model. **Japanese journal of applied statistics**, [s. l.], v. 24, n. 2, p. 77–88, 1994.

SCHULTZ, E. dos S. **A importância do ponto de operação nas técnicas de Self-Optimizing Control**. 2015. Universidade Federal do Rio Grande do Sul, [s. l.], 2015.

SCHWENK, H.; BENGIO, Y. Boosting Neural Networks. **Neural Computation**, [s. l.], v. 12, n. 8, p. 1869–1887, 2000.

SHANG, C. et al. Data-driven soft sensor development based on deep learning technique. **Journal of Process Control**, [s. l.], v. 24, n. 3, p. 223–233, 2014. Disponível em: <<http://dx.doi.org/10.1016/j.jprocont.2014.01.012>>

SHI, S. et al. Benchmarking state-of-the-art deep learning software tools. **Proceedings - 2016 7th International Conference on Cloud Computing and Big Data, CCBDB 2016**, [s. l.], p. 99–104, 2017.

THE THEANO DEVELOPMENT TEAM et al. Theano: A Python framework for fast computation of mathematical expressions. [s. l.], p. 1–19, 2016. Disponível em: <<http://arxiv.org/abs/1605.02688>>

VENKATASUBRAMANIAN, V. et al. A review of fault detection and diagnosis. Part III: Process history based methods. **Computers and Chemical Engineering**, [s. l.], v. 27, p.

327–346, 2003.

VILALTA, R.; DRISSI, Y. A Perspective View And Survey Of Meta-Learning. **Artificial Intelligence Review**, [s. l.], v. 18, p. 77–95, 2002.

WANG, X.; KRUGER, U.; IRWIN, G. W. Process monitoring approach using fast moving window PCA. **Industrial and Engineering Chemistry Research**, [s. l.], v. 44, n. 15, p. 5691–5702, 2005.

YU, D. et al. An Introduction to Computational Networks and the Computational Network Toolkit. **Microsoft Technical Report**, [s. l.], v. 112, n. MSR-TR-2014-112, 2015.

ZEILER, M. D. ADADELTA: An Adaptive Learning Rate Method. [s. l.], 2012. Disponível em: <<http://arxiv.org/abs/1212.5701>>

Apêndice A

Código utilizado para geração das bases de dados artificiais utilizadas. Disponível em https://github.com/henriquebm18/gimscop_autoencoder

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Jun 18 07:53:18 2019

@author: henrique

Geração de dados para estudos de inferências com autoencoders

"""
#%%
import numpy as np
from numpy.random import seed
import pandas as pd
from os import sep
#%%
seed(18)
df = pd.DataFrame(np.random.rand(1001, 4), columns=['x1', 'x2', 'x4', 'x5'])
#%%
df_LL = df.copy(deep=True)
df_LL['x3'] = 3*df_LL['x1']-2*df_LL['x2']
df_LL['x6'] = 2*df_LL['x5']-0.5*df_LL['x4']
df_LL['y'] = df_LL['x3']+df_LL['x6']
#%%
df_LN = df.copy(deep=True)
df_LN['x3'] = 3*df_LN['x1']+2*df_LN['x2']
df_LN['x6'] = 2*df_LN['x5']+0.5*df_LN['x4']
df_LN['y'] = 2**(1-
np.log(df_LN['x3']))+3*df_LN['x6']**2+np.log10(df_LN['x6'])
#%%
```

```

df_NL = df.copy(deep=True)
df_NL['x3'] = (df_NL['x1'])**2+0.3*np.log10(df_NL['x2'])
df_NL['x6'] = df_NL['x5']+0.5**(1-np.log(df_NL['x4']))
df_NL['y'] = 2*df_NL['x3']+3*df_NL['x6']
###
df_NN = df.copy(deep=True)
df_NN['x3'] = (df_NN['x1'])**2+0.3*np.log10(df_NN['x2'])**2
df_NN['x6'] = df_NN['x5']+0.5**(1+np.log(df_NN['x4']))
df_NN['y'] =
2**(1+np.log(df_NN['x3']))+3*df_NN['x6']**2+np.log10(df_NN['x6'])
###
# df_LL.to_csv('data'+sep+'dataset_LL', index=False)
# df_LN.to_csv('data'+sep+'dataset_LN', index=False)
# df_NL.to_csv('data'+sep+'dataset_NL', index=False)
# df_NN.to_csv('data'+sep+'dataset_NN', index=False)

###
df_LL_noise = df_LL + pd.DataFrame(np.random.rand(1001, 7),
columns=df_LL.columns)*0.01
df_LN_noise = df_LN + pd.DataFrame(np.random.rand(1001, 7),
columns=df_LL.columns)*0.01
df_NL_noise = df_NL + pd.DataFrame(np.random.rand(1001, 7),
columns=df_LL.columns)*0.01
df_NN_noise = df_NN + pd.DataFrame(np.random.rand(1001, 7),
columns=df_LL.columns)*0.01
###
df_LL_noise.to_csv('data'+sep+'dataset_LL_noise', index=False)
df_LN_noise.to_csv('data'+sep+'dataset_LN_noise', index=False)
df_NL_noise.to_csv('data'+sep+'dataset_NL_noise', index=False)
df_NN_noise.to_csv('data'+sep+'dataset_NN_noise', index=False)

```

Apêndice B

Implementação utilizada para varredura de diferentes autoencoders. Também disponível em https://github.com/henriquebm18/gimscop_autoencoder

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Aug 5 06:50:08 2018

@author: henrique

Este arquivo é parte da dissertação de mestrado Autoencoders para criação de
inferências na indústria química.
Funciona com as versões mais recentes do WinPython (3.5 em diante) sem
necessidade de instalação de bibliotecas adicionais
Necessita o acompanhamento de datasets; ver função read_data

"""
#%%
import logging
import numpy as np
import pandas as pd
import seaborn as sns
# import statsmodels.api as sm
import matplotlib.pyplot as plt
import tensorflow as tf
tf.logging.set_verbosity(tf.logging.ERROR)
from os import makedirs, listdir, sep, path
from itertools import compress
from math import ceil
from scipy.stats import f, t
from time import time, localtime, strftime
from keras.layers import Input, Dense, Dropout
from random import shuffle, sample, choice, uniform
from keras.models import Model, load_model, Sequential
from keras.optimizers import RMSprop, SGD, Adadelta, Nadam
from keras.callbacks import EarlyStopping
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.linear_model import LinearRegression, LassoLarsIC, Ridge
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
# from sklearn import linear_model

#%%
# from numpy.random import seed
# seed(18)
```

```

# from tensorflow import set_random_seed
# set_random_seed(18)
# %%
def writeLog(text, to_see=True):
    """
    Retorna o que está sendo despejado no log ou não
    """
    if to_see:
        print(text)

    logger.warning(text)

def defLogger(name, folder):
    """
    Define configuracoes basicas do logger

    Returns
    -----
    Retorna uma instancia do objeto logger
    """
    mkdirs(folder, exist_ok=True)
    mkdirs(folder+sep+'logs', exist_ok=True)

    LOG_FORMAT = "%(asctime)s - %(message)s"
    logging.basicConfig(filename = folder+sep+'logs'+sep+name,
                        level = logging.WARNING,
                        format = LOG_FORMAT,
                        filemode = 'a')

    logger = logging.getLogger()
    return logger

def read_data(dataset, feed_composition_as_input=False):
    """
    Le e carrega o dataset que deve estar salvo em uma pasta chamada data no
    mesmo local.

    feed_composition_as_input chama considera a composição de entrada das
    colunas de destilação como entradas e é utilizada para modelagem do sistema.

    Returns
    -----
    Retorna o dataset em pandas, nomes das colunas de entrada e nomes das
    colunas de saída
    """
    if dataset == 'COL1':
        dataset = pd.read_csv('data'+sep+'COL1.csv')
        dataset.columns = dataset.columns.str.strip()
        dataset.drop([0, 1], inplace=True)
        if feed_composition_as_input:

```

```
        compositions = ['YETANO', 'YPROPENO', 'YPROPANO',
                        'YPES', 'XETANO', 'XPROPENO', 'XPROPANO']
        inputs = ['Z propeno', 'F in', 'D/F', 'RR', 'QREF', 'QCOND',
                 'FTOPO', 'FFUNDO', 'TEMPTOPO', 'TEMPFUND', 'DELTAP']
    else:
        compositions = ['Z propeno', 'YETANO', 'YPROPENO',
                        'YPROPANO', 'YPES', 'XETANO', 'XPROPENO',
'XPROPANO']
        inputs = ['F in', 'D/F', 'RR', 'QREF', 'QCOND',
                 'FTOPO', 'FFUNDO', 'TEMPTOPO', 'TEMPFUND', 'DELTAP']
    elif dataset == 'COL2':
        dataset = pd.read_csv('data'+sep+'COL2.csv')
        dataset.columns = dataset.columns.str.strip()
        if feed_composition_as_input:
            compositions = ['ypropym', 'ypropam',
                            'yetanom', 'xpropym', 'xpropam', 'xetanom']
            inputs = ['FEED', 'ZPROPENO', 'ZPROPANO', 'ZETANO', 'RR', 'F/C',
'TOPOMAS',
                    'FUNDOMAS', 'FPRES', 'FTEMP', 'QCOND', 'QREF',
'DELTAP1', 'DELTAP2']
        else:
            compositions = ['ZPROPENO', 'ZPROPANO', 'ZETANO', 'ypropym',
                            'ypropam', 'yetanom', 'xpropym', 'xpropam',
'xetanom']
            inputs = ['FEED', 'RR', 'F/C', 'TOPOMAS', 'FUNDOMAS',
                    'FPRES', 'FTEMP', 'QCOND', 'QREF', 'DELTAP1',
'DELTAP2']
    elif dataset == 'COL3':
        dataset = pd.read_csv('data'+sep+'COL3.csv')
        dataset = dataset.drop(['Unnamed: 0'], axis=1)
        dataset.columns = dataset.columns.str.strip()
        if feed_composition_as_input:
            compositions = ['C11PROPA', 'C11PROPE', 'C15PROPA', 'C15PROPE']
            inputs = ['Vazão', 'Z Propeno', 'B8', 'B4', 'C11MAS', 'C11TEMP',
'C11PRESS', 'C15MAS', 'C15TEMP', 'C14AMASS',
                    'C16AMASS', 'B2WORK', 'B22WORK', 'B6HEAT', 'B21HEAT',
                    'TEMPTOPO', 'TEMPFUND', 'PRESFUND', 'C9VFRAC',
'RETMASS']
        else:
            compositions = ['Z Propeno', 'C11PROPA',
                            'C11PROPE', 'C15PROPA', 'C15PROPE']
            inputs = ['Vazão', 'B8', 'B4', 'C11MAS', 'C11TEMP', 'C11PRESS',
'C15MAS', 'C15TEMP', 'C14AMASS',
                    'C16AMASS', 'B2WORK', 'B22WORK', 'B6HEAT', 'B21HEAT',
                    'TEMPTOPO', 'TEMPFUND', 'PRESFUND', 'C9VFRAC',
'RETMASS']
    elif dataset[:8] == 'dataset_':
        dataset = pd.read_csv('data'+sep+dataset+'.csv')
        inputs = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6']
        compositions = ['y']
    else:
```

```

        print('not found')
    return dataset, inputs, compositions

def _files_in_workspace(where):
    """
    Função auxiliar para escrita de resultados em csvs

    Returns
    -----
    Retorna lista de pastas e arquivos.csvs na pasta where
    """
    _files = listdir(where)
    folders = []
    csvs = []
    for item in _files:
        if path.isdir(where+sep+item) and item[0] != '.':
            folders += [item]
        if item[-3:] == 'csv':
            csvs += [item]
    return folders, csvs

def read_evaluations_from_csv(work_folder, merge_all=True):
    """
    Concatena todos os resultados já escritos em csvs

    merge_all por padrão retorna um dataframe com todos as metricas
    observadas nos autoencoders,
    se falso, retorna dicionário separando os resultados por pasta
    """
    _, csvs = _files_in_workspace(work_folder)
    dfs = {}
    if len(csvs) == 0:
        print('no results written here')
        return
    # elif len(csvs) > 1:
    #     print(csvs)
    #     chosen = input('Which one? (1 for the first, 2 for the second...\n
    ')
    else:
        # chosen=1
        for files in csvs:
            dfs[files[:-4]] = pd.read_csv(work_folder+sep+files)
    if merge_all:
        if len(dfs)==1:
            print('there is only {}'.format(list(dfs.keys())[0]))
            return dfs[list(dfs.keys())[0]]
        elif len(dfs)>1:
            complete_df = pd.DataFrame()

```

```

        for name in dfs:
            complete_df = pd.concat([complete_df, dfs[name]])
            complete_df=complete_df.set_index(['model'])
        return complete_df
    else:
        print('how the hell you got here?')
else:
    return dfs

class data_treatment(object):
    """
    tratamento dos dados, deve ser feito para cada dataset levando em
    consideração suas características
    """

    def pretreatment(self, dataset, output_columns='', num_clusters=1,
hotteling=False, y_rank='', scaler_type='Robust'):
        """
        Pre-treatment handler
        """
        writeLog('-----')
        writeLog('-----PreTreatment-----')
        writeLog('-----')
        writeLog('-----')
        writeLog('data shape      : {}'.format(dataset.shape))
        writeLog('clusters        : {}'.format(num_clusters))
        writeLog('Y rank          : {}'.format(y_rank))
        writeLog('Scaler          : {}'.format(scaler_type))
        writeLog('T2 Hotteling    : {}'.format(hotteling))
        writeLog('-----')
        if hotteling:
            self.datatreated = self.hotelling_tsquared(dataset,
output_columns, verbose=True)
        else:
            self.datatreated = dataset

        clusters = self.clusters(self.datatreated, output_columns,
num_clusters)

        if y_rank != '':
            if len(y_rank) != 4:
                print('You need a valid y_rank! Expected format: \n [%Train,
%Test, %Validation , SortingColumnName]')
            return
            [size_train,size_test,size_val,ordinator] = y_rank
        else:
            [size_train,size_test,size_val,ordinator] =
[0.60,0.2,0.2,dataset.columns[-1]]

        if size_train+size_test+size_val != 1:
            print('Your proportions are not equal to ONE!')

```

```

        return

        train_val = pd.concat([self.y_rank_proportion(clusters[i],
ordinator,train = 1-size_test)[0] for i in range(len(clusters))], axis = 0)
        self.test_x = pd.concat([self.y_rank_proportion(clusters[i],
ordinator,train = 1-size_test)[1] for i in range(len(clusters))], axis = 0)
        self.train_x, self.val_x = self.y_rank_proportion(train_val,
ordinator, size_train/(size_train+size_val))
        if len(self.test_x) == 0:
            print('Zero samples selected for Testing - you need less
clusters or more samples!!')
            return

        if output_columns != '':
            self.train_y = self.train_x.loc[:, output_columns]
            self.train_x = self.train_x.drop(output_columns, axis = 1)
            self.test_y = self.test_x.loc[:, output_columns]
            self.test_x = self.test_x.drop(output_columns, axis = 1)
            self.val_y = self.val_x.loc[:, output_columns]
            self.val_x = self.val_x.drop(output_columns, axis = 1)
            self.datatreated_y = self.datatreated.loc[:, output_columns]
            self.datatreated_x = self.datatreated.drop(output_columns, axis
= 1)
        else:
            self.datatreated_x = self.datatreated

        if (scaler_type == 'Robust' or scaler_type == 'Standard'):
            self.train_x = self.scaler(self.train_x, type=scaler_type)
            self.test_x = self.scaler(self.test_x, scaler_obj=self.Scaler)
            self.val_x = self.scaler(self.val_x, scaler_obj=self.Scaler)
            self.datatreated_x = self.scaler(self.datatreated_x,
scaler_obj=self.Scaler)
        else:
            print('Data Not Scaled - Scaler not implemented yet')

        if output_columns != '':
            return {'train_x' : self.train_x, 'train_y': self.train_y,
                    'test_x' : self.test_x, 'test_y': self.test_y,
                    'val_x' : self.val_x, 'val_y': self.val_y,
                    'datatreated_x' : self.datatreated_x,
                    'datatreated_y' : self.datatreated_y}
        else:
            return {'train_x' : self.train_x,
                    'test_x' : self.test_x,
                    'val_x' : self.val_x,
                    'datatreated' : self.datatreated_x}

    def clusters(self, dataset, output_columns='', k=1):

```



```

"""
    call the function: cluster_1,cluster_2,...,cluster_k =
clusters(X,k)\n
    The funcion returns clusters in format of DataFrames\n
    default: k = 1\n
    X is the DataFrame of data\n
    k is the number of clusters\n
"""

if k > len(dataset):
    print('More clusters than inputs!! - now you have
'+str(len(dataset)) + ' clusters!')
    k = len(dataset)-1

if output_columns != '':
    Y = (dataset.drop(output_columns, axis=1)).values
else:
    Y = dataset.values
km = KMeans(n_clusters = k,init='k-
means++',n_init=10,max_iter=300,tol=1e-05,random_state=10)
y_km = km.fit_predict(Y)
Clusters = [dataset.iloc[y_km == i,:] for i in range(k)]
Clusters = list(compress(Clusters,[len(Clusters[i])>0 for i in
range(len(Clusters))]))
return Clusters

def scaler(self, dataset, type=None, scaler_obj=None):
dataset_scaled = dataset.copy()
if scaler_obj == None:
    if type == 'Standard':
        Scaler = StandardScaler()
        Scaler.fit(dataset)
    elif type == 'Robust':
        Scaler = RobustScaler()
        Scaler.fit(dataset)
    else:
        print('no scaler defined')
dataset_scaled.iloc[:,:] = Scaler.transform(dataset)
else:
    if type != None:
        print('scaler_obj passed, ignoring the type input')
        Scaler = scaler_obj
        dataset_scaled.iloc[:,:] = Scaler.transform(dataset)
self.Scaler = Scaler
return dataset_scaled

def hotelling_tsquared(self, dataset, output_columns='', alpha=0.01,
to_plot=False, verbose=True):
data = dataset.copy()
Scaler = RobustScaler()
pc = PCA()

```

```

    if output_columns != '':
        pca =
pc.fit_transform(Scaler.fit_transform(data.drop(output_columns, axis = 1)))
    else:
        pca = pc.fit_transform(Scaler.fit_transform(data))
x = pca.T
cov = np.cov(x)
w = np.linalg.solve(cov, x)
data['T2'] = (x * w).sum(axis=0)
(samples, variables) = data.shape
f_stat = f.ppf(1-alpha, variables, samples-variables)
t2_limit = variables*(samples-1)/(samples-variables)*f_stat
dataset_cleared = data[data['T2']<=t2_limit].drop(columns='T2')
if to_plot:
    with sns.axes_style("whitegrid"):
        plt.figure()
        plt.plot(data['T2'])
        plt.plot(range(len(pca)), np.ones(len(pca))*t2_limit,
                 label='Limiar T2 Hotelling (99% confiabilidade)',
                 linestyle="--", linewidth=5, color='red')
        plt.legend(loc='best')
        plt.ylabel('T2 Hotelling PCA', fontsize=13)
        plt.xlabel('Sample', fontsize=13)
        plt.show()
if verbose:
    writeLog('Dropped {} points'.format(samples-
dataset_cleared.shape[0]))
return dataset_cleared

def y_rank_proportion(self, dataset, y_column, train=0.65):
    """
    Algoritmo para separar as amostras em grupos de calibracao e teste,
    na proporcao desejada e seguindo a metodologia de systematic
sampling modificada
    """
    data = dataset.copy()
    data = data.sort_values(by=[y_column], ascending = True)
    if len(data)*train == int(len(data)*train):
        size_train = int(len(data)*train)
    else:
        size_train = int(len(data)*train) + 1
    size_test = len(data) - size_train
    DNA = []
    if len(data) == 1:
        DNA = ['c']
    elif len(data) == 2:
        DNA = ['c', 't']
    elif len(data) == 3:

```

```

    DNA = ['c', 't', 'c']
else:
    if size_train == 1:
        size_train = 2
        size_test = size_test - 1
    if size_test == 0:
        size_test = 1
        size_train = size_train - 1
    if size_train < size_test:
        RNA_m = ['c'] + ['t']*ceil(size_test/(size_train - 1))
        RNA_t = RNA_m*(size_train)
        DNA = RNA_t[0:len(data)]
        DNA[-1] = 'c'
        if DNA.count('c') != size_train:
            auxC = list(loc for loc, val in enumerate(DNA) if val ==
'c')

            counter = -2
            while DNA.count('c') != size_train:
                DNA[auxC[counter] -1] = 'c'
                counter = counter - 1
    if size_train > size_test:
        RNA_m = ['c']*ceil((size_train - 1)/(size_test+1)) + ['t']
        RNA_t = RNA_m*(size_test + 1)
        DNA = RNA_t[0:len(data)]
        DNA[-1] = 'c'
        if DNA.count('c') != size_train:
            auxC = list(loc for loc, val in enumerate(DNA) if val ==
't')

            counter = -1
            while DNA.count('c') != size_train:
                DNA[auxC[counter]-1] = 't'
                counter = counter - 1
    if size_train == size_test:
        RNA_t = ['c', 't']*(size_train - 1)
        DNA = RNA_t + ['t'] + ['c']
data['indexx'] = DNA
if len(data) == 1:
    calibracao = data.loc[data['indexx'] == 'c',:].drop('indexx',
axis = 1)
    teste = pd.DataFrame()
elif len(data) == 2:
    calibracao = data.loc[data['indexx'] == 'c',:].drop('indexx',
axis = 1)
    teste = data.loc[data['indexx'] == 't',:].drop('indexx', axis =
1)
else:
    calibracao = data.loc[data['indexx'] == 'c',:].drop('indexx',
axis = 1)
    teste = data.loc[data['indexx'] == 't',:].drop('indexx', axis =
1)
if (len(data.T),) == teste.shape:

```

```

        teste = pd.DataFrame(teste).T
    if (len(data.T),) == calibracao.shape:
        calibracao = pd.DataFrame(calibracao).T
    return calibracao, teste

def pca_dimension_analysis(self, dataset_scaled, output='', to_plot=''):
    pcs = PCA()
    pcs.fit_transform(dataset_scaled, output)
    pcs_vars = pcs.explained_variance_ratio_
    pcs_var_sum = np.cumsum(pcs_vars)
    n = range(pcs_vars.shape[0])
    if to_plot:
        plt.bar(n, pcs_vars)
        plt.scatter(n, pcs_var_sum)
        plt.title('PCs Var')
        plt.ylabel('%')
        plt.xlabel('PCs')
        plt.legend(['train', 'val'], loc='center right')
        plt.show()

class Autoencoder_Sweeper(object):
    def __init__(self, data, work_folder, architecture='shallow',
just_eval=''):
        """
        Inicia a varredura dos hiper-parametros do autoencoder, os quais
        podem ser ajustados e escolhidos abaixo.

        data é um dicionario com o dataset separado em treino, teste e
        validação;
        work_folder é o nome da pasta onde serão salvos os autoencoders;
        architecture, por padrão treina autoencoders rasos. Passando '525'
        treina uma rede com 3 camadas internas e dois neuronios no espaço latente.
        just_eval quando True faz apenas a avaliação dos autoencoders dadas
        as metricas indicadas aqui, sem treinar AEs
        """
        self.work_folder = work_folder
        mkdirs(work_folder, exist_ok=True)

        self.metrics = ['accuracy', 'msle', 'mse']

        self.header = ['model', 'hidden_layers', 'optimizer',

'loss_train', '{}_train'.format(self.metrics[0]), '{}_train'.format(self.metri
cs[1]), '{}_train'.format(self.metrics[2]),

'loss_test', '{}_test'.format(self.metrics[0]), '{}_test'.format(self.metrics[
1]), '{}_test'.format(self.metrics[2]),

```

```
'loss_val', '{}_val'.format(self.metrics[0]), '{}_val'.format(self.metrics[1])
, '{}_val'.format(self.metrics[2]),

'loss_all_data', '{}_all_data'.format(self.metrics[0]), '{}_all_data'.format(s
elf.metrics[1]), '{}_all_data'.format(self.metrics[2]))

    self.datatreated = [data['train_x'].values, data['test_x'].values,
data['val_x'].values, data['datatreated_x'].values]

    self.losses = ['mse']
    # self.activations = ['relu', 'softplus', 'softsign', 'tanh',
'linear', 'selu', 'hard_sigmoid', 'sigmoid']
    self.activations = ['selu', 'relu', 'tanh', 'linear', 'sigmoid']
    # self.activations = ['selu']
    # self.optimizers = ['SGD', 'RMSprop', 'Adadelta', 'Adam', 'Nadam']
    self.optimizers = ['Nadam']
    # self.optimizers = [RMSprop(lr=0.0001, decay=1e-6)]
    # self.optimizers = Nadam(lr=0.002, beta_1=0.9, beta_2=0.999,
epsilon=None, schedule_decay=0.004)
    self.wei_inits = ['lecun_uniform', 'glorot_uniform', 'he_uniform',
'lecun_normal', 'glorot_normal', 'he_normal']
    # self.wei_inits = ['glorot_uniform']

    if just_eval:
        self.write_evaluations()
    else:
        if architecture == 'shallow':
            self.shallow_aes_trainer()
        elif len(architecture) == 3:
            self.deep_aes_trainer(architecture)
        else:
            print('architecture not expected')

    def shallow_aes_trainer(self):
        makedirs(self.work_folder+sep+'shallow_models', exist_ok=True)
        train_x = self.datatreated[0]
        val_x = self.datatreated[2]
        variables = train_x.shape[1]

        epochs = 100000
        es = EarlyStopping(monitor='val_loss', min_delta=0, patience=1000,
verbose=1, mode='auto')
        # Rplt = ReduceLROnPlateau(monitor='val_loss', factor=0.8,
patience=500, verbose=0)

        neurons = range(variables-1, 0, -1)
        neurons = [4]

        shuffle(self.activations)
```

```

        how_many =
len(self.activations)**2*len(self.losses)*len(self.optimizers)*len(neurons)*
len(self.wei_inits)
        writeLog('-----')
        writeLog('-----shallow-models-----')
        writeLog('-----')
        writeLog('Gridsearch for {} configurations'.format(how_many))
        writeLog('-----')
        writeLog('metrics observed: {}'.format(self.metrics))
        writeLog('activations      : {}'.format(self.activations))
        writeLog('losses          : {}'.format(self.losses))
        writeLog('optimizers      : {}'.format(self.optimizers))
        writeLog('weighth inits   : {}'.format(self.wei_inits))
        writeLog('-----')
        start_time = time()
        counter = 0

# zzz = pd.DataFrame(columns=['ae', 'e', 'h'])

        for neuron in neurons:
            for act1 in self.activations:
                for act2 in self.activations:
                    for opt in self.optimizers:
                        for w_i in self.wei_inits:
                            for loss in self.losses:

                                counter += 1
                                current = '{}_{}_{}_{}_{}_{}'.format(neuron,
act1, act2, loss, opt, w_i)
                                print('{} of {} => {}'.format(counter,
how_many, current))

                                already_done =
listdir(self.work_folder+sep+'shallow_models')

                                if any(current in s for s in already_done):
                                    print('Already Done! Skipped.')
                                else:
                                    time_counter = time()
                                    input_layer = Input(shape = (variables,
))

                                    encoded_layer = Dense(neuron,
activation=act1, kernel_initializer=w_i)(input_layer)
                                    output_layer = Dense(variables,
activation=act2, kernel_initializer=w_i)(encoded_layer)

                                    encoder = Model(input_layer,
encoded_layer)

```

```

output_layer)

autoencoder = Model(input_layer,

autoencoder.compile(optimizer=opt,

encoder.compile(optimizer=opt,

loss=loss, metrics=self.metrics)

loss=loss, metrics=self.metrics)

h = autoencoder.fit(x=train_x,
                    y=train_x,
                    epochs=epochs,

batch_size=int(len(train_x)/2),

shuffle=True,

validation_data=(val_x, val_x),

verbose=0,
callbacks=[es])

print('trained in
{s}'.format(int(time()-time_counter)))

autoencoder.save(self.work_folder+sep+'shallow_models'+sep+current+'_AUTOENC
ODER.hdf5')

encoder.save(self.work_folder+sep+'shallow_models'+sep+current+'_ENCODER.hdf
5')

encoder, h]

# zzz.iloc[current] = [autoencoder,

# zzz.to_hdf('sweeper.hdf5', key='obj',
mode='a')

lasted_for = time() - start_time
writeLog('\nperformed in {} minutes'.format(int(lasted_for/60)))

def deep_aes_trainer(self, architecture):
    mkdirs(self.work_folder+sep+'{}_models'.format(architecture),
exist_ok=True)
    train_x = self.datatreated[0]
    val_x = self.datatreated[2]
    variables = train_x.shape[1]

    epochs = 100000
    es = EarlyStopping(monitor='val_loss', min_delta=0.001,
patience=100, verbose=1, mode='auto')
    # Rplt = ReduceLROnPlateau(monitor='val_loss', factor=0.8,
patience=500, verbose=0)

    hidden1 = int(architecture[0])
    hidden2 = int(architecture[1])
    hidden3 = int(architecture[2])

```

```

shuffle(self.activations)

how_many =
len(self.activations)**2*len(self.losses)*len(self.optimizers)*len(self.wei_
inits)
writeLog('-----')
writeLog('-----deep-models-----')
writeLog('-----')
writeLog('Gridsearch for {} configurations'.format(how_many))
writeLog('-----')
writeLog('metrics observed: {}'.format(self.metrics))
writeLog('activations      : {}'.format(self.activations))
writeLog('losses           : {}'.format(self.losses))
writeLog('optimizers        : {}'.format(self.optimizers))
writeLog('weighth inits     : {}'.format(self.wei_inits))
writeLog('-----')

start_time = time()
counter = 0

for act1 in self.activations:
    for act2 in self.activations:
        for opt in self.optimizers:
            for w_i in self.wei_inits:
                for loss in self.losses:
                    counter += 1
                    current =
'{}_{}_{}_{}_{}_{}'.format(architecture, act1, act2, loss, opt, w_i)
                    print('{} of {} => {}'.format(counter, how_many,
current))

                    already_done =
listdir(self.work_folder+sep+'{}_models'.format(architecture))

                    if any(current in s for s in already_done):
                        print('Already Done! Skipped.')
                    else:
                        time_counter = time()

                        input_layer = Input(shape=(variables, ))
                        encoding_layer = Dense(hidden1,
activation=act1, kernel_initializer=w_i)(input_layer)
                        encoded_layer = Dense(hidden2,
activation=act1, kernel_initializer=w_i)(encoding_layer)
                        decoding_layer = Dense(hidden3,
activation=act2, kernel_initializer=w_i)(encoded_layer)
                        output_layer = Dense(variables,
activation=act2, kernel_initializer=w_i)(decoding_layer)

```



```

encoder = Model(input_layer, encoded_layer)
autoencoder = Model(input_layer,
output_layer)

autoencoder.compile(optimizer=opt,
loss=loss, metrics=self.metrics)

autoencoder.fit(x=train_x,
                y=train_x,
                epochs=epochs,

batch_size=int(len(train_x)/2),

                shuffle=True,
                validation_data=(val_x,
val_x),

                verbose=0,
                callbacks=[es])
inter_time = time()

autoencoder.save(self.work_folder+sep+'{}_models'.format(architecture)+sep+c
urrent+'_AUTOENCODER.hdf5')

encoder.save(self.work_folder+sep+'{}_models'.format(architecture)+sep+curre
nt+'_ENCODER.hdf5')

writeLog('trained in {} and saved in
{s}'.format(int(inter_time-time_counter), int(time()-inter_time)))

lasted_for = time() - start_time
writeLog('\nperformed in {} hours and {}
minutes'.format(int(lasted_for/3600), int(lasted_for/60)))

def _evaluate_model(self, model):
    """
    Função auxiliar que carrega o autoencoder e faz a avaliação dos
    diferentes grupos de treino[0], teste[1] e validação[2] bem como de todo
    dataset[3] nas metricas observadas no __init__
    """
    try:
        autoencoder = load_model(model)
    except Exception as e:
        writeLog("++++++++++++++++++++++++++++++++ error at
"+str(model)+'\n'+str(e))
        return []
    errors = []
    errors +=
autoencoder.evaluate(self.datatreated[0],self.datatreated[0],batch_size=len(
self.datatreated[0]), verbose=0)
    errors +=
autoencoder.evaluate(self.datatreated[1],self.datatreated[1],batch_size=len(
self.datatreated[1]), verbose=0)

```

```

        errors +=
autoencoder.evaluate(self.datatreated[2],self.datatreated[2],batch_size=len(
self.datatreated[2]), verbose=0)
        errors +=
autoencoder.evaluate(self.datatreated[3],self.datatreated[3],batch_size=len(
self.datatreated[3]), verbose=0)
        return errors

    def write_evaluations(self):
        """
        Gera ou completa arquivos csvs com as metricas observadas dos
autoencoders treinados.
        chama a _evaluate_model
        """
        folders, csvs = _files_in_workspace(self.work_folder)

        # if len(folders) == 1:
        #     folder = folders[0]
        # else:
        #     print(folders)
        #     _posit = input('Which folder? [1 for the first, 2 for the
second and goes on]\n ')
        #     folder = folders[int(_posit)-1]
        for folder in folders:
            print('\n{} folder:'.format(folder))
            if any(folder+'.csv' in x for x in csvs):
                started = True
                done =
list(pd.read_csv(self.work_folder+sep+folder+'.csv')['model'])
                done = [model_name + '_AUTOENCODER.hdf5' for model_name in
done]
            else:
                started = False

            ae_models = []
            for item in listdir(self.work_folder+sep+folder):
                if item[-16:] == 'AUTOENCODER.hdf5':
                    ae_models += [item]
            if started:
                ae_models = list(set(ae_models)-set(done))

            if ae_models == []:
                print('all models are already in the csv file, nothing to do
here')
            else:
                results = pd.DataFrame(columns = self.header)
                if started:
                    print('appending new results to the csv')

```

```

        else:
            results.to_csv(self.work_folder+sep+folder+'.csv',
index=False, mode='a')

            counter = 0
            writeLog("Writting {} evaluations to
csv".format(len(ae_models)))
            for mod_ in ae_models:
                start_time = time()
                _evals =
self._evaluate_model(self.work_folder+sep+folder+sep+mod_)
                if _evals == []:
                    writeLog("++++++++++++++++++++ error at
model {}".format(mod_))
                else:
                    results.loc[0] = [mod_.replace('_AUTOENCODER.hdf5',
''), mod_.split('_')[0], mod_.split('_')[-2]] + _evals
                    results.to_csv(self.work_folder+sep+folder+'.csv',
index=False, header=False, mode='a')
                    counter += 1
                    elapsed_time = time() - start_time
                    print('{} / {} - in {} seconds'.format(counter,
len(ae_models), int(elapsed_time)))
                    # if elapsed_time > 3:
                    #     print('_____')
                    #     restartkernel()

def main():
    inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername)
    # inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
just_eval=True)
    # inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
architecture='979')
    # inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
architecture='969')
    # inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
architecture='959')
    # inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
architecture='949')
    # inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
architecture='939')
    # inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
architecture='878')
    # inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
architecture='868')
    # inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
architecture='858')
    # inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
architecture='848')

```

```
# inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
architecture='838')
# inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
architecture='767')
# inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
architecture='757')
# inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
architecture='747')
# inst = Autoencoder_Sweeper(data=data_treated, work_folder=foldername,
architecture='737')

#%%
if __name__ == '__main__':

    foldername = 'COL2_602020'

    logger = defLogger('{}.log'.format(strftime('%Y_%m_%d', localtime())),
foldername)

    data, inputs, compositions = read_data('COL2')

    data_treated = data_treatment().pretreatment(data,
                                                    output_columns =
compositions,
                                                    num_clusters=1,
                                                    hotteling=True,
                                                    y_rank=[0.60, 0.20, 0.20,
'F/C'],
                                                    scaler_type='Standard')

    # D/F -%Hh%Mmin%Ss
    # F/C
    # C11TEMP
    main()
#%%
```

Apêndice C

Códigos adicionais utilizados para geração de gráficos, comparação de resultados e predição de variáveis dos estudos de caso.

Disponível em https://github.com/henriquebm18/gimscop_autoencoder.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
```

```
Created on Sun Aug 5 06:50:08 2018
```

```
@author: henrique
```

Este arquivo é parte da dissertação de mestrado Autoencoders para criação de inferências na indústria química.

Necessita o acompanhamento do dataset: COL1 ou 2.npy

```
"""
#%%
import numpy as np
import pandas as pd
import seaborn as sns
# import statsmodels.api as sm
import matplotlib.pyplot as plt
import tensorflow as tf
tf.logging.set_verbosity(tf.logging.ERROR)
from os import makedirs, listdir, sep, path
from itertools import compress
from math import ceil
from scipy.stats import f, t
from time import time, localtime, strftime
from keras.layers import Input, Dense, Dropout
from random import shuffle, sample, choice, uniform
from keras.models import Model, load_model, Sequential
from keras.optimizers import RMSprop, SGD, Adadelta, Nadam
from keras.callbacks import EarlyStopping
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.linear_model import LinearRegression, LassoLarsIC, Ridge
from sklearn.metrics import r2_score, mean_squared_error,
mean_squared_log_error
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

#%%
# from numpy.random import seed
# seed(18)
# from tensorflow import set_random_seed
# set_random_seed(18)
```

```
###
```

```
def read_data(dataset, feed_composition_as_input=False):
    """
    Lê e carrega o dataset que deve estar salvo em uma pasta chamada data no
    mesmo local.
    feed_composition_as_input chama considera a composição de entrada das
    colunas de destilação como entradas e é utilizada para modelagem do sistema.

    Returns
    -----
    Retorna o dataset em pandas, nomes das colunas de entrada e nomes das
    colunas de saída
    """
    if dataset == 'COL1':
        dataset = pd.read_csv('data'+sep+'COL1.csv')
        dataset.columns = dataset.columns.str.strip()
        dataset.drop([0, 1], inplace=True)
        if feed_composition_as_input:
            compositions = ['YETANO', 'YPROPENO', 'YPROPANO',
                           'YPES', 'XETANO', 'XPROPENO', 'XPROPANO']
            inputs = ['Z propeno', 'F in', 'D/F', 'RR', 'QREF', 'QCOND',
                     'FTOPO', 'FFUNDO', 'TEMPTOPO', 'TEMPFUND', 'DELTAP']
        else:
            compositions = ['Z propeno', 'YETANO', 'YPROPENO',
                           'YPROPANO', 'YPES', 'XETANO', 'XPROPENO',
                           'XPROPANO']
            inputs = ['F in', 'D/F', 'RR', 'QREF', 'QCOND',
                     'FTOPO', 'FFUNDO', 'TEMPTOPO', 'TEMPFUND', 'DELTAP']
    elif dataset == 'COL2':
        dataset = pd.read_csv('data'+sep+'COL2.csv')
        dataset.columns = dataset.columns.str.strip()
        if feed_composition_as_input:
            compositions = ['ypropym', 'ypropam',
                           'yetanom', 'xpropym', 'xpropam', 'xetanom']
            inputs = ['FEED', 'ZPROPENO', 'ZPROPANO', 'ZETANO', 'RR', 'F/C',
                     'TOPOMAS',
                     'FUNDOMAS', 'FPRES', 'FTEMP', 'QCOND', 'QREF',
                     'DELTAP1', 'DELTAP2']
        else:
            compositions = ['ZPROPENO', 'ZPROPANO', 'ZETANO', 'ypropym',
                           'ypropam', 'yetanom', 'xpropym', 'xpropam',
                           'xetanom']
            inputs = ['FEED', 'RR', 'F/C', 'TOPOMAS', 'FUNDOMAS',
                     'FPRES', 'FTEMP', 'QCOND', 'QREF', 'DELTAP1',
                     'DELTAP2']
```

```

elif dataset == 'COL3':
    dataset = pd.read_csv('data'+sep+'COL3.csv')
    dataset = dataset.drop(['Unnamed: 0'], axis=1)
    dataset.columns = dataset.columns.str.strip()
    if feed_composition_as_input:
        compositions = ['C11PROPA', 'C11PROPE', 'C15PROPA', 'C15PROPE']
        inputs = ['Vazão', 'Z Propeno', 'B8', 'B4', 'C11MAS', 'C11TEMP',
'C11PRESS', 'C15MAS', 'C15TEMP', 'C14AMASS',
                'C16AMASS', 'B2WORK', 'B22WORK', 'B6HEAT', 'B21HEAT',
                'TEMPTOPO', 'TEMPFUND', 'PRESFUND', 'C9VFRAC',
'RETMASS']
    else:
        compositions = ['Z Propeno', 'C11PROPA',
                        'C11PROPE', 'C15PROPA', 'C15PROPE']
        inputs = ['Vazão', 'B8', 'B4', 'C11MAS', 'C11TEMP', 'C11PRESS',
'C15MAS', 'C15TEMP', 'C14AMASS',
                'C16AMASS', 'B2WORK', 'B22WORK', 'B6HEAT', 'B21HEAT',
                'TEMPTOPO', 'TEMPFUND', 'PRESFUND', 'C9VFRAC',
'RETMASS']
elif dataset[:8] == 'dataset_':
    dataset = pd.read_csv('data'+sep+dataset+'.csv')
    inputs = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6']
    compositions = ['y']
else:
    print('not found')
return dataset, inputs, compositions

def _files_in_workspace(where):
    """
    Função auxiliar para escrita de resultados em csvs

    Returns
    -----
    Retorna lista de pastas e arquivos.csvs na pasta where
    """
    _files = listdir(where)
    folders = []
    csvs = []
    for item in _files:
        if path.isdir(where+sep+item) and item[0] != '.':
            folders += [item]
        if item[-3:] == 'csv':
            csvs += [item]
    return folders, csvs

def read_evaluations_from_csv(work_folder, merge_all=True):
    """
    Concatena todos os resultados já escritos em csvs

    merge_all por padrão retorna um dataframe com todos as metricas
    observadas nos autoencoders,

```

```

se falso, retorna dicionário separando os resultados por pasta
"""
_, csvs = _files_in_workspace(work_folder)
dfs = {}
if len(csvs) == 0:
    print('no results written here')
    return
# elif len(csvs) > 1:
#     print(csvs)
#     chosen = input('Which one? (1 for the first, 2 for the second...\n
')
else:
    # chosen=1
    for files in csvs:
        dfs[files[:-4]] = pd.read_csv(work_folder+sep+files)
if merge_all:
    if len(dfs)==1:
        print('there is only {}'.format(list(dfs.keys())[0]))
        return dfs[list(dfs.keys())[0]]
    elif len(dfs)>1:
        complete_df = pd.DataFrame()
        for name in dfs:
            complete_df = pd.concat([complete_df, dfs[name]])
            complete_df=complete_df.set_index(['model'])
        return complete_df
    else:
        print('how the hell you got here?')
else:
    return dfs

def old_read_evaluations_from_csv(work_folder, merge_all=True):
    _files = listdir(work_folder)
    csvs = []
    for item in _files:
        if item[-3:] == 'csv':
            csvs += [item]
    dfs = {}
    if len(csvs) == 0:
        print('no results written here')
        return dfs
    else:
        for files in csvs:
            dfs[files[:-4]] = pd.read_csv(work_folder+sep+files)

if merge_all == True:
    if len(dfs) == 1:
        print('there is only {}'.format(list(dfs.keys())[0]))
        return dfs[list(dfs.keys())[0]]

```



```
elif len(dfs) > 1:
    complete_df = pd.DataFrame()
    for name in dfs:
        complete_df = pd.concat([complete_df, dfs[name]])
    complete_df = complete_df.set_index(['model'])
    return complete_df
else:
    print('how the hell you got here?')
return dfs

def load_aes(work_folder, criterium, architecture='', how_many=1):
    dfs = read_evaluations_from_csv(work_folder)
    if architecture:
        path = work_folder+sep+'{}_models'.format(architecture)+sep
        df = dfs['{}_models'.format(architecture)]
    else:
        path = work_folder+sep+'shallow_models'+sep
        df = dfs['shallow_models']

    for met_ in ['loss_train', 'loss_test', 'loss_val', 'loss_all_data']:
        df[met_] = df[met_]**(0.5)
        df.rename(columns={met_: met_.replace('loss', 'rmse')},
inplace=True)
    df.set_index(['model'], inplace=True)

    if 'accuracy' in criterium:
        aes_df = df.sort_values([criterium], ascending=False).head(how_many)
    else:
        aes_df = df.sort_values([criterium], ascending=True).head(how_many)

    for name in aes_df.index:
        aes_df.loc[name, 'autoencoder'] =
load_model(path+name+'_AUTOENCODER.hdf5')
        aes_df.loc[name, 'encoder'] = load_model(path+name+'_ENCODER.hdf5')

    return aes_df

def plot_ae_reconstruction(ae_model, data, scatter='', subtracted=''):
    ae_result = pd.DataFrame(ae_model.predict(data), columns=data.columns,
index=data.index)
    if subtracted:
        residual = ae_result-data
        for iterator in range(residual.shape[1]):
            ax = plt.subplot(3, 4, iterator+1)
            if scatter:
plt.scatter(range(len(residual)),abs(residual.values[:,iterator]), color =
'b', s = 7)
                plt.title(residual.columns[iterator])
            else:
                plt.plot(abs(residual.values[:,iterator]))
```

```

        plt.title(residual.columns[iterator])
    else:
        for iterator in range(ae_result.shape[1]):
            ax = plt.subplot(3, 4, iterator+1)
            if scatter:
                plt.scatter(range(len(data)),data.values[:,iterator], color
= 'k', s = 5)
                plt.scatter(range(len(data)),ae_result.values[:,iterator],
color = 'b', s = 7)
                plt.title(data.columns[iterator])
            else:
                plt.plot(data.values[:,iterator], color = 'k')
                plt.plot(ae_result.values[:,iterator], color = 'b')
                plt.title(data.columns[iterator])

    # pylab.savefig(model_to_load+'.png')
    # https://stackoverflow.com/questions/24116318/how-to-show-residual-in-the-bottom-of-a-matplotlib-plot

def plot_shallow_ae_weights(ae_model):
    weights_encoder = pd.DataFrame(ae_model.get_weights()[0],
                                   index=inputs,
                                   columns=['n{}'.format(x+1) for x in
range(encoded_size)])
    # bias_encoder = ae_model.get_weights()[1]

    weights_decoder = pd.DataFrame(ae_model.get_weights()[2],
                                   index=['n{}'.format(x+1) for x in
range(encoded_size)],
                                   columns=inputs)
    # bias_decoder = ae_model.get_weights()[3]

    sns.heatmap(weights_encoder.transpose(),
                cmap="coolwarm",
                center=0,
                annot=True,
                fmt=".1f",
                linewidth=.5)
    sns.heatmap(weights_decoder,
                cmap="coolwarm",
                center=0,
                annot=True,
                fmt=".1f",
                linewidth=.5)

def _bic(sse, n, k):
    return n*np.log(sse/n)+k*np.log(n)

```

```

def _aic(sse, n, k):
    return 2*k - 2*np.log(sse)

def linear_model_from_encoded(encoded, data_treated, identifier, y_name):
    # identifier - train, test or val como string
    # encoder = load_model(mainpath+sep+folder+sep+model+'_ENCODER.hdf5')
    x = pd.DataFrame(encoded.predict(data_treated['test_x'].values))
    x2 = sm.add_constant(x)

    y_real = list(data_treated['test_y'][y_name])
    log10_y_real = np.log10(y_real)
    ln_y_real = np.log(y_real)

    ys = [y_real, log10_y_real, ln_y_real]
    ty = ['normal', 'log10', 'ln']
    models = pd.DataFrame(columns=['None', 'Lasso', 'Ridge'])
    for y, t in zip(ys, ty):
        model = sm.OLS(y, x)
        result = model.fit()
        print(result.summary())
        result_lasso = model.fit_regularized(L1_wt=1.0)
        print(result_lasso.summary())
        result_ridge = model.fit_regularized(L1_wt=0.0)
        print(result_ridge.summary())
        models.loc[t] = [result, result_lasso, result_ridge]

        # if rest=='LassoLars':
        #     lin_model = linear_model.LassoLarsIC()
        # elif rest=='Ridge':
        #     lin_model = linear_model.Ridge()
        # else:
        #     lin_model = linear_model.LinearRegression()

        # lin_model.fit(encoder_results, y_real)
        # y_hat = lin_model.predict(encoder_results)

        # rss = sum((y_real-y_hat)**2)
        # n_samples = data_treated['{}_x'.format(identifier)].shape[0]
        # n_params = sum(1 for x in lin_model.coef_ if x != 0)
        # R2 = r2_score(y_real, y_hat)
        # MSE = mean_squared_error(y_real, y_hat)
        # RMSE = np.sqrt(mean_squared_error(y_real, y_hat))
        # AIC = _aic(rss, n_samples, n_params)
        # BIC = _bic(rss, n_samples, n_params)

        # print('coef: {}'.format(lin_model.coef_))
        # print('R2: {}'.format(R2))
        # print('mse: {}'.format(MSE))
        # print('RMSE: {}'.format(RMSE))
        # print('AIC: {}'.format(AIC))

```

```

        # print('BIC: {}'.format(BIC))

    # if plot_chart:
    # plt.figure()
    # plt.scatter(y_real, y_hat)
    # plt.plot(y_real, y_real)
    # plt.title('Target: {} Data Type: {} Restriction: {}'.format(y_name,
    # identifier, 'none'))
    # print('Data Type: {} Model Type: {} \n'.format(identifier, rest))
    # return [y_real, y_hat, lin_model, mean_squared_error(y_real, y_hat)]
    # return [BIC, R2, RMSE, rest, lin_model]
    return models

def _plot2(lm_, lm_LL, x, y):
    plt.figure(figsize=(13, 5), dpi=80, facecolor='w', edgecolor='k')

    plt.subplot(1, 2, 1)
    plt.plot(lm_.predict(x), y, 'o')
    plt.plot(y, y, 'k')
    plt.xlabel('Predictions')
    plt.ylabel('Target Values')
    # plt.title('Linear Regression')

    plt.subplot(1, 2, 2)
    plt.plot(lm_LL.predict(x), y, 'o')
    plt.plot(y, y, 'k')
    plt.xlabel('Predictions')
    plt.ylabel('Target Values')
    # plt.title('Lasso-Lars')

def _plots_and_stats(lm_, lm_LL, lm_RG, x, y):

    plt.figure(figsize=(18, 5), dpi=80, facecolor='w', edgecolor='k')

    plt.subplot(1, 3, 1)
    plt.plot(lm_.predict(x), y, 'ko')
    plt.plot(y, y, 'k')
    plt.xlabel('Predictions')
    plt.ylabel('Real Values')
    plt.title('Linear Regression')

    plt.subplot(1, 3, 2)
    plt.plot(lm_LL.predict(x), y, 'ko')
    plt.plot(y, y, 'k')
    plt.xlabel('Predictions')
    plt.ylabel('Real Values')
    plt.title('Lasso-Lars')

```

```
plt.subplot(1, 3, 3)
plt.plot(lm_RG.predict(x), y, 'ko')
plt.plot(y, y, 'k')
plt.xlabel('Predictions')
plt.ylabel('Real Values')
plt.title('Ridge')

plt.show()

for lms in [lm_, lm_LL, lm_RG]:
    _stats_lm(lms, x, y)

def _stats_lm(lm, x_test, y_test):
    params = np.append(lm.intercept_, lm.coef_)
    predictions = lm.predict(x_test)

    newX = pd.DataFrame({"Constant":
np.ones(len(x_test))}).join(pd.DataFrame(x_test))
    sse = sum((y_test-predictions)**2)

    MSE = (sse)/(len(newX)-len(newX.columns))
    MSLE = sum((np.log(y_test+1)-np.log(predictions+1))**2)/(len(newX)-
len(newX.columns))

    var_b = MSE*(np.linalg.inv(np.dot(newX.T, newX)).diagonal())
    sd_b = np.sqrt(var_b)
    ts_b = params / sd_b

    p_values = [2*(1-t.cdf(np.abs(i), (len(newX)-1))) for i in ts_b]

    sd_b = np.round(sd_b, 3)
    ts_b = np.round(ts_b, 3)
    p_values = np.round(p_values, 3)
    params = np.round(params, 4)

    n_params = sum(1 for x in params if x != 0)
    n_samples = x_test.shape[0]

    BIC = n_samples*np.log(sse/n_samples)+n_params*np.log(n_samples)
    AIC = 2*n_params - 2*np.log(sse)

    myDF3 = pd.DataFrame()
    myDF3["Coefficients"] = params
    myDF3["Std Errors"] = sd_b
    # myDF3["t values"] = ts_b
    myDF3["Probabilites"] = p_values

    print('RMSE: {}'.format(np.sqrt(MSE)))
    print('MSLE {}'.format(MSLE))
    print('R2: {}'.format(lm.score(x_test, y_test)))
    print('AIC: {}'.format(AIC))
```

```

print('BIC: {}'.format(BIC))
print('_____')
print(myDF3)
# print('coef: {}'.format(params))
print('_____')
# print('np: {}'.format(n_params))

def _linear_model(X, y, rest='', centered=''):
    if centered:
        if rest == 'LassoLars':
            lm = LassoLarsIC(fit_intercept=False)
        elif rest == 'Ridge':
            lm = Ridge(fit_intercept=False)
        else:
            lm = LinearRegression(fit_intercept=False)
    else:
        if rest == 'LassoLars':
            lm = LassoLarsIC()
        elif rest == 'Ridge':
            lm = Ridge()
        else:
            lm = LinearRegression()

    lm.fit(X, y)

    # plt.figure()
    # plt.scatter(y_real, y_hat)
    # plt.plot(y_real, y_hat)
    # plt.title('Target: {} Data Type: {} Restriction: {}'.format(y_name,
    # identifier, 'none'))
    # print('Data Type: {} Model Type: {} \n'.format(identifier, rest))
    # return [y_real, y_hat, lin_model, mean_squared_error(y_real, y_hat)]
    return lm

def _plot_pca(pca):
    plt.figure(figsize=(13, 5), dpi=80, facecolor='w', edgecolor='k')

    plt.plot(range(pca.n_components_), np.cumsum(pca.explained_variance_ratio_),
    '-o', label = 'variância explicada acumulada')
    plt.bar(range(pca.n_components_), pca.explained_variance_ratio_, width =
    0.8, align = 'center', tick_label = ['PC.%s' %i for i in
    range(1, pca.n_components_+1)])

    #plt.bar(range(pca.n_components_), np.cumsum(pca.explained_variance_ratio_) -
    pca.explained_variance_ratio_, bottom = pca.explained_variance_ratio_, width =
    0.8, align = 'center')
    plt.tick_params(axis='x', which='major', labelsize=8)

```



```
if to_plot:
    autoencoder.summary()

#declaring encoder as part of the autoencoder
input_img = Input(shape=(variables,))
encoder_layer1 = autoencoder.layers[0]
encoder_layer2 = autoencoder.layers[1]
encoder = Model(input_img,
encoder_layer2(encoder_layer1(input_img)))

# autoencoder.summary()
# encoder.summary()

# stop training criterion
es = EarlyStopping(monitor='val_loss', min_delta=0.0001,
                    patience=100, verbose=1, mode='auto')
opt = Nadam(lr=0.002, beta_1=0.9, beta_2=0.999,
            epsilon=0.001, schedule_decay=0.004)
# opt = RMSprop(lr=0.0001, decay=1e-6)

autoencoder.compile(optimizer=opt,
                    loss='mse',
                    metrics=['accuracy', 'mape', 'msle'])

h = autoencoder.fit(x=train_x,
                  y=train_x,
                  epochs=10000,
                  batch_size=int(len(train_x)/2),
                  shuffle=True,
                  validation_data=(val_x, val_x),
                  verbose=0,
                  callbacks=[es])

if to_plot:
    plot_history(h)
else:
    print(' ')

results_train = autoencoder.evaluate(
    train_x, train_x, batch_size=len(train_x), verbose=0)
results_test = autoencoder.evaluate(
    test_x, test_x, batch_size=len(test_x), verbose=0)
results_val = autoencoder.evaluate(
    val_x, val_x, batch_size=len(val_x), verbose=0)

# plt.plot()

# print('input \t\t output')
```



```

# for iter, iter2 in zip(test_x[0], autoencoder.predict(test_x)[0]):
#     print('{:.4f} / {:.4f}'.format(iter, iter2))

print('          Loss / Accuracy')
print(
    'Train: {:.4f} / {:.1f}%'.format(results_train[0],
results_train[1]*100))
print('Val  : {:.4f} / {:.1f}%'.format(results_val[0],
results_val[1]*100))
print(
    'Test : {:.4f} / {:.1f}%'.format(results_test[0],
results_test[1]*100))
model_rmse = results_test[0]**(0.5)
print('results test rmse: {}'.format(model_rmse))
print('\n')

activ0 = activ_
activ1 = activ_
loss = 'mse'
optimizer = 'nadam'
name = '{}_{}_{}_{}_{}'.format(
    architecture, activ0, activ1, loss, optimizer, wei_inits)

return [model_rmse, name, autoencoder, encoder, h]

def multiple_custom_autoencoders(datatreated, architecture, how_many=5):
    results = pd.DataFrame(
        columns=['ae_rmse_test', 'model', 'ae', 'encoder', 'history'])
    for hm in range(how_many):
        ind_result = custom_autoencoder(datatreated, architecture)
        results.loc[hm] = ind_result
    return results

def multiple_linear_models(datatreated, df_mca='', architecture='757',
how_many=5, plotting=False):
    if df_mca:
        pass
    else:
        df_mca = multiple_custom_autoencoders(
            datatreated, architecture, how_many)
    df_mlm = pd.DataFrame(columns=['BIC', 'R2', 'RMSE', 'rest',
'lin_model'])
    for nn in range(df_mca.shape[0]):
        df_mlm.loc[nn] = linear_model_from_encoded(
            df_mca.loc[nn, 'encoder'], datatreated, 'test', 'YPES',
rest='None', plot_chart=plotting)
    df = pd.concat([df_mlm, df_mca], axis=1)
    return df

```

```

class _dopping(object):
    def manual_dopper(self, df, ratio_of_samples=0.5, number_of_variables=1,
error_size=0.1, spec=''):
        noise_index = list(df.sample(frac=ratio_of_samples).index)
        noise_columns = []
        noise = pd.DataFrame(np.ones(df.shape), index=df.index,
columns=df.columns)
        for row in noise_index:
            var_ = np.random.choice(df.shape[1], number_of_variables)
            if spec:
                noise.loc[row][spec] = 1+error_size*np.random.choice([-1,1])
            else:
                for col in var_:
                    noise.loc[row][col] = 1+error_size*np.random.choice([-
1,1])
                noise_columns += [var_]
        self.noise_index = noise_index
        self.noise_columns = noise_columns
        self.noise = noise
        return noise*df

class data_treatment(object):
    def pretreatment(self, dataset, output_columns='', num_clusters=1,
hotteling=False, y_rank='', scaler_type='Robust'):
        """
        Pre-treatment handler
        """
        print('-----')
        print('-----PreTreatment-----')
        print('-----')
        print('-----')
        print('data shape      : {}'.format(dataset.shape))
        print('clusters          : {}'.format(num_clusters))
        print('Y rank             : {}'.format(y_rank))
        print('Scaler             : {}'.format(scaler_type))
        print('T2 Hotteling      : {}'.format(hotteling))
        print('-----')
        if hotteling:
            self.datatreated = self.hotelling_tsquared(dataset,
output_columns, verbose=True)
        else:
            self.datatreated = dataset

        clusters = self.clusters(self.datatreated, output_columns,
num_clusters)

        if y_rank != '':
            if len(y_rank) != 4:

```

```

        print('You need a valid y_rank! Expected format: \n [%Train,
%Test, %Validation , SortingColumnName]')
        return
        [size_train,size_test,size_val,ordinator] = y_rank
    else:
        [size_train,size_test,size_val,ordinator] =
[0.60,0.2,0.2,dataset.columns[-1]]

    if size_train+size_test+size_val != 1:
        print('Your proportions are not equal to ONE!')
        return

    train_val = pd.concat([self.y_rank_proportion(clusters[i],
ordinator,train = 1-size_test)[0] for i in range(len(clusters))], axis = 0)
    self.test_x = pd.concat([self.y_rank_proportion(clusters[i],
ordinator,train = 1-size_test)[1] for i in range(len(clusters))], axis = 0)
    self.train_x, self.val_x = self.y_rank_proportion(train_val,
ordinator, size_train/(size_train+size_val))
    if len(self.test_x) == 0:
        print('Zero samples selected for Testing - you need less
clusters or more samples!!')
        return

    if output_columns != '':
        self.train_y = self.train_x.loc[:, output_columns]
        self.train_x = self.train_x.drop(output_columns, axis = 1)
        self.test_y = self.test_x.loc[:, output_columns]
        self.test_x = self.test_x.drop(output_columns, axis = 1)
        self.val_y = self.val_x.loc[:, output_columns]
        self.val_x = self.val_x.drop(output_columns, axis = 1)
        self.datatreated_y = self.datatreated.loc[:, output_columns]
        self.datatreated_x = self.datatreated.drop(output_columns, axis
= 1)
    else:
        self.datatreated_x = self.datatreated

    if (scaler_type == 'Robust' or scaler_type == 'Standard'):
        self.train_x = self.scaler(self.train_x, type=scaler_type)
        self.test_x = self.scaler(self.test_x, scaler_obj=self.Scaler)
        self.val_x = self.scaler(self.val_x, scaler_obj=self.Scaler)
        self.datatreated_x = self.scaler(self.datatreated_x,
scaler_obj=self.Scaler)
    else:
        print('Data Not Scaled - Scaler not implemented yet')

    if output_columns != '':
        return {'train_x' : self.train_x, 'train_y': self.train_y,
            'test_x' : self.test_x, 'test_y': self.test_y,
            'val_x' : self.val_x, 'val_y': self.val_y,
            'datatreated_x' : self.datatreated_x,
            'datatreated_y' : self.datatreated_y}

```

```

else:
    return {'train_x' : self.train_x,
            'test_x' : self.test_x,
            'val_x' : self.val_x,
            'datatreated' : self.datatreated_x}

def clusters(self, dataset, output_columns='', k=1):
    """
    call the function: cluster_1,cluster_2,...,cluster_k =
clusters(X,k)\n
    The funcion returns clusters in format of DataFrames\n
    default: k = 1\n
    X is the DataFrame of data\n
    k is the number of clusters\n
    """

    if k > len(dataset):
        print('More clusters than inputs!! - now you have
'+str(len(dataset)) + ' clusters!')
        k = len(dataset)-1

    if output_columns != '':
        Y = (dataset.drop(output_columns, axis=1)).values
    else:
        Y = dataset.values
    km = KMeans(n_clusters = k, init='k-means++', n_init=10,
max_iter=300, tol=1e-05, random_state=10)
    y_km = km.fit_predict(Y)
    Clusters = [dataset.iloc[y_km == i,:] for i in range(k)]
    Clusters = list(compress(Clusters,[len(Clusters[i])>0 for i in
range(len(Clusters))]))
    return Clusters

def scaler(self, dataset, type=None, scaler_obj=None):
    dataset_scaled = dataset.copy()
    if scaler_obj == None:
        if type == 'Standard':
            Scaler = StandardScaler()
            Scaler.fit(dataset)
        elif type == 'Robust':
            Scaler = RobustScaler()
            Scaler.fit(dataset)
        else:
            print('no scaler defined')
        dataset_scaled.iloc[:,:] = Scaler.transform(dataset)
    else:
        if type != None:
            print('scaler_obj passed, ignoring the type input')

```

```

        Scaler = scaler_obj
        dataset_scaled.iloc[:, :] = Scaler.transform(dataset)
    self.Scaler = Scaler
    return dataset_scaled

    def hotelling_tsquared(self, dataset, output_columns='', alpha=0.01,
to_plot=False, verbose=True):
    data = dataset.copy()
    Scaler = RobustScaler()
    pc = PCA()
    if output_columns != '':
        pca =
pc.fit_transform(Scaler.fit_transform(data.drop(output_columns, axis = 1)))
    else:
        pca = pc.fit_transform(Scaler.fit_transform(data))
    x = pca.T
    cov = np.cov(x)
    w = np.linalg.solve(cov, x)
    data['T2'] = (x * w).sum(axis=0)
    (samples, variables) = data.shape
    f_stat = f.ppf(1-alpha, variables, samples-variables)
    t2_limit = variables*(samples-1)/(samples-variables)*f_stat
    dataset_cleared = data[data['T2']<=t2_limit].drop(columns='T2')
    if to_plot:
        with sns.axes_style("whitegrid"):
            plt.figure(figsize=(13, 5), dpi=80, facecolor='w',
edgecolor='k')
            plt.plot(data['T2'], '.')
            plt.plot(range(len(pca)), np.ones(len(pca))*t2_limit,
                    label='Limiar T2 Hotelling (99% confiabilidade)',
                    linestyle="--", linewidth=2, color='red')
            plt.legend(loc='best')
            plt.ylabel('T2 Hotelling PCA', fontsize=13)
            plt.xlabel('Samples', fontsize=13)
            plt.show()
    if verbose:
        print('Dropped {} points'.format(samples-
dataset_cleared.shape[0]))
    return dataset_cleared

    def y_rank_proportion(self, dataset, y_column, train=0.65):
        """
        Algoritmo para separar as amostras em grupos de calibracao e teste,
na proporcao desejada e seguindo a metodologia de systematic
sampling modificada
        """
        data = dataset.copy()
        data = data.sort_values(by=[y_column], ascending = True)
        if len(data)*train == int(len(data)*train):
            size_train = int(len(data)*train)
        else:

```

```

        size_train = int(len(data)*train) + 1
        size_test = len(data) - size_train
        DNA = []
        if len(data) == 1:
            DNA = ['c']
        elif len(data) == 2:
            DNA = ['c','t']
        elif len(data) == 3:
            DNA = ['c','t','c']
        else:
            if size_train == 1:
                size_train = 2
                size_test = size_test - 1
            if size_test == 0:
                size_test = 1
                size_train = size_train - 1
            if size_train < size_test:
                RNA_m = ['c'] + ['t']*ceil(size_test/(size_train - 1))
                RNA_t = RNA_m*(size_train)
                DNA = RNA_t[0:len(data)]
                DNA[-1] = 'c'
                if DNA.count('c') != size_train:
                    auxC = list(loc for loc, val in enumerate(DNA) if val ==
'c')

                    counter = -2
                    while DNA.count('c') != size_train:
                        DNA[auxC[counter] - 1] = 'c'
                        counter = counter - 1
            if size_train > size_test:
                RNA_m = ['c']*ceil((size_train - 1)/(size_test+1)) + ['t']
                RNA_t = RNA_m*(size_test + 1)
                DNA = RNA_t[0:len(data)]
                DNA[-1] = 'c'
                if DNA.count('c') != size_train:
                    auxC = list(loc for loc, val in enumerate(DNA) if val ==
't')

                    counter = -1
                    while DNA.count('c') != size_train:
                        DNA[auxC[counter]-1] = 't'
                        counter = counter - 1
            if size_train == size_test:
                RNA_t = ['c','t']*(size_train - 1)
                DNA = RNA_t + ['t'] + ['c']
        data['indexx'] = DNA
        if len(data) == 1:
            calibracao = data.loc[data['indexx'] == 'c',:].drop('indexx',
axis = 1)
            teste = pd.DataFrame()

```

```
        elif len(data) == 2:
            calibracao = data.loc[data['indexx'] == 'c',:].drop('indexx',
axis = 1)
            teste = data.loc[data['indexx'] == 't',:].drop('indexx', axis =
1)
        else:
            calibracao = data.loc[data['indexx'] == 'c',:].drop('indexx',
axis = 1)
            teste = data.loc[data['indexx'] == 't',:].drop('indexx', axis =
1)
        if (len(data.T),) == teste.shape:
            teste = pd.DataFrame(teste).T
        if (len(data.T),) == calibracao.shape:
            calibracao = pd.DataFrame(calibracao).T
        return calibracao, teste

def pca_dimension_analysis(self, dataset_scaled, output='', to_plot=''):
    pcs = PCA()
    pcs.fit_transform(dataset_scaled, output)
    pcs_vars = pcs.explained_variance_ratio_
    pcs_var_sum = np.cumsum(pcs_vars)
    n = range(pcs_vars.shape[0])
    if to_plot:
        plt.bar(n, pcs_vars)
        plt.scatter(n, pcs_var_sum)
        plt.title('PCs Var')
        plt.ylabel('%')
        plt.xlabel('PCs')
        plt.legend(['train', 'val'], loc='center right')
        plt.show()

def main():
    print('all loaded, now, do something')
    #%%
    if __name__ == '__main__':

        foldername = 'COL1_602020'

        data, inputs, compositions = read_data('COL1')

        data_treated = data_treatment().pretreatment(data,
                                                    output_columns =
compositions,
                                                    num_clusters=2,
                                                    hotteling=True,
                                                    y_rank=[0.60, 0.2, 0.2,
'D/F'],
                                                    scaler_type='Standard')

        # COL1 - D/F
        # COL2 - F/C
```

```
# COL3 - C11TEMP
# datasets artificiais - x1

main()
#%
```

Apêndice D

Algoritmo de adição de dados espúrios nas bases de dados, em recorte, também disponível em: https://github.com/henriquebm18/gimscop_autoencoder

```
class _dopping(object):
    def manual_dopper(self, df, ratio_of_samples=0.5, number_of_variables=1,
error_size=0.1, spec=''):
        noise_index = list(df.sample(frac=ratio_of_samples).index)
        noise_columns = []
        noise = pd.DataFrame(np.ones(df.shape), index=df.index,
columns=df.columns)
        for row in noise_index:
            var_ = np.random.choice(df.shape[1], number_of_variables)
            if spec:
                noise.loc[row][spec] = 1+error_size*np.random.choice([-1,1])
            else:
                for col in var_:
                    noise.loc[row][col] = 1+error_size*np.random.choice([-
1,1])
                noise_columns += [var_]
        self.noise_index = noise_index
        self.noise_columns = noise_columns
        self.noise = noise
        return noise*df
```