

UNIVERSIDADE FEDERAL DO ACRE - UFAC
Programa de Pós-Graduação em Ciência da Computação - PPgCC

EDKALLENN SILVA DE LIMA

Aplicação de Mineração de dados e Aprendizagem de Máquina na detecção de conflitos diretos entre políticas de controle de acesso

RIO BRANCO - ACRE

2021

EDKALLENN SILVA DE LIMA

Aplicação de Mineração de dados e Aprendizagem de Máquina na detecção de conflitos diretos entre políticas de controle de acesso

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal do Acre para a obtenção do Grau de Mestre em Ciência da Computação. Área de concentração: Engenharia de Sistemas e Informação

UNIVERSIDADE FEDERAL DO ACRE – UFAC

Programa de Pós-Graduação em Ciência da Computação – PPgCC

Orientador: PROF[a] DR[a]. LAURA COSTA SARKIS

RIO BRANCO - ACRE

2021

Ficha catalográfica elaborada pela Biblioteca Central da UFAC

L732a Lima, Edkallenn Silva de, 1980- .
Aplicação de mineração de dados e aprendizagem de máquina na detecção de conflitos diretos entre políticas de controle de acesso / Edkallenn Silva de Lima ; Orientadora: Profa. Dra. Laura Costa Sarkis. -- 2021.
131f. : il. ; 30 cm.

Dissertação (Mestrado) - Universidade Federal do Acre. Programa de Pós-Graduação em Ciência da Computação. Mestrado em Ciência da Computação. Área de concentração: Engenharia de Sistemas e Informação. Rio Branco, Acre, 2021.
Inclui referências.

1. Controle de acesso. 2. Mineração de dados. 3. Aprendizagem de máquina. 4. Detecção de conflitos. I. Sarkis, Laura Costa (orientadora). II. Título.

CDD: 004

Bibliotecária Kezia Santos CRB-11/508

EDKALLENN SILVA DE LIMA

Aplicação de Mineração de dados e Aprendizagem de Máquina na detecção de conflitos diretos entre políticas de controle de acesso

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal do Acre para a obtenção do Grau de Mestre em Ciência da Computação. Área de concentração: Engenharia de Sistemas e Informação

Aprovada em JULHO de 2021.

BANCA EXAMINADORA

PROFª. DRª. LAURA COSTA SARKIS - Orientador, UFAC

PROFª. DRª. ANA BEATRIZ ALVAREZ MAMANI, UFAC

PROFº. DR.LUIZ AUGUSTO MATOS DA SILVA, UFAC

PROFª. DR.RAONI SIMÕES FERREIRA, UFAC

RIO BRANCO - ACRE

2021

Dedicatória: Este trabalho não seria possível sem a educação que me foi concedida, os ensinamentos e a sabedoria oriundos de você, Lucimar do Rego Albuquerque de Lima, muito mais que uma mãe, uma inspiração. <IN MEMORIAN>.

Agradecimentos

À Deus pela saúde e pelas condições de viver uma vida plena de significados, sentimentos, emoções e sentido.

À CAPES, à UFAC e a todos os profissionais que fizeram a história dessas instituições que bravamente lutam contra a crescente idiotização do mundo.

À família maravilhosa, em todos os sentidos, que tenho. Pai, mães (sim, mais de uma!), irmãos, irmãs, avós e avôs que se foram (e deixaram saudades eternas), tios, tias, primos e primas, agregados, enfim, todos que fazem a loucura que é qualquer festa "só com os parentes mais próximos". Eu amo todos vocês.

Às mulheres da minha vida, minha filha, Ana Ester e minha esposa, Vanessa Lima, por me suportarem — em todos os sentidos, claro, mas, principalmente por me amarem incondicionalmente.

À todos os professores do PPgCC, os que lecionaram disciplinas, os que lutaram para que o programa existisse, os que participam da banca e, claro, para a minha orientadora pela paciência (muita, muita!) e, sobretudo, carinho.

Mais uma vez, à você, Lucimar, nossa querida Lúcia, por sua sabedoria e ensinamentos. Pelo seu sorriso que sempre vinha cheio de significados. Pela sua vida ter sido a nossa vida. Por ter se doado tanto. Por ter trabalhado tanto para construir esta família linda (que, claro, nunca será a mesma sem você, jamais). Pelas inúmeras vezes em que você me chamava de "meu filho" (se eu soubesse que a contagem era regressiva tinha aproveitado mais). Por tudo o que você representou nas nossas vidas, obrigado. Nenhuma palavra que existe ou que será inventada em qualquer língua tem significado suficiente para descrever o que você era em nossas vidas. Na minha vida. Mais uma vez, obrigado...

Nunca te esquecerei...

"Assim como casas são feitas de pedras, a ciência é feita de fatos. Mas uma pilha de pedras não é uma casa e uma coleção de fatos não é, necessariamente, ciência".

Jules Henri Poincaré, matemático, físico e filósofo da ciência francês

Resumo

Um dos principais componentes de segurança em sistemas computacionais é o controle de acesso que busca garantir meios para proteção, confidencialidade e confiabilidade dos acessos dos usuários aos objetos dentro dos sistemas de uma organização. As políticas de controle de acesso descrevem, assim, as regras de acesso aplicadas aos usuários de um sistema. Em sistemas com múltiplos sujeitos, muitas ações e diversos objetos, eventualmente, ocorrerão conflitos entre políticas. Um conflito ocorre quando os objetivos de duas ou mais políticas não podem ser atendidos simultaneamente em um determinado contexto. Políticas conflitantes podem levar o sistema a estados inconsistentes, tais como, acessos não autorizados ou negações a acessos legítimos. Geralmente, a literatura especializada em detecção de conflitos se baseia em técnicas que analisam as políticas em pares e este tipo de análise pode gerar um problema NP-completo, considerado computacionalmente custoso. Já as abordagens que usam mineração de dados e aprendizagem de máquina, geralmente, não têm a detecção de conflitos entre políticas como foco principal. Este trabalho propõe que o problema da detecção de conflitos em políticas pode ser convertido em um problema de *data mining* (mineração de dados) resolvido pela tarefa da classificação além de modelar e sintetizar uma forma de detectar estes conflitos mediante o uso de diferentes algoritmos e técnicas da aprendizagem de máquina e fornecer modelos genéricos o suficiente que possam ser usados em outros contextos. O trabalho apresenta, portanto: (i) A mineração de dados associada a técnicas de aprendizagem de máquina para detectar os conflitos de modalidade entre políticas; (ii) A análise da inserção de novas regras no sistema utilizando o modelo de aprendizagem de máquina gerado; (iii) O comparativo entre algoritmos de mineração apresentando quais deles possibilitam modelar a melhor forma de detecção automática de conflitos entre políticas (com seus hiperparâmetros) e, finalmente, que (iv) o problema de detectar conflitos entre políticas de controle de acesso pode ser resolvido como um problema de classificação utilizando os algoritmos de mineração de dados e aprendizagem de máquina com boa acurácia e precisão. Para alcançar os objetivos propostos foram realizados diversos e múltiplos experimentos onde foram utilizados os principais algoritmos de classificação com diferentes datasets. Para as avaliações quantitativas e outras análises estatísticas diversas métricas foram empregadas e após os experimentos concluiu-se que as Redes Neurais, o SVM e o Random Forest são os melhores algoritmos para este fim sendo as redes neurais o modelo que apresentou as maiores acurácias médias.

Palavras-chave: Controle de Acesso. Mineração de dados. Aprendizagem de máquina. Conflitos diretos. Detecção de conflitos.

Abstract

One of the main security components in computer systems is access control, which seeks to guarantee means to protection, confidentiality and reliability of user access to objects within an organization's systems. Access control policies thus describe the access rules applied to users of a system. In systems with multiple subjects, many actions and diverse objects will eventually conflict between policies. A conflict occurs when the objectives of two or more policies cannot be met simultaneously in a given context. Conflicting policies can lead the system to inconsistent states, such as unauthorized access or denial of legitimate access. Generally, the specialized literature on conflict detection is based on techniques that analyze policies in pairs and this type of analysis can generate an NP-complete problem, considered computationally expensive. Approaches that use data mining and machine learning, on the other hand, generally do not have the detection of conflicts between policies as their main focus. This work proposes that the problem of detecting conflicts in policies can be converted into a *data mining* problem solved by the classification task, in addition to modeling and synthesizing a way to detect these conflicts through the use of different algorithms and techniques of machine learning and provide models that are generic enough might be used in other contexts. Therefore, the work presents: (i) Data mining associated with machine learning techniques to detect modality conflicts between policies; (ii) Analysis of the insertion of new rules in the system using the generated machine learning model; (iii) The comparison between mining algorithms showing which of them make it possible to model the best way to automatically detect conflicts between policies (with their hyperparameters) and, finally, that (iv) the problem of detecting conflicts between access control policies can be solved as a classification problem using data mining and machine learning algorithms with good accuracy and precision. In order to achieve the proposed objectives, several and multiple experiments were carried out using the main classification algorithms with different datasets. For quantitative evaluations and other statistical analyzes several metrics were used and after the experiments it was concluded that Neural Networks, SVM and Random Forest are the best algorithms for this purpose, with neural networks being the model with the highest accuracy averages.

Keywords: Access control. Data mining. Machine learning. Direct conflicts. Conflict detection.

Lista de ilustrações

Figura 1 – Modelo das políticas utilizadas no estudo 35
Figura 2 – Etapas do processo de descoberta do conhecimento em bases de dados - KDD 37
Figura 3 – Modelo sobreajustado e regularizado para o mesmo *dataset*. 43
Figura 4 – Funcionamento da técnica holdout. 44
Figura 5 – Funcionamento da técnica *cross validation* 45
Figura 6 – Preenchimento de uma matriz de confusão 48
Figura 7 – Matriz de confusão para o caso de um classificador binário 48
Figura 8 – Exemplo do algoritmo k-NN 54
Figura 9 – Modelo matemático de um neurônio 55
Figura 10 – Adição de um *offset* (*bias*) no modelo do neurônio 56
Figura 11 – Fronteira de separação (perceptron com duas entradas) 57
Figura 12 – Representação simplificada de uma RNA 58
Figura 13 – Rede Perceptron de multicamadas 60
Figura 14 – Função de ativação Limiar 61
Figura 15 – Função de ativação Linear 61
Figura 16 – Função de ativação Logística (sigmoide) 62
Figura 17 – Função de ativação Tangente Hiperbólica 62
Figura 18 – Função de ativação ReLu - unidade linear retificada 63
Figura 19 – Vetores de Suporte 66
Figura 20 – Adição de características para tornar um conjunto linearmente separável 66
Figura 21 – Classificador Linear SVM com o uso de características polinomiais . . . 67
Figura 22 – Forma Geral dos experimentos 80
Figura 23 – Parte do programa para geração de políticas automaticamente 82
Figura 24 – Segunda parte do programa para geração de políticas automaticamente 83
Figura 25 – Terceira Parte do programa para geração de políticas automaticamente 84
Figura 26 – Aspecto do arquivo das políticas geradas para os experimentos 84
Figura 27 – Aspecto do arquivo com 68 políticas após o pré-processamento 85
Figura 28 – Saída do software WEKA. Classificador: SVM 86
Figura 29 – Saída do software WEKA. Classificador: *MultiLayer Perceptron* 86
Figura 30 – Parte do arquivo com 281 políticas após o pré-processamento 87
Figura 31 – Aspecto do dataset importado 89
Figura 32 – Engenharia de atributos - dados categóricos textuais 90
Figura 33 – Aspecto dos atributos previsores 90
Figura 34 – Aspecto do atributo classe 91
Figura 35 – Código do **MLPClassifier** com as primeiras iterações 92

Figura 36 – Validações para o modelo **MLPClassifier** 93
Figura 37 – Convergência da Loss Function - **MLPClassifier** 94
Figura 38 – Dimensionalidade dos dados: atributos sujeito, ação e objeto 94
Figura 39 – Comparação visual entre alguns classificadores 95
Figura 40 – Separação dos dados de teste e treino 96
Figura 41 – Implementação da classe Politicas 97
Figura 42 – Implementação da classe que modela a arquitetura da rede 98
Figura 43 – Implementação da função de treino da rede 98
Figura 44 – Implementação da função de teste da rede 99
Figura 45 – Implementação da função que mescla o treino e o teste 99
Figura 46 – Convergência das épocas entre o treino e o teste da MLP 100
Figura 47 – Arquitetura da rede neural 100
Figura 48 – MLPClassifier - resultado dataset 392 políticas 101
Figura 49 – MLPClassifier - hiperparâmetros do modelo 102
Figura 50 – SVM kernel linear - resultado dataset 392 políticas 102
Figura 51 – SVM kernel linear - hiperparâmetros do modelo 102
Figura 52 – Random Forest - resultado dataset 392 políticas 103
Figura 53 – Random Forest - resultado dataset 392 políticas 103
Figura 54 – Batch - código das 150 execuções distintas por algoritmo 105
Figura 55 – Curvas de distribuição das acurácias dos algoritmos 107
Figura 56 – Resultados do teste de Tukey 107
Figura 57 – Comparação visual do resultado do teste estatístico dos classificadores . 108
Figura 58 – Instâncias geradas em formato dataframe 109
Figura 59 – Resultado da predição 110
Figura 60 – Resultado 1 da detecção de conflito 110
Figura 61 – Resultado 2 da detecção de conflito 110
Figura 62 – Resultado 3 da detecção de conflito 111
Figura 63 – Geração políticas 2 112
Figura 64 – Arquivo gerado 113
Figura 65 – Semântica Sujeito 113
Figura 66 – Semântica organização e ação 113
Figura 67 – Função retorna-politica 115
Figura 68 – Função que propaga as politicas 115
Figura 69 – Novas instâncias geradas 116
Figura 70 – Resultados 116

Lista de códigos

Código 3.1 – Código da Padronização 91
Código 3.2 – Hiperparâmetros do MLPClassifier 92
Código 3.3 – Classse Política 114
Código 3.4 – Função que retorna políticas por sujeito 114
Código 3.5 – Políticas propagadas 115

Lista de tabelas

Tabela 1 – Acurácia dos classificadores 85
Tabela 2 – Acurácia do MLP 88
Tabela 3 – Acurácia do SVM 88
Tabela 4 – Acurácia do Random Forest 88
Tabela 5 – Acurácia dos classificadores - dataset 392 políticas 95
Tabela 6 – Coeficiente de variação e valores mínimos/máximos das execuções . . . 104
Tabela 7 – Média e desvio-padrão da distribuição das acurácias por algoritmo . . . 105
Tabela 8 – Médias da distribuição dos resultados dos classificadores 106

Lista de abreviaturas e siglas

ABAC	*Attribute-Based Access Control*
ACPs	*Access Control Policies*
AM	*Aprendizado de Máquina*
ATM	*Air-Traffic Management*
BPNN	*Basic Probabilistic Neural Network*
CPNN	*Constructive Probabilistic Neural Network*
DAC	*Discretionary Access Control*
DARPA	*Defense Advanced Research Projects Agency*
DoS	*Denial of Service*
FN	*False Negative*
FP	*False Positive*
FNR	*False Negative Rate*
FPR	*False Positive Rate*
GPGU	*General Purpose Graphic Processor Unit*
GPU	*Graphic Processor Unit*
KDD	*Knowledege Discovery in Data Bases*
MAC	*Mandatory Access Control*
ML	*Machine Learning*
MLP	*MultiLayer Perceptron*
NFL	*No-Free Lucnch*
PMC	*Perceptron MultiCamadas*
RABAC	*Role and Attribute-Based Access Control*
RBAC	*Role-Based Access Control*

ReBAC	*Relationship-Based Access Control*
RNA	*Rede Neural Artificial*
RTLS	*Real Time Location Systems*
SMA	*Sistema Multi Agentes*
SVM	*Support Vector Machiner*
TI	*Tecnologia da Informação*
TN	*True Negative*
TP	*True Positive*
TNR	*True Negative Rate*
TPR	*True Positive Rate*
XACML	*eXtensible Access Control Markup Language*
XML	*eXtensible Markup Language*

Sumário

1 INTRODUÇÃO 18
1.1 Contextualização 18
1.1.1 Detecção de conflitos 20
1.1.1.1 Classificação dos conflitos 20
1.1.2 Motivação 21
1.1.3 Problema de pesquisa 24
1.1.4 Hipótese 24
1.2 Objetivos 25
1.2.1 Objetivo geral 25
1.2.2 Objetivos específicos 25
1.3 Solução Proposta 25
1.4 Método de Pesquisa 26
1.5 Contribuições diferenciais do Trabalho 27
1.6 Limitações do Trabalho 28
1.7 Organização do trabalho 28

2 REVISÃO BIBLIOGRÁFICA 30
2.1 Segurança de dados computacionais 30
2.2 Controle de acesso 31
2.2.1 Políticas de Controle de acesso 33
2.2.2 Modelos de políticas 34
2.2.3 Modelo de Política utilizado 34
2.3 Mineração de Dados 36
2.3.1 KDD - Knowledge Discovery in Databases 37
2.3.2 Modelo de conhecimento 38
2.3.2.1 Arquitetura do modelo 38
2.4 Aprendizagem de máquina 39
2.4.1 Definição 39
2.5 Algoritmos de classificação 41
2.5.1 Teoria da aprendizagem e algoritmos de classificação 42
2.5.2 Taxa de erro 42
2.5.3 Estratégias de validação 43
2.5.3.1 Medidas de avaliação 46
2.6 Naïve Bayes 50
2.7 K-NN — K-vizinhos mais próximos 51

2.8 Redes Neurais Artificiais - RNA . . . 54
2.8.1 Definição . . . 54
2.8.2 Modelo de neurônio artificial . . . 55
2.8.3 Redes do tipo Perceptron de múltiplas camadas . . . 58
2.8.4 Funções de ativação . . . 59
2.8.4.1 Função de ativação limiar . . . 60
2.8.4.2 Função de ativação linear . . . 61
2.8.4.3 Funções de ativação semilineares . . . 61
2.8.4.4 Função de ativação ReLu . . . 62
2.8.5 Otimização . . . 63
2.8.5.1 TensorFlow e Pytorch . . . 64
2.9 SVM - Support Vector Machines . . . 65
2.9.1 Kernel . . . 67
2.10 Random Forest - Florestas Aleatórias . . . 68
2.11 Trabalhos Relacionados . . . 69
2.11.1 Verificação e Detecção de conflitos . . . 69
2.11.1.1 (KUANG et al., 2020) . . . 69
2.11.1.2 Sarkis (2017) . . . 70
2.11.1.3 Vijayalakshmi e Jayalakshmi (2020) . . . 71
2.11.1.4 Zheng e Xiao (2019) . . . 71
2.11.1.5 Yahiaoui, Zinedine e Harti (2018) . . . 71
2.11.1.6 Silvestre (2017) . . . 71
2.11.1.7 Sun et al. (2011) . . . 72
2.11.1.8 Hwang et al. (2010) . . . 72
2.11.1.9 Shu, Yang e Arenas (2009) . . . 72
2.11.2 Mineração de políticas . . . 72
2.11.2.1 Bui, Stoller e Li (2019) . . . 72
2.11.2.2 Chakraborty, Sandhu e Krishnan (2019) . . . 73
2.11.2.3 Kalaskar e Barkade (2018) . . . 73
2.11.2.4 Xu e Stoller (2013) . . . 73
2.11.2.5 Hachana, Cuppens-Boulahia e Cuppens (2015) . . . 73
2.11.2.6 Martin e Tao Xie (2006) . . . 74
2.11.3 Detecção e Resolução de outros tipos conflitos . . . 74
2.11.3.1 Lupu e Sloman (1999) . . . 74
2.11.4 Mineração de dados e aprendizagem de máquina . . . 74
2.11.4.1 Guerrero-Higueras, DeCastro-Garcia e Matellan (2018) . . . 74
2.11.4.2 Bui, Stoller e Le (2019) . . . 75
2.11.4.3 Obaidat e Macchairolo (1994) e Mukkamala, Janoski e Sung (2002) . . . 75
2.11.4.4 Jin, Cheu e Srinivasan (2002) e Debar, Becker e Siboni (1992) . . . 76

2.11.5 Relacionamento dos trabalhos com a dissertação 76

3 EXPERIMENTOS 79
3.1 Visão geral dos experimentos 79
3.2 Base de dados, pré-processamento e recursos computacionais 80
3.2.1 Datasets 80
3.2.2 Recursos computacionais 83
3.2.3 Experimentos iniciais - arquivo com 68 políticas 83
3.2.4 Resultados - Arquivo com 68 políticas 84
3.3 Outros experimentos - arquivos com 276 e 392 políticas 87
3.3.1 Experimentos com Pandas, NumPy e sklearn 89
3.3.2 Experimentos com Redes Neurais usando TensorFlow e Pytorch 95
3.3.3 Cross-validation, OneHotEncoder e Testes Estatísticos 101
3.3.3.1 Tuning dos parâmetros 103
3.3.3.2 Testes Estatísticos e descoberta do melhor algoritmo 104
3.4 Experimento com novas instâncias 108
3.5 Uma abordagem para os conflitos indiretos 111
3.6 Análise dos resultados 117

4 CONSIDERAÇÕES FINAIS 118

REFERÊNCIAS 120

1 Introdução

Neste capítulo introdutório serão descritos uma contextualização para apresentação do problema da pesquisa, além da justificativa, a hipótese do trabalho, os objetivos, as soluções propostas, resultados, limitações da pesquisa e como este trabalho está organizado.

1.1 Contextualização

De acordo com Alecrim (2019) e Machado (2014), o volume de dados e informações cresce exponencialmente a cada ano, portanto, há uma frequente e ininterrupta demanda por mais infraestrutura de TI nas empresas, instituições, nos governos e mesmo nos usuários domésticos e, mais ainda, por um correto tratamento, destino e interpretação à imensidão de dados gerados por pessoas, empresas e governos. Em uma ampla variedade de campos, os dados estão sendo coletados e acumulados em um ritmo acelerado e há, assim, uma crescente demanda por análise adequada destes (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996; LIMA; PEREIRA, 2012). Neste contexto se insere a mineração de dados com suas técnicas para tratamento e extração de conhecimento desse volume crescente de dados (SILVA; PERES; BOSCARIOLI, 2017; FERRARI; SILVA, 2017).

As *políticas* de proteção, confidencialidade e confiabilidade da informação, como as de *controle de acesso*, sendo parte da área de *segurança computacional*, são uma das formas de garantir, mediante o estabelecimento de regras, padrões e normas a salvaguarda e a disponibilidade das informações dos sistemas (BUI; STOLLER; LE, 2019). O controle de acesso, cf. Ueda (2012, p.1) é um objeto de estudo importante no contexto da segurança de sistemas, porque é um dos elementos principais em qualquer sistema de computação.

Portanto, formular, gerenciar e dar manutenção a políticas de segurança da informação, principalmente as de controle de acesso é um aspecto relevante (LI; TRIPUNITARA, 2006). A construção dessas políticas é direcionada por modelos que geram diversas regras e ferramentas para o funcionamento seguro de sistemas. Todavia, o gerenciamento destas políticas pode se tornar um processo complicado, tendo em vista que deve ser assegurada a eficácia e que o desempenho dos sistemas não seja afetado (UEDA, 2012).

De acordo com Bellettini, Bertino e Ferrari (2001, p. 21), a administração da segurança abrange, entre outras situações, conceder e cancelar permissões de acesso a usuários nos múltiplos objetos a serem protegidos. Assim, essas permissões de acesso, originalmente chamadas de *autorizações*, indicam de que maneira (ou modo) qual sujeito pode acessar um dado objeto em um determinado contexto. Essas *maneiras* ou modos, especificam as operações que podem ser realizadas nos objetos e recursos que estão

protegidos, como, por exemplo, a leitura e gravação de arquivos (BELLETTINI; BERTINO; FERRARI, 2001, p. 21).

Sempre que o número de sujeitos, objetos e ações é elevado a quantidade de tais "autorizações" pode se tornar extremamente grande. Deste modo, se a população de usuários deste sistema for altamente dinâmica, a quantidade de operações de concessão e revogação dessas "autorizações" a serem realizadas poderá se tornar muito difícil de gerenciar (BELLETTINI; BERTINO; FERRARI, 2001, p. 21).

Neste contexto é que se insere a motivação e a justificativa da utilização das técnicas e algoritmos de mineração de dados e aprendizagem de máquina com o objetivo precípuo de atingir este problema específico que é a quantidade crescente dessas citadas "autorizações" tornar o gerenciamento complexo e demorado. Importante salientar que neste trabalho, comumente, as "autorizações" são referenciadas como *políticas*, termo mais genérico e com mais suporte acadêmico.

Uma **política**, como as de controle de acesso, descrevem qual ação um sujeito (em um sistema) pode fazer (*permissão*), não pode fazer (*proibição*) ou é obrigado a fazer (*obrigação*) sobre um objeto em um dado contexto (SARKIS, 2017). E são, de acordo com Ueda (2012, p. 1), "Um tema de pesquisa importante dentro do contexto de segurança de sistemas, pois é um dos componentes fundamentais em qualquer sistema de computação".

Conforme Silva (2018), políticas de controle de acesso são criadas fundamentadas nas políticas de segurança do qual o sistema fará parte, e estas devem ser representadas e exprimidas em políticas de controle de acesso bem definidas e não-ambíguas. Para Wang et al. (2010), apesar do controle de acesso parecer, de formal conceitual, algo simples, ele é complicado e suscetível, na prática, a erros. Para a garantia da consistência das políticas de controle de acesso e para cumprir os requisitos corretos de segurança não deve haver conflito através da aplicação destas políticas.

Embora exista a necessidade das políticas serem bem definidas e não-ambíguas, muitas vezes as políticas são complexas e ambíguas dependendo de decisões de acesso ligadas à aplicação de diferentes regras e fatores, como leis, normas e condições ambientais como data, hora e local. As políticas de controle de acesso devem, assim, ser determinadas com o objetivo de satisfazer todos os requisitos de segurança estabelecidos contemplando todas as políticas de segurança da organização além de possíveis falhas de segurança decorrentes do uso contínuo do sistema (SILVA, 2018; SANDHU; SAMARATI, 1994).

Em contextos reais, contudo, muitas vezes as políticas de segurança apresentam conflitos entre si. Estes surgem quando, por exemplo, duas políticas regulando o mesmo comportamento de determinado objeto em um sistema estão ativas, mas uma delas obriga (ou permite) a realização de determinado comportamento ou ação enquanto a outra proíbe o mesmo (LUPU; SLOMAN, 1999; SILVESTRE, 2017). Ou seja, quando o cumprimento

de uma regra/política/norma viola a outra e vice-versa. A característica de um sistema que tem a capacidade de reconhecer um estado inconsistente como este que esteja em andamento ou prestes a ocorrer intitula-se detecção de conflitos.

Com o exposto, este trabalho usa tarefas da mineração de dados para *modelar uma hipótese que possibilite detectar conflitos diretos em políticas* de controle de acesso. Para isso, algoritmos de classificação serão explorados, descritos e utilizados com ênfase nos que tem relação com o aprendizado de máquina e outras técnicas de classificação.

1.1.1 Detecção de conflitos

Segundo Kalam et al. (2003), quando um modelo de controle de acesso inclui a possibilidade de especificar *permissões*, *proibições* e *obrigações* podem ocorrer alguns conflitos entre políticas.

Os conflitos podem acontecer quando diferentes conjuntos de condições resultam em *permitir* e *negar* simultaneamente, ao mesmo papel, à mesma solicitação, ou *proibir* e *obrigar* o mesmo papel, à mesma solicitação, isto é, quando os *objetivos* de duas ou mais políticas *não podem ser atendidos simultaneamente* (CUPPENS; CUPPENS-BOULAHIA; GHORBEL, 2007).

A detecção de conflitos se insere dentro do contexto de verificação de consistência de estado dos sistemas. Esta checagem corresponde a analisar a aderência das regras de uma determinada política dentro do cenário em que ela é aplicada e levando em consideração as demais políticas incorporadas. Um conflito é, portanto, uma inconsistência, ou estado inconsistente representado por uma violação de uma regra já que impede que uma ou mais políticas sejam atendidas/aplicadas (UEDA, 2012).

1.1.1.1 Classificação dos conflitos

Dunlop, Indulska e Raymond (2002) usam uma abordagem para os conflitos diretos e simples que será replicada neste trabalho. Diz-se que duas regras estão em conflito *quando o cumprimento de uma das regras viola a outra e vice-versa.* Ou seja, a verificação se há o conflito é feita entre duas políticas que possuem modalidades contraditórias ou antagônicas, definidas na mesma organização, executadas pelos mesmos sujeitos, efetuando a mesma ação em relação a um objeto específico.

Exemplo de um *conflito de modalidade ou direto*:

{P1= Permitido, na Universidade X, Ana Ester, acessar processos administrativos}

{P2= Proibido, na Universidade X, Ana Ester, acessar processos administrativos}

Os exemplos acima mostram que quando uma política proíbe e a outra permite um sujeito de realizar uma ação estabelecida sobre um objeto específico em uma organização particular ocorre um conflito direto. O conflito, segundo Autrel, Coma e al (2008), pode ser identificado fundamentalmente utilizando a sobreposição dos atributos das políticas.

Já em um *conflito indireto ou de aplicação* as políticas conflitantes regulam ações diferentes mas, relacionadas, executadas por distintos sujeitos porém, correlacionados sobre objetos desiguais mas, de alguma forma, semanticamente associados, em organizações diferentes porém, em algum sentido, relacionadas (CUPPENS; CUPPENS-BOULAHIA; GHORBEL, 2007). Além disso, um conflito indireto pode ainda ocorrer, mesmo quando as políticas em conflito não têm modalidades contraditórias ou contrárias (SLOMAN; LUPU, 2002; LUPU; SLOMAN, 1999).

Ex:

```
P3 = Proibido, Empresa E, Funcionário, receber, avaliação, mensal

P4= Permitido, Empresa E, Analista, conceder, avaliação, mensal
```

Este conflito não seria detectado diretamente, porém há um conflito se considerarmos os relacionamentos já que o tipo de política é contraditória ($Proibido \times Permitido$) e um 'Analista' também é um um 'Funcionário' e vice-versa. A capacidade de um sistema reconhecer um estado inconsistente acontecendo ou em potencial de ocorrer é denominada **detecção de conflitos.**

Para Sarkis (2017, p. 14), identificar conflitos entre políticas de controle de acesso é passo importante para impedir a ocorrência de imprecisões e falhas no sistema em relação à aplicação de políticas, considerando que a ausência de políticas em conflito refletem o projeto de segurança do sistema de modo correto. Assim, é importante para um sistema que utiliza políticas de controle de acesso detectar os conflitos que, porventura, possam existir entre políticas existentes e novas. Torna-se, assim, relevante a utilização de abordagens que analisem previamente a existência de conflitos como o proposto nesta dissertação.

1.1.2 Motivação

Na detecção de conflitos em políticas, geralmente, conforme a revisão da literatura (explicitada no Capítulo 2), usam-se abordagens como as de Sarkis (2017) e Mohan et al. (2012) — estritamente analíticas e formais, baseadas em análise de ontologias entre os atributos que compõem uma política e os relacionamentos e regras de propagação destas políticas ou mesmo estratégias como as de Wang et al. (2010) e Ferraiolo, Atluri e Gavrila (2011) que usam uma arquitetura que pode ser empregada como uma máquina de proteção de propósito geral usando métodos matemáticos formais para atingir seus objetivos.

Dentro do conjunto de modelos formais, envolvendo relacionamentos, na detecção

de conflitos, há também o trabalho de Sun et al. (2011), que usa o "propósito envolvido nos modelos de controle de acesso" para atingir seu fim. *Estes, fatalmente, analisam as políticas em pares ou em grupos sem filtros de agrupamentos.*

Há também os métodos baseados em requisitos como os propostos por He e Antón (2009) que integram a especificação de política no processo de desenvolvimento de software, garantindo, assim, a consistência entre os artefatos de software e fornecendo orientação prescritiva sobre como especificar ACPs (*Access Control Policies)* — políticas de controle de acesso, mas que *tem o inconveniente de ocorrer durante o desenvolvimento e a fase de análise do sistema e não em tempo de execução do mesmo.*

Adicionalmente, na literatura especializada são encontrados procedimentos como os descritos em Silvestre (2017) que utilizam lógica deôntica[1] para encontrar os conflitos. Estas abordagens citadas definem tipos de conflitos que podem ocorrer entre políticas computacionais e os tentam detectá-los, cada uma usando uma perspectiva específica.

Para detectar os conflitos entre políticas, nestes trabalhos citados anteriormente, estas foram analisadas, geralmente, em pares (e sem filtros para agrupamentos — quando os há) e mesmo quando foram verificadas múltiplas normas ou políticas, toda a base precisou ser "consultada" ou "varrida" novamente a cada conjunto de novas instâncias de políticas inseridas ou analisadas no sistema (para que o conflito seja ou não detectado).

De acordo com Shoham e Tennenholtz (1995) *esta forma de analisar políticas em pares é um problema NP-completo*[2], ou seja, ainda não foi provado que esta classe de problemas pode ser resolvida *em tempo polinomial*, sendo assim, *são tratados como computacionalmente custosos* a cada vez que uma instância nova de política é analisada (em tempo de execução, sendo, normalmente, exponencial).

Consequentemente, com o crescimento orgânico, natural e temporal das políticas em um sistema computacional, a manutenção e o gerenciamento dessas políticas será, por si só, além de já um problema naturalmente complexo (e preterido) bem como de difícil administração, de acordo com Ueda (2012) e Bellettini, Bertino e Ferrari (2001), também será, eventualmente, um problema computacionalmente oneroso, de acordo com Shoham e Tennenholtz (1995).

Hwang et al. (2010), assim como Sarkis (2017), também desenvolve uma ferramenta, chamada ACPT (*Access Control Test Policy*), que ajuda a modelar e implementar políticas corretamente durante a modelagem, implementação e verificação de políticas, mas tem a desvantagem (ao menos em termos de custo computacional) de, também, verificar as políticas em pares (ou, ainda, até em grupos maiores) ou não contemplar essa análise em tempo de execução.

---

[1] A lógica *deôntica* é um tipo de lógica usada para analisar de modo formal as normas e as proposições que tratam dessas normas (SILVESTRE, 2017)

[2] Para maiores informações sobre NP-completude, ver Aho, Hopcroft e UIIman (1974)

Alternativamente, técnicas e algoritmos de aprendizagem de máquina juntamente com as de mineração de dados foram utilizadas com resultados promissores na detecção de conflitos, principalmente em Obaidat e Macchairolo (1994), Chen (2011), bem como em Christodoulou e Kontogeorgou (2008) e Jin, Cheu e Srinivasan (2002).

Estes estudos abordam problemas variados como detecção de colisões em voos, segurança de acesso computacional, incidentes em rodovias e intrusão de sistemas — todos de alguma forma relacionados à conflitos entre normas, regras, políticas ou direção e que podem, claro, serem extrapolados para o problema descrito e estudado neste trabalho, pois, de acordo com Brown e Stallings (2017, p. 98), um administrador de segurança alimenta um banco de dados de autorização que classifica qual tipo de acesso a quais recursos é permitido para um usuário específico, ou seja, mantém um banco de dados com políticas de controle de acesso que pode, portanto, ser manejado com as técnicas de mineração de dados e aprendizagem de máquina como já o são dados em outros contextos.

Há também os trabalhos de Bui, Stoller e Li (2019), Hachana, Cuppens-Boulahia e Cuppens (2015), Kalaskar e Barkade (2018), Martin e Tao Xie (2006), Chakraborty, Sandhu e Krishnan (2019) e os de Xu e Stoller (2013) e Xu e Stoller (2014) que analisam, de formas diferentes, aspectos relacionados com a mineração de dados em políticas de controle de acesso. Entretanto, a maioria dos trabalhos citados tem a desvantagem de abordarem o conflito de políticas, em si, de forma mais periférica e secundária (até acidental), sendo que em nenhum destes trabalhos, o foco principal é a detecção de conflitos entre políticas propriamente ditas (sejam eles quais forem), concentrando-se primordialmente seja em migração/conversão de políticas de um modelo para outro; ou no problema de políticas refletirem fielmente o desejo do autor e até mesmo em outros contextos (muito específicos), como o gerenciamento de firewalls ou mesmo usando abordagens particulares demais, com o foco em regras de associação (que não se aplicam ou resolvem diretamente o problema investigado nesta dissertação).

Neste contexto e tendo em vista que:

- em grandes organizações as políticas de segurança, como as de controle de acesso, pela quantidade de objetos, modalidades, sujeitos e ações inerentes a essas instituições tendem a ter grande quantidade de informações que aumentam diariamente e constantemente nos sistemas computacionais (FUGINI; BELLETTINI, 2004; BELLETTINI; BERTINO; FERRARI, 2001; UEDA, 2012);
- que pode ocorrer, com a análise de políticas em pares (ou sem filtros de agrupamentos), conforme descrito no trabalho de Shoham e Tennenholtz (1995), um problema NP-completo que onera o custo computacional;
- que pode-se otimizar conhecimento adquirido e já existente nas organizações (os *datasets* de políticas) mediante o uso de modelos de mineração de dados e aprendizagem

de máquina, aproveitando-se da "história" temporal das políticas da organização;

Propõe-se, portanto, neste trabalho, aplicar a mineração de dados com técnicas de aprendizagem de máquina, como possibilidade de solução na detecção de conflitos entre o crescente número de políticas computacionais de uma organização buscando evitar o custo computacional elevado, aproveitando-se, pois, da "história" temporal das políticas da organização, mediante o conhecimento adquirido, "treinado" e otimizado pelos algoritmos de aprendizagem de máquina.

1.1.3 Problema de pesquisa

Para Wang et al. (2010), "a garantia dos requisitos de segurança, descritos pelas políticas de controle de acesso (ACPs), não pode ser obtida quando existem conflitos nos ACPs".

E, conforme Sun et al. (2011), podem, de fato, surgir problemas de conflito de políticas quando novas políticas de acesso são geradas podendo, assim, entrar em confronto com as políticas existentes. Como resultado dos conflitos de política, as informações privadas podem não ser bem protegidas.

A detecção automatizada destes conflitos, citados na seção 1.1 e na subseção 1.1.1, com acurácia apropriada para utilização desta detecção em tempo de execução, é, assim, o cerne do problema de pesquisa deste trabalho.

O problema investigado nesta dissertação consiste na *detecção de conflitos de forma automatizada usando técnicas de mineração de dados e aprendizagem de máquina* que apresentem acurácias suficientemente convenientes para que estes conflitos sejam detectados em tempo de execução e que, ao se inserir novas instâncias e várias políticas sejam analisadas simultaneamente isto não acarrete um custo computacional elevado.

1.1.4 Hipótese

Diante do contexto apresentado na seção anterior e também por Fugini e Bellettini (2004), Bellettini, Bertino e Ferrari (2001) e Ueda (2012) tem-se *como hipótese deste trabalho*, que o problema de detectar conflitos entre políticas *pode ser convertido e transformado* em uma tarefa de classificação da mineração de dados e que o uso de algoritmos de aprendizagem de máquina associados a técnicas de *data mining* para detectar estes conflitos configure um método que apresente precisão e acurácias pertinentes com a possibilidade de se verificar os conflitos sem onerar demasiadamente o custo computacional.

1.2 Objetivos

Nesta seção serão descritos o objetivo geral e os específicos deste trabalho.

1.2.1 Objetivo geral

O objetivo deste trabalho é propor que o problema da detecção de conflitos diretos entre políticas de controle de acesso pode ser resolvido como um problema de *data mining* (mineração de dados) solucionado pela tarefa da *classificação* além de especificar modelos de mineração de dados e aprendizagem de máquina com seus hiperparâmetros suficientemente genéricos de modo que possibilitem formas de detectar estes conflitos.

1.2.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Estabelecer a relação entre machine learning, técnicas de mineração de dados e o problema do conflito direto entre políticas;
- Determinar e comparar quais algoritmos e técnicas são mais adequados para a detecção de conflitos diretos entre políticas de controle de acesso usando as suas acurácias, precisão e recall como comparativo;
- Usar e comparar a precisão e a taxa de acertos dos principais algoritmos de classificação na resolução do problema da detecção de conflitos;
- Usar e comparar o desempenho, a precisão e taxa de acertos das principais técnicas usadas no aprendizado de máquina em simulações em lote realizando testes estatísticos para determinar o melhor algoritmo de detecção de conflitos diretos;
- Empregar *frameworks* de aprendizado de máquina como TensorFlow (KADIMISETTY, 2018), ou Torch (PASZKE et al., 2019) na construção, treinamento e teste de arquiteturas de redes neurais para a resolução do problema específico deste trabalho;
- Estabelecer a detecção de conflitos entre políticas de controle de acesso como uma classe de problemas a serem resolvidos de forma eficiente por técnicas de aprendizagem de máquina.

1.3 Solução Proposta

Diante da hipótese apresentada na subseção 1.1.4, a solução para o problema exposto neste trabalho na subseção 1.1.3 concentra-se, prioritariamente, em mostrar que

converter (ou *transformar*) a detecção de conflitos a um *problema de classificação* da mineração de dados associado a técnicas de aprendizagem de máquina, reestruturando os atributos do *dataset*, se necessário, se configura um método com acurácia satisfatória para que os conflitos entre as políticas de controle de acesso possam ser detectados automaticamente em tempo de execução.

A primeira solução proposta para conflitos diretos entre políticas é usar as técnicas e algoritmos de classificação (aprendizado supervisionado) para realizar a detecção automatizada de conflitos. Para isso, propõem-se:

- Usar e comparar as acurácias, precisões e outras métricas entre os principais algoritmos de classificação;
- Usar, inicialmente, uma rede neural (um perceptron de uma camada ou com somente uma camada oculta e apenas com *forward*) como técnica algorítmica para a detecção de conflitos e
- Construir a arquitetura de uma rede neural multicamadas (com camadas ocultas), e retropropagação (*backpropagation*), comparando-a com outro classificadores no contexto do aprendizado de máquina, como, por exemplo, o SVM ou o Random Forest, para estabelecer qual técnica de mineração de dados na detecção de conflitos em políticas é mais precisa.

Realizado os múltiplos experimentos, atestar a hipótese realizando testes estatísticos mediante os resultados apresentados que suportem a hipótese.

1.4 Método de Pesquisa

O método de pesquisa deste trabalho é descrito como:

- *quanto à natureza* é uma obra original;
- no que se refere *aos objetivos* é uma pesquisa explicativa;
- e *quanto aos procedimentos técnicos* é um *estudo experimental* sendo que a técnica e a forma geral de como a estrutura e a arquitetura de todos os experimentos foram realizados, relatando, com particularidades, detalhes e minúcias, todos os passos necessários para atingir os objetivos descritos na seção 1.2 e se o forem, de quais formas eles foram atingidos, estão pormenorizadamente detalhados no Capítulo 3 deste trabalho.

O cerne do método usado nesta dissertação é a experimentação, pois, cf. Travassos e Gurov (2002, p. 3), este método "oferece o modo sistemático, disciplinado, computável e controlado para avaliação".

O experimento é uma investigação formal, rigorosa e controlada. Em um experimento, os fatores chave são identificados e manipulados, enquanto outros fatores do contexto são mantidos sem mudança (WOHLIN et al., 2000, p. 11). Isto é realizado para investigar e encontrar dependências entre estes fatores (WAZLAWICK, 2009, p. 63) .

No caso específico deste trabalho, o método experimental, genericamente, segue a seguinte organização. (TRAVASSOS; GUROV, 2002, p. 3):

1. sugere-se um modelo;
2. desenvolve-se as técnicas quantitativas/estatísticas e qualitativas (dos algoritmos de mineração e aprendizagem de máquina);
3. aplica-se e executa-se os experimentos;
4. mede-se e analisa-se os resultados;
5. avalia-se o modelo com métricas apropriadas; e
6. repete-se o processo orientando-o à melhoria do modelo que suporte a hipótese.

Seguindo, assim, as fases de (i) definição do experimento; (ii) planejamento; (iii) execução; (iv) análise; e posterior (v) apresentação e empacotamento (organização dos dados para apresentação) mostradas em Travassos e Gurov (2002, p. 21 e 22).

Este processo se inicia, de fato, com a sugestão de um modelo, baseado em um modelo já existente na literatura e nos *frameworks* de ciência de dados, e, então, estuda-se os efeitos do processo experimental descrito acima nos dados sugeridos pelo modelo proposto. Os experimentos, portanto, são o suporte que ajudam a assentar e verificar a previsão teórica (a hipótese) deste trabalho, construindo, deste modo, uma base de conhecimento confiável sobre o tema (TRAVASSOS; GUROV, 2002, p. 3 e 4).

Para dar alicerce a essas avaliações quantitativas e estatísticas da análise dos modelos diversas métricas de verificação foram usadas no decorrer do trabalho, sendo as principais a taxa de erro, a acurácia e o *cross-validation* com 5 *folds* explicitadas na subseção 2.5.2, na subseção 2.5.3 e na subseção 2.5.3.1. Isto porque, "a medição é a parte central de um estudo experimental" (TRAVASSOS; GUROV, 2002, p. 10).

1.5 Contribuições diferenciais do Trabalho

Espera-se como contribuições diferenciais deste trabalho:

- a aplicação de mineração de dados e técnicas de aprendizagem de máquina para solucionar o problema da detecção de conflitos em tempo de execução com isso, evitando o custo computacional elevado (ocasionado pela análise das políticas em pares ou em conjuntos de amostras — *batches*);
- lidar de forma pertinente com a quantidade crescente de informações de políticas de controle de acesso que aumentam constantemente nos sistemas computacionais aproveitando-se, assim, de forma conveniente, da "história" temporal das políticas de uma organização para a construção de modelos cada vez mais confiáveis e que detectem os conflitos com precisão aprimorada com o tempo;
- apresentar uma alternativa ao gerenciamento da quantidade crescente das políticas de controle de acesso de uma organização sempre que os sujeitos, objetos e ações é alto e a população do sistema for dinâmica.

1.6 Limitações do Trabalho

Não faz parte do escopo deste trabalho:

- Delinear um modelo de política com objetivos semânticos diferenciados. Para os experimentos deste trabalho será usado um modelo de políticas semelhante ao descrito em Sarkis (2017) e em Sarkis, Silva e Braga (2016)
- Analisar comparativamente os modelos de extensão de políticas em um determinado contexto;
- Usar redes neurais convolucionais profundas na detecção dos conflitos. Prioritariamente pela limitação temporal e tamanho da base;
- Abordar a semântica em políticas;
- Por conta da base de dados (*dataset*) ser gerada de forma aleatória (já que organizações, por motivos de segurança, não disponibilizarem suas bases de políticas), ela pode não representar fielmente os modelos de políticas aplicados no mundo real, embora seja baseada em um modelo suficientemente genérico.

1.7 Organização do trabalho

Além deste capítulo, este trabalho está organizado da seguinte forma:

No Capítulo 2 apresenta-se todo o referencial teórico compreendendo uma revisão bibliográfica sobre os principais temas desta proposta de dissertação, como políticas,

detecção de conflitos, mineração de dados e aprendizagem de máquina (e seus algoritmos principais)

No Capítulo 3 são mostrados o método e os múltiplos experimentos além dos resultados obtidos. No Capítulo 4 apresenta-se as conclusões atingidas desta pesquisa e as considerações finais do trabalho.

2 Revisão Bibliográfica

Neste capítulo, alguns conceitos fundamentais da pesquisa bibliográfica realizada serão explanados com o intuito de atingir os objetivos descritos na seção 1.2 e o entendimento da solução proposta neste trabalho. Serão abordados modelos de políticas, como as de controle de acesso e conflitos entre as mesmas. Temas como mineração de dados, aprendizagem de máquina, algoritmos de classificação, hiperparâmetros, técnicas de aprendizado de máquina como as **RNA's** –*Redes Neurais Artificiais* e as **SVM** — *Support Vector Machines* e outros classificadores também serão descritos e suas definições teóricas serão discutidas. Ao final, serão descritos alguns trabalhos relacionados ao tema desta dissertação.

2.1 Segurança de dados computacionais

Para Wang et al. (2010, p. 264), "O controle de acesso e a segurança da informação estão intimamente relacionados" e as informações e os dados disponíveis sofreram um aumento frenético nas últimas décadas, fato que gerou uma crescente preocupação com o correto tratamento destes dados e, principalmente com questões de *segurança* relacionadas à *proteção* destas informações que seguem, a cada ano sendo coletadas e acumuladas em ritmo vertiginoso (ALECRIM, 2019; MACHADO, 2014; LIMA; PEREIRA, 2012; FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996; SANTOS, 2007).

Considerando, por exemplo, o ambiente corporativo, cf. Fontes (2012, p.1), a "informação é um recurso essencial para toda organização, independentemente do seu porte e do seu segmento de atuação no mercado". Neste contexto, tanto a obtenção da informação é um processo importante quanto o são as formas de armazenamento e proteção, além, claro, o fato de que a análise e interpretação destes dados e informações tornam-se atividades essenciais para a manutenção de negócios e o próprio desenvolvimento da sociedade. (SILVA; PERES; BOSCARIOLI, 2017) (MARCIANO, 2006).

A informação, como observa-se, é um recurso crítico para qualquer instituição exigindo, portanto, *a adoção de políticas de segurança* adequadas que visem a sua proteção, salvaguarda e manutenção para que este ativo, tão significativo, mantenha seu valor, sua abrangência e importância dentro do cenário das organizações, governos e usuários comuns (MARCIANO, 2006). Isto porque assim como a quantidade de dados e informações cresce rapidamente, aumentam também, a cada ano, *as ameaças* aos ativos de informação variando desde fraudes informáticas e sabotagens até vandalismo ou espionagem (CASACA; CORREIA, 2013). Nos primórdios da computação o mundo ainda não estava totalmente interconectado por redes de computadores, em especial, pela Internet que trouxe, além de

inúmeros benefícios, diversos outros problemas de segurança da informação e representou um desafio novo à proteção dos dados das organizações, governos e usuários domésticos (FONTES, 2012). Assim, para Mukkamala, Janoski e Sung (2002), observa-se que que a segurança da informação é uma questão de preocupação global séria. Pois, a complexidade, acessibilidade e capilaridade da Internet serviram para aumentar as ameças à segurança dos sistemas de informação.

Fontes (2012, p. 2) afirma que, por exemplo, informações que antes estavam persistidas em relatórios e poderiam ser protegidas fisicamente em gavetas ou armários hoje podem ficar disponíveis na Internet e acessíveis no mundo todo. Mínimas falhas ou ações criminosas podem disponibilizar informações sigilosas ou privadas ou bloquear o acesso a informações críticas para a realização do objetivos da organização, como alguns ataques cibernéticos o fazem. Portanto, para Casaca e Correia (2013), estes ativos de informação *devem ser protegidos* de ameaças através de estratégias e *políticas de segurança da informação*, feitas com base em modelos e métricas bem conhecidas e previamente definidas, permitindo às organizações melhorar seu processo de segurança da informação.

Assim, cf. Doherty e Fulford (2005), como as informações são um ativo corporativo crítico que se tornou cada vez mais vulnerável a ataques de vírus, hackers, criminosos e erros humanos, as organizações precisam priorizar a segurança de seus sistemas para garantir que seus ativos de informação mantenham sua precisão, confidencialidade e disponibilidade. Uma das muitas formas de prover essas características, confidencialidade, confiabilidade é usando políticas de controle de acesso que garantem, mediante o estabelecimento de diretrizes, especificações e normas, a proteção e disponibilidade das informações dos sistemas (BELLETTINI; BERTINO; FERRARI, 2001) e (UEDA, 2012).

2.2 Controle de acesso

Para Brown e Stallings (2017, p. 97), "podemos ver o controle de acesso como o elemento central da segurança de computadores" já que os objetivos fundamentais da segurança de computadores são proibir que usuários não autorizados tenham acesso a recursos, evitar que usuários autênticos acessem recursos de forma não autorizada e conceder acesso a usuários legítimos a recursos do sistema de forma autorizada (BROWN; STALLINGS, 2017, p. 97).

Para Knapp et al. (2009), o objetivo de uma política é fornecer orientação gerencial e suporte à segurança da informação de acordo com os requisitos de negócios, as leis e regulamentos relevantes em determinada organização ou instituição. A política de segurança em um sistema computacional garante, portanto, a proteção de suas informações. Dentre as diversas tecnologias utilizadas para assegurar essas propriedades, temos, por exemplo, o controle de acesso (SAMARATI; VIMERCATI, 2001).

O controle de acesso é o mecanismo central para atingir os requisitos de segurança em sistemas de informação, pode ser usado para obter confidencialidade, integridade e privacidade de dados, assegurando que o acesso a um sistema de informação e a seus recursos possam ser monitorados conforme um conjunto de políticas que já foram definidas (WANG et al., 2010). Dessa forma, trata-se de uma tecnologia requerida para quem faz uso de qualquer tipo de sistema, podendo basear-se ou coexistir com outros serviços de segurança (SAMARATI; VIMERCATI, 2001) .

Os níveis de proteção de um sistema dependem dos requisitos de segurança especificados no projeto do mesmo, pois em determinadas aplicações, um controle de acesso simples é satisfatório, como, por exemplo, conceder acesso a todos os recursos uma vez que o usuário está autenticado. Entretanto, em aplicações com recursos mais sensíveis, confidenciais ou com informações classificadas, como sistemas governamentais, organizações e grandes empresas com segredos industriais ou de posse de vantagens competitivas exigem um controle de acesso mais profundo e sofisticado com regras mais aprimoradas e que dependam de múltiplos fatores e alguns mecanismos de proteção contra erros e falhas (FERRAIOLO; ATLURI; GAVRILA, 2011; SILVA, 2018).

Desta forma o controle de acesso busca impedir a execução de ações que possam levar a inconsistências, falhas de segurança e que comprometam o funcionamento do sistema como um todo, afetando a confidencialidade, a integridade e a disponibilidade de serviços, recursos e informações. (SAMARATI; VIMERCATI, 2001) e (SANDHU; SAMARATI, 1996)

Kropiwiec (2005, p. 8) afirma que "para entender o que é e como funcionam os mecanismos de controle de acesso de um sistema [...], faz-se necessário classificar os elementos do sistema em três grupos." Que são, segundo este autor, o conjunto de *sujeitos*, de *objetos* e o de *ações*.

De acordo com Kropiwiec (2005, p. 8), o *conjunto de sujeitos* "englobam qualquer elemento que pode realizar ações sobre objetos, e é composto por usuários, processos e o próprio sistema". Já o *conjunto de objetos* "englobam os elementos sobre os quais podem ser realizadas ações". E, por sua vez, o *conjunto de ações* "compreende a lista de ações que podem ser realizadas sobre cada um dos objetos do sistema". Como, por exemplo, ler, escrever, apagar, abrir conexão, enviar mensagem, encerrar conexão entre muitas outras ações relativas a cada sistema em particular.

Os modelos de controle de acesso fornecem um conjunto de regras e mecanismos para o funcionamento seguro dos sistemas, sendo responsáveis pela definição de políticas específicas de controle de acesso. Para Monteiro (2017), as políticas são diretrizes de *alto nível* que determinam como os acessos são controlados e decisões de acessos são estabelecidas (VIMERCATI; SAMARATI; JAJODIA, 2005; SAMARATI; VIMERCATI, 2001; LOPES, 2012).

Embora não seja o escopo deste trabalho é importante salientar que existem vários modelos, na literatura, de controle de acesso. Entre os principais, pode-se citar, o controle de acesso discricionário (*Discretionary Acces Control* - DAC); o controle de acesso mandatório (*Mandatory Access Control* - MAC); o controle e acesso baseado em papéis (*Role-Based Access Control* - RBAC) e o controle de acesso baseado em relacionamentos (*Relationship-Based Access Control* - ReBAC).

O modelo de políticas adotado neste trabalho é baseado no proposto por Sarkis (2017) e (SARKIS; SILVA; BRAGA, 2016). Este modelo está, também, fundamentado nos propostos pelos autores Cuppens, Cuppens-Boulahia e Ghorbel (2007), além do trabalho de Elrakaiby, Cuppens e Cuppens-Boulahia (2012), bem como no artigo de Kalam et al. (2003) por dois motivos:

- Este modelo ser suficientemente genérico, conforme demonstrado no texto de Sarkar, Bali e Sharma (2017);
- Este específico modelo abarcar, de forma conveniente e coerente, características importantes dos padrões RBAC e ReBAC.

A respeito do ABAC e do RABAC indica-se o artigo de Qi, Di e Li (2018). Sobre o DAC e o MAC, sugere-se os trabalhos de Sandhu e Samarati (1996) e Vimercati, Samarati e Jajodia (2005). Sobre RBAC, recomenda-se Ferraiolo et al. (2001) e Sandhu e Samarati (1996). Sobre o ReBAC aconselha-se seguir o trabalho de Bui, Stoller e Le (2019) e Fong (2011). E sobre fundamentos gerais acerca de todos os modelos recomenda-se também o trabalho de Brown e Stallings (2017).

2.2.1 Políticas de Controle de acesso

De acordo com Brown e Stallings (2017, p. 97), "O controle de acesso implementa uma política de segurança que especifica quem ou o que [...] pode ter acesso a cada recurso específico do sistema e o tipo de acesso que é permitido em cada instância".

Uma política de controle de acesso tem como objetivo definir ou limitar o comportamento atual ou futuro de *sujeitos* e *objetos* para garantir que as suas *ações* estejam alinhadas com os objetivos da empresa de acordo com o escopo de acesso de cada sujeito ou grupo de sujeitos, com as permissões, proibições ou obrigações que estes tenham sobre os objetos e quais dados ou recursos lhe são concedidos (DUNLOP; INDULSKA; RAYMOND, 2002).

As políticas de controle de acesso convencionais, inicialmente foram chamadas de *autorizações* e tinham a seguinte forma: `{sujeito, objeto, ação}`. Estas *autorizações* especificavam quais operações os *sujeitos* podiam executar sobre os *objetos* no contexto de um sistema (VIMERCATI; SAMARATI; JAJODIA, 2005).

Com o posterior desenvolvimento dos sistemas e das políticas, estas últimas passaram a ser direcionadas, principalmente, na especificação e administração de requisitos de controle de acesso expressos na forma de *proibições*, *permissões* e, posteriormente de *obrigações* que são as composições principais na aplicação destas políticas (SARKAR; BALI; SHARMA, 2017).

Importante salientar que as políticas, normalmente, são específicas de um dado sistema, pois, cf. Silva (2018, p. 21) "políticas definidas no escopo de um sistema dificilmente teriam utilidade ou assegurariam a segurança de outro sistema, devido a diferentes características e requisitos de segurança de cada um".

2.2.2 Modelos de políticas

Os autores Moffett e Sloman (1994) afirmam que um modelo de política deve ter os seguintes atributos fundamentais: `{modalidade, sujeito, objeto e ação}`. A particularidade da política compreende estabelecer uma *autorização*, uma *permissão* ou *proibição*.

Para Moffett e Sloman (1994), o *sujeito* da política é a quem ela é orientada. O *objeto* define o conjunto de objetos no qual a política está focada. A *ação* é estabelecida como procedimentos que podem ser efetuados em *objetos* no sistema. Outros trabalhos da literatura também consideram estes atributos apresentados aqui, com as mesmas conotações, para a definição de uma política, como o trabalho de Sarkis (2017).

Outros modelos de políticas, cada um com as suas particularidades, são descritos em Lupu e Sloman (1999), também em Sloman e Lupu (2002) e Koch, Mancini e Parisi-Presicce (2002) além de Bui, Stoller e Le (2019) e Dunlop, Indulska e Raymond (2002), mas não serão utilizados neste trabalho conforme já explicitado na seção 2.2.

2.2.3 Modelo de Política utilizado

Estabelecer políticas de controle de acesso não é um processo trivial, pois eventualmente políticas complexas devem ser representadas formalmente, e.g. aquelas originadas em práticas de regras, leis e regulamentos da organização. Desta forma, a definição da política deve combinar todos estes diferentes regulamentos para ser executada e considerar ainda todas as possíveis ameaças adicionais relativas ao uso de sistemas (VIMERCATI; SAMARATI; JAJODIA, 2005).

O modelo de política utilizado neste trabalho é baseado no paradigma proposto por Sarkis (2017) e Sarkis, Silva e Braga (2016) pelos motivos explicados na seção 2.2. Este padrão é também fundamentado e influenciado nos modelos propostos pelos autores Cuppens, Cuppens-Boulahia e Ghorbel (2007), e no trabalho de Elrakaiby, Cuppens e Cuppens-Boulahia (2012), bem como no artigo de Kalam et al. (2003).

Portanto, no modelo de Sarkis (2017, p.36) e Sarkis, Silva e Braga (2016), uma política é uma tupla da forma:

$$Policy = KP \times Org \times SR \times AA \times OV \times Ac \times Dc \tag{2.1}$$

Onde **KP** retrata a categoria da política que pode ser uma proibição (F), do inglês *Forbidden*; uma permissão (P); ou uma obrigação (O). Já o termo **Org**. caracteriza o lugar ou ambiente onde a política deve ser efetuada, isto é, a organização na qual os sujeitos devem cumprir a política. O termo **SR** representa para quem, ou seja, a quais entidades, se aplica a política, este pode ser um sujeito $s \in S$ ou um papel $r \in R$, do inglês *role*, portanto, $SR = S \cup R$. Um sujeito, s, pode ser um usuário $u \in U$, este representando o conjunto de todos os Usuários, ou uma organização $org \in Org$ — esta última sendo o conjunto de organizações — representando o grupo de sujeitos que devem cumprir com a política, isto é, $S = U \cup Org$). Já o termo **AA** define uma ação $a \in A$ ou uma atividade $act \in Act$ — uma atividade é a união de muitas ações relacionadas. **OV** especifica um objeto $o \in O$ ou uma visão $v \in V$ que está sendo utilizada pela ação ou atividade — uma visão é a união de vários objetos. Por último, os termos, **Ac** é a circunstância de ativação da política e **Dc** é a condição de desativação da política. Uma condição constitui a configuração para um evento, em termos de que a política deva seguir. Neste modelo abordado foi definido Ac e Dc como datas, portanto, Ac é a data de ativação da política e Dc é a data de desativação.

Figura 1 – Modelo das políticas utilizadas no estudo

```
Policie10-> [Permitted, Administrative_Unit, PROTOCOLIZADOR3, Close, ProcessDispatch, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]
Policie11-> [Permitted, Administrative_Unit, PROTOCOLIZADOR3, Create, ProcessDispatch, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]
Policie12-> [Permitted, Administrative_Unit, PROTOCOLIZADOR3, Record, ProcessDispatch, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]

Policie25-> [Forbidden, Institution, null, Record, Process, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]
Policie26-> [Forbidden, Institution, null, Generate, Process, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]
Policie27-> [Forbidden, Institution, PROTOCOLIZADOR3, Move, ProcNURCADesp, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]
Policie28-> [Forbidden, Institution, PROTOCOLIZADOR3, Acess, ProcNURCADesp, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]
Policie29-> [Forbidden, Institution, PROTOCOLIZADOR3, Record, ProcNURCADesp, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]

Policie43-> [Obliged, Administrative_Unit, null, Open, Process, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]
Policie44-> [Obliged, Administrative_Unit, null, Acess, Process, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]
Policie45-> [Obliged, Administrative_Unit, PROTOCOLIZADOR, Open, Process, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]
Policie46-> [Obliged, Administrative_Unit, PROTOCOLIZADOR, Acess, Process, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]
Policie47-> [Obliged, Administrative_Unit, PROTOCOLIZADOR3, Open, Process, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]
Policie48-> [Obliged, Administrative_Unit, PROTOCOLIZADOR3, Acess, Process, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]
Policie49-> [Obliged, Administrative_Unit, MARY, Open, Process, Sun Aug 18 20:06:14 BRT 15, Tue Dec 31 20:06:14 BRT 15]
```

Fonte: compilação do autor

Um exemplo de política, conforme descrita acima é:

[Permitted, UFAC, John Snow, Abertura, Documentos, 03/03/19, 05/09/20]

onde:

- **'Permitted'** representa o **KP** (tipo de política);

- **'UFAC'** retrata a **Org** (organização);
- **'John Snow'** descreve o **SR** (Sujeito ou papel);
- **'Abertura'** caracteriza a **AA** (Ação ou Atividade);
- **'Documentos'** corresponde a **OV** (objeto ou visão — vários objetos);
- **'03/03/19'** representa a **Ac** (condição de ativação);
- **'05/09/20**' retrata a **Dc** (condição de desativação);

A Figura 1 exemplifica o modelo de políticas utilizadas neste estudo para a mineração de dados e aprendizagem de máquina. Na próxima seção os conceitos de mineração de dados utilizado neste trabalho serão descritos.

2.3 Mineração de Dados

Uma das características de nossa era é produção de dados em grande volume, velocidade e variedade de todas as formas, por dispositivos espalhados em toda parte. Entretanto, dados, mesmo em grande quantidade, são apenas dados. É preciso produzir informação e conhecimento para explorar as vantagens que essa massa pode trazer. O dado necessita ser, de alguma forma, analisado, tratado para que informações e conhecimento possam ser, deles, extraídos (AMARAL, 2016; FERRARI; SILVA, 2017).

Conforme Fayyad, Piatetsky-Shapiro e Smyth (1996):

> Os computadores permitiram que os humanos coletassem mais dados do que podemos digerir, é natural [,portanto,] recorrer a técnicas computacionais para nos ajudar a desenterrar padrões e estruturas significativas a partir dos numerosos volumes de dados. Por isso, [a mineração de dados] é uma tentativa de resolver um problema que a era da informação digital transformou em realidade para todos nós: sobrecarga de dados.

Para Silva, Peres e Boscarioli (2017), a *mineração de dados* pode ser definida como um processo automatizado ou semiautomatizado de explorar grandes bases de dados de forma extensiva, com o objetivo de encontrar padrões relevantes que ocorrem nos dados e que sejam significativos para embasar a absorção de informação importante, contribuindo para a geração de conhecimento.

Para Fayyad, Piatetsky-Shapiro e Smyth (1996), o termo "mineração de dados" tem sido usado por estatísticos, analistas de dados e comunidades de sistemas de informações ganhando popularidade no campo do banco de dados. Já o termo *descoberta de conhecimento em bancos de dados* [(KDD, da sigla em Inglês)] foi cunhada para enfatizar que o conhecimento é o produto final de uma descoberta baseada em dados. Está sendo utilizado nos campos de IA e aprendizado de máquina.

2.3.1 KDD - Knowledge Discovery in Databases

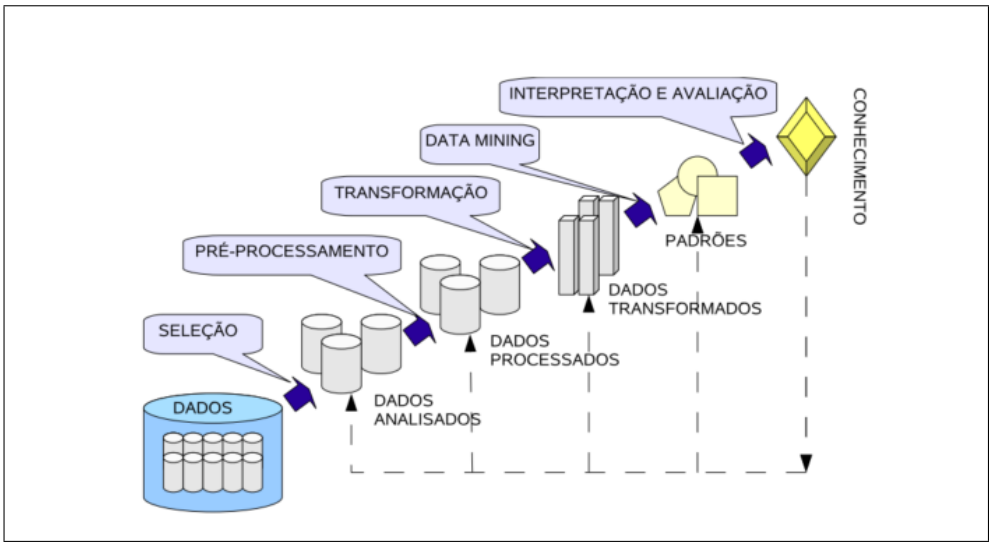
A mineração de dados é parte integrante de um processo mais amplo, conhecido como descoberta de conhecimento em bases de dados (*Knowledge Discovery in Databases*, ou *KDD*)(FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996).

Embora se use *mineração de dados* como sinônimo de KDD, a terminologia é empregada para a etapa de *descoberta* do processo de KDD, que inclui a *seleção* e *integração* das bases de dados, a *limpeza* da base, a *seleção e transformação* dos dados, a *mineração*(propriamente) e a *avaliação* dos dados (FERRARI; SILVA, 2017; SILVA; PERES; BOSCARIOLI, 2017). Assim, a mineração de dados é definida em termos de esforços para a descoberta de padrões em bases de dados. A partir destes padrões descobertos, há condições de se gerar conhecimento útil para um processo de tomada de decisão (ou a geração de conhecimento para esta tomada).

O KDD (*Knowledge Discovery in Database*), portanto, é um processo de busca de conhecimento em bancos de dados e, de modo geral, consiste de uma sequência iterativa de passos (ou **etapas**)[1]: limpeza de dados; integração dos dados; seleção, transformação e mineração dos dados; avaliação dos padrões e apresentação e assimilação do conhecimento. Este processo é iterativo e, em alguma etapa, pode-se voltar para uma anterior (SILVA; PERES; BOSCARIOLI, 2017).

A Figura 2 mostra o funcionamento iterativo do processo de KDD (*Knowledge Discovery in Database*) - Descoberta de Conhecimento em Bases de Dados.

Figura 2 – Etapas do processo de descoberta do conhecimento em bases de dados - KDD



Fonte: Fayyad, Piatetsky-Shapiro e Smyth (1996) apud Vasconcelos e Carvalho (2018, p. 7)

Neste trabalho as tarefas de seleção e transformação dos dados farão parte da etapa chamada de pré-processamento de acordo com, Silva, Peres e Boscarioli (2017) e serão

[1] O processo de KDD, segundo (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996) é composto por: *Seleção de dados; Pré-processamento; Transformação; Mineração; Análise e assimilação de resultados*

descritas com maior riqueza de detalhes no Capítulo 3.

2.3.2 Modelo de conhecimento

O termo **modelo de conhecimento** (ou hipótese) é utilizado na literatura (e neste trabalho) para fazer referência a um padrão ou conjunto de padrões descobertos (que é, enfim, o *propósito* do processo de KDD). Estes padrões são conhecimentos representados segundo as normas sintáticas de alguma linguagem formal. Estes padrões podem ser classificados em dois tipos: *preditivos* e *descritivos* (FERRARI; SILVA, 2017).

O intuito dos preditivos é resolver um problema específico de prever os resultados ou valores de um ou mais atributos, em função dos valores de outros atributos. Os descritivos (ou informativos) tem o intuito de apresentar informações interessantes e importantes sobre os dados que um especialista de domínio possa não conhecer (GOLDSCHMIDT; PASSOS, 2005).

Modelos de conhecimento compostos exclusivamente por padrões preditivos são chamados de *modelos preditivos*, enquanto que modelos descritivos são modelos de conhecimento compostos tão somente por padrões descritivos (SILVA; PERES; BOSCARIOLI, 2017).

Para Minewiskan (2020), um *modelo* de mineração é gerado a parti9r da aplicação de um algoritmo a um conjunto de dados, porém é mais que um algoritmo ou um reservatório de metadados, pois é um conjunto de dados, estatísticas e padrões que vão ser aplicados a outros dados para gerar previsões ou fazer inferências sobre relações nestes novos dados.

Neste contexto, este trabalho se concentra, portanto, em criar modelos de forma a detectar, mediante o uso de técnicas da mineração de dados (e aprendizagem de máquina) os conflitos entre as políticas de controle de acesso de um sistema. Diversos modelos e seus hiperparâmetros serão desenvolvidos e confrontados usando-se vários algoritmos e técnicas analisados com métricas específicas.

2.3.2.1 Arquitetura do modelo

Segundo Minewiskan (2020), "Um modelo de mineração obtém dados de uma estrutura de mineração e analisa esses dados usando um algoritmo de mineração de dados". Entretanto, é importante diferenciar a estrutura e o modelo de mineração. Ainda de acordo com Minewiskan (2020), a *estrutura* armazena informações que definem a fonte de dados, já um *modelo* de mineração armazena informações derivadas do processamento estatístico dos dados, como padrões encontrados em decorrência da investigação.

Assim, o modelo fica "limpo" até que os dados que foram guarnecidos pela estrutura de mineração sejam processados e avaliados. Depois de produzido o modelo contém

metadados, resultados e associações e pode, então, ser utilizado para a obtenção de conhecimento.

A arquitetura pode conter também, variáveis, *hiperparâmetros*, definições do modelo, filtros utilizados e, claro, o algoritmo utilizado na tarefa de análise dos dados (ACADEMY, 2020).

2.4 Aprendizagem de máquina

Segundo Goldschmidt e Passos (2005, p. 10), uma das etapas do processo de KDD, o de extração de padrões — a Mineração de Dados, utiliza técnicas de Aprendizado de Máquina (AM), ou em inglês *Machine Learning* - ML, para encontrar regularidades, padrões ou percepções em conjuntos de dados.

A principal diferença, segundo os autores, Goldschmidt e Passos (2005) entre Aprendizagem de Máquina e KDD reside no fato de "grande parte da literatura em AM se concentra apenas no mecanismo de descoberta de padrões e/ou conceitos, sem se preocupar com o grau de utilidade". Já em KDD, ainda segundo Goldschmidt e Passos (2005), "os padrões extraídos são avaliados para aferir sua utilidade para o usuário em relação à tomada de decisão". Ou seja, o aprendizado na Aprendizagem de Máquina é um atividade-fim enquanto o aprendizado no KDD é uma atividade-meio para a obtenção do conhecimento.

2.4.1 Definição

Aprendizado de máquina ou *machine learning* é um ramo da Inteligência Artificial que emprega técnicas e algoritmos na criação de modelos computacionais dos quais a característica principal é a capacidade de descobrir padrões em um grande volume de dados ou de melhorar o desempenho de uma determinada tarefa através da experiência (do *reforço*) (MOHRI; ROSTAMIZADEH; TALWALKAR, 2018; ALPAYDIN, 2014; SWAMYNATHAN, 2019)

Então, de acordo com Baeza-Yates e Ribeiro-Neto (2013, p. 278), "os padrões aprendidos, que podem ser bem complexos, são então usados para fazer predições relativas a dados ainda não vistos e novos".

Nas palavras de Arthur Lee Samuel Wiederhold e McCarthy (1992), considerado um dos pioneiros na área de inteligência artificial, aprendizado de máquina é "o campo de estudo que dá aos computadores a capacidade de aprender sem serem explicitamente programado" (SIMON, 2013, p. 89). Aprendizado de máquina tem sido aplicado na automatização de funções que para os humanos são executadas intuitivamente, mas que são difíceis de definir formalmente (SARKAR; BALI; SHARMA, 2017).

Conforme Baeza-Yates e Ribeiro-Neto (2013, p. 279), os algoritmos de aprendizado de máquina "são fundamentalmente dependentes de uma fase de aprendizado, a qual é usada para produzir um modelo ou uma função que codifica padrões presentes nos dados de entrada".

Então, dependendo de qual é a abordagem de aprendizado usada, os algoritmos de aprendizado de máquina podem ser, basicamente de 3 principais tipos, que são: a aprendizagem supervisionada, a aprendizagem não-supervisionada e a aprendizagem por reforço [2] (RUSSELL; NORVIG, 2013; BAEZA-YATES; RIBEIRO-NETO, 2013).

Na **aprendizagem não-supervisionada** o modelo/hipótese busca padrões na entrada, embora não seja fornecido nenhum *feedback* explícito. Portanto, na abordagem não-supervisionada não há, nos dados, uma classe, não há um rótulo prévio, ou seja, não existe a informação da saída desejada. O processo de aprendizado busca identificar regularidades entre os dados e não é necessária a divisão prévia dos dados em dados de treinamento, validação e teste. A tarefa mais comum de aprendizagem não supervisionada é o agrupamento. mas, os algoritmos de aprendizagem não-supervisionada incluem ainda, modelos de redes neurais, análise de componentes independentes e o já citado *clustering* (agrupamento) (RUSSELL; NORVIG, 2013; SILVA; PERES; BOSCARIOLI, 2017; GOLDSCHMIDT; PASSOS, 2005; AMARAL, 2016).

Na **aprendizagem supervisionada** o modelo/hipótese observa alguns exemplos de pares de entrada e saída, e aprende uma função (ou modelo) que faz o mapeamento entre a entrada e a saída. Portanto, ela compreende a abstração de um modelo a partir dos dados apresentados na forma de pares ordenados (*entrada, saída, saída desejada*).

Há, assim, uma *classe*, ou um atributo especial com o qual se pode comparar e validar o resultado. Esta categoria de aprendizagem de máquina requer uma função de aprendizado dos dados de treinamento fornecidos como entrada. Esses dados, então, são usados para aprender uma função de classificação que pode, assim, ser usada para realizar predições de classes para dados ainda não vistos ou novos (RUSSELL; NORVIG, 2013) (LUGER, 2015; BAEZA-YATES; RIBEIRO-NETO, 2013).

Na **aprendizagem por reforço**, aprende-se a partir de uma série de reforços — recompensas ou punições. Não está disponível, geralmente, na aprendizagem por reforço, para o algoritmo de aprendizado de máquina, um conjunto de dados para treinamento. O aprendizado se dá, então, pela interação com o ambiente que se deseja atuar por um determinado período com o objetivo de melhorar o desempenho de uma determinada tarefa (RUSSELL; NORVIG, 2013; AMARAL, 2016; SILVA, 2019).

[2] Há ainda os tipos de aprendizado *semissupervisionado* e a *transdução* (ou inferência transdutiva) que não serão discutidos neste trabalho

2.5 Algoritmos de classificação

A classificação é considerada uma das tarefas principais do processo de aprendizagem de máquina, sendo, inclusive, a tarefa mais comum (AMARAL, 2018; FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996).

Segundo, Amaral (2018, p. 86), "Na classificação, os dados devem possuir uma *classe* a qual queremos prever". Conforme Rocha et al. (2012), "o termo *classe* deve ser usado quando existe informação sobre quantas e quais são as partições presentes em um conjunto de dados, bem como qual exemplar pertence a qual partição".

Comumente, denomina-se *classificação* o processo pelo qual se determina uma função de mapeamento capaz de indicar a qual classe pertence algum exemplar de um domínio sob análise, baseando-se em um conjunto já classificado (SILVA; PERES; BOSCARIOLI, 2017).

Assim, de acordo com Kesavaraj e Sukumaran (2013), classificação é uma técnica de mineração de dados (aprendizado de máquina) usada para prever a associação ao grupo para instâncias de dados. É, segundo Amaral (2016) e Shazmeen, Mustafa e Baig (2013), como citado anteriormente, a tarefa mais utilizada em mineração de dados. Além de ser a mais complexa e a que possui a maior quantidade de algoritmos disponíveis, conforme descrito em Kesavaraj e Sukumaran (2013).

A classificação é uma das tarefas *preditivas* de Mineração de Dados e aprendizado de máquina. Tarefas de predição consistem na análise de um *dataset* (conjunto de dados), descritos por atributos e rótulos associados com o objetivo de descobrir um **modelo** capaz de mapear corretamente cada um dos dados a seus rótulos apropriadamente.

Esse objetivo é alcançado por meio de técnicas, normalmente, chamadas de supervisionadas. Este tipo de análise preditiva pode ser dividida em *categórica*, também chamada de *classificação* ou em *numérica*, também chamada de regressão. Cf. Silva, Peres e Boscarioli (2017) e Kesavaraj e Sukumaran (2013), também exposto em Ferrari e Silva (2017) e Goldschmidt e Passos (2005).

Formalmente, a tarefa de classificação pode ser descrita como a busca por uma função de mapeamento para um conjunto X de vetores de entrada (ou, exemplares — os dados) $\vec{x_i} \in E^d$ para um conjunto finito de rótulos C de cardinalidade c. A função F é, então, definida como $F : E^d \times W \rightarrow C$, em que d é a dimensão do espaço E, ou seja, a quantidade de coordenadas do vetor $\vec{x_i}$, e W é um espaço de parâmetros ajustáveis por meio do algoritmo de indução supervisionada (SILVA; PERES; BOSCARIOLI, 2017).

Pode ser dividida em, ao menos, duas categorias: *classificação binária* e *classificação multiclasse*. Na binária, a cardinalidade c é 2. Para o caso em que $c > 2$, o problema é considerado de múltiplas classes (SILVA; PERES; BOSCARIOLI, 2017; KESAVARAJ;

SUKUMARAN, 2013).

Os textos de Kesavaraj e Sukumaran (2013), Shazmeen, Mustafa e Baig (2013), além dos de Wolpert (1996), Kumar (2012) e Al-Radaideh e Nagi (2012) trazem mais análises, técnicas, comparações e explicações mais aprofundadas de muitos algoritmos de classificação, entre eles, árvores de decisão, k-vizinhos mais próximos, Naive Bayes e Redes Bayesianas, Redes Neurais Artificiais, Máquinas de Vetores de Suporte (SVM) entre outros.

2.5.1 Teoria da aprendizagem e algoritmos de classificação

Sobre *teoria da aprendizagem* e *algoritmos de classificação* há uma discussão em Russell e Norvig (2013) sobre qual seria, em relação às hipóteses de modelos de aprendizagem, aquela (ou aquelas) que melhor se ajuste aos dados futuros. Os autores citam a ***suposição de estacionaridade***, ou seja, que há uma distribuição de probabilidade sobre os dados que permanece estacionária ao longo do tempo. Supõe-se, portanto que cada exemplo de ponto de dados é uma variável aleatória E_j cujo valor observado $e_j = (x_j, y_j)$ é amostrado da distribuição e é independente dos exemplos anteriores.

Assim:

$$P(E_j|E_{j-1}, E_{j-2}, ...) = P(E_j), \tag{2.2}$$

e cada exemplo tem uma distribuição de probabilidade anterior idêntica:

$$P(E_j) = P(E_{j-1}) = P(E_{j-2}) = \dots \tag{2.3}$$

Estes exemplos são chamados de *independentes e identicamente distribuídos* ou **i.i.d**. Esta suposição é, segundo os autores, necessária para *tentar a previsão sobre o futuro dos dados*. Há claro, ainda em Russell e Norvig (2013), um alerta sobre o fato de ser possível a aprendizagem ocorrer caso haja pequenas alterações (lentas) na distribuição. Outro fato importante para a definição e avaliação da escolha da melhor hipótese (modelo) de um algoritmo de classificação é definir o "melhor ajuste". Em seu conhecido livro de Inteligência Artificial, os autores Russell e Norvig (2013) definem a **taxa de erro** de uma hipótese como uma métrica importante para definir o "melhor ajuste" de um modelo/hipótese.

2.5.2 Taxa de erro

A taxa de erro é, assim, a proporção de erros que o algoritmo classificador comete — a proporção de vezes que $h(x) \neq y$ para o exemplo (x, y) — sendo $h(x)$ a função que mapeia uma *hipótese/modelo* h com a *previsão/valor* conhecido y. Nem sempre, como alerta, Russell e Norvig (2013), uma hipótese/modelo h que tenha uma taxa de erro baixa no conjunto de treinamento generaliza bem e se comporta eficientemente para dados

não conhecidos. A forma de testar o algoritmo é importante. Para isso há, na literatura, algumas técnicas que são utilizadas como estratégia de treinamento, validação e teste.

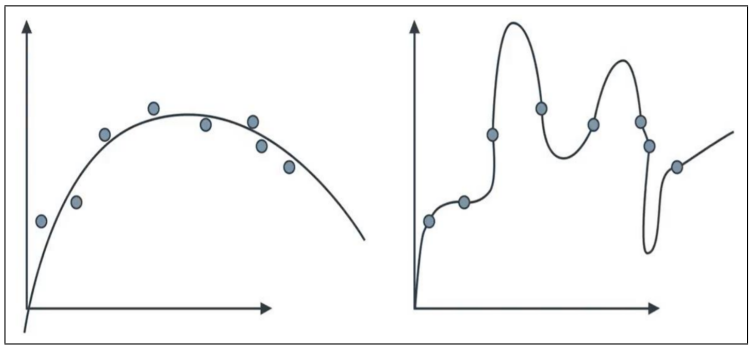
2.5.3 Estratégias de validação

Como citado por Russell e Norvig (2013) e Luger (2015) entre diversos autores e, cf. Silva, Peres e Boscarioli (2017, p. 125), a despeito da medida de avaliação que for utilizada para testar a qualidade do modelo construído, simplesmente não é apropriado apenas pelo seu desempenho em relação às instâncias dos dados que ele conhece, ou seja, os exemplares utilizados no processo de treinamento, ou *indução*. É sempre fundamental verificar como o modelo procede quando é aplicado a instâncias que ele ainda não conhece, i.e. não usados no processo de configuração e regularem dos seus parâmetros.

Essa observação, claro, é porque modelos preditivos, dependendo de como são criados, podem levar ao ***overfitting*** (sobreajuste). O fenômeno do sobreajuste ocorre, cf. Silva, Peres e Boscarioli (2017) "quando o modelo preditivo é gerado de forma a representar os exemplares usados para sua geração com uma fidelidade mais alta que o necessário". Assim, modelos sobreajustados (ou superajustados) não são capazes de realizar predições adequadas para dados novos. Busca-se, na construção de modelos preditivos, uma maior *generalização* e no caso do sobreajuste o fenômeno contrário ocorre.

A maior causa de sobreajuste é quando os dados de treinamento não representam fielmente os dados novos (ou de produção) ou por serem diferentes (caso de dados antigos, por exemplo) ou por não serem significativos (poucos dados)[3]. A Figura 3 mostra um modelo preditivo em que o lado direito representa um modelo sobreajustado e o lado esquerdo um modelo regularizado[4] para o mesmo *dataset*.

Figura 3 – Modelo sobreajustado e regularizado para o mesmo *dataset*.



Fonte: baseado no encontrado em Kasturi (2019)

Nas palavras de Amaral (2018, p. 95), "o processo de construção de um modelo de aprendizado de máquina busca, obviamente, maximizar a precisão e minimizar a taxa

[3] é possível também o sobreajuste devido a *ruídos* nos dados, pelo uso de um modelo de forma inapropriada ou de uma *classe rara* Amaral (2016, p.35)

[4] As técnicas de ***regularização*** são usadas para reduzir o overfitting. As técnicas mais utilizadas são o decaimento de pesos (weight decay) ou regularização L2 e o custo quadrático. (ACADEMY, 2020).

de erros". Para tanto, a literatura cita diversas técnicas e estratégias para a avaliação de modelos preditivos.

Autores diversos, como Silva, Peres e Boscarioli (2017), Amaral (2016), além de Grus e Nascimento (2016) e Ferrari e Silva (2017) citam, geralmente, como estratégia de treinamento, validação e teste as seguintes técnicas:

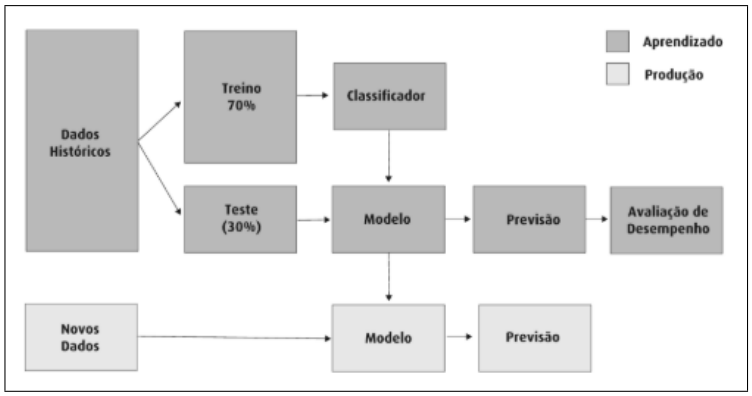
- Resubstituição;
- Holdout;
- Validação cruzada;
- Bootstrap;

Na resubstituição, segundo Silva, Peres e Boscarioli (2017), as medidas de avaliação dos classificadores são aplicadas no próprio conjunto de dados usados para indução do modelo. Essa técnica, embora tenha alguns vantagens discutidas em Ferrari e Silva (2017) e Silva, Peres e Boscarioli (2017), pode levar ao, já citado, sobreajuste (*overfitting*) e é discutido em Grus e Nascimento (2016), também em Amaral (2016) e Russell e Norvig (2013). Como já afirmado anteriormente, o sobreajuste é quando se produz um modelo de bom desempenho com os dados de treinamento, mas que não lida bem com novos dados.

Na técnica de **Holdout**, pressupõem-se uma divisão, ou criação de dois subconjuntos de dados distintos, a partir do conjunto de dados disponível pra uso na indução do modelo/hipótese. Um desses subconjuntos será usado para treinamento (indução) do modelo de previsão e o segundo, para teste após o término do treinamento e, consequentemente, na aplicação das medidas de avaliação do modelo/hipótese (SILVA; PERES; BOSCARIOLI, 2017).

A Figura 4 mostra o funcionamento da técnica de holdout de forma mais detalhada.

Figura 4 – Funcionamento da técnica holdout.



Fonte:(AMARAL, 2016)

Na estratégia de **validação cruzada**, todos os dados farão parte, em algum momento, do conjunto de dados usado no teste do modelo/hipótese. A ideia é que cada exemplo sirva duplamente — como dados de treinamento e dados de teste. Primeiro divide-se o conjunto em k subconjuntos iguais. Em seguida realiza-se k rodadas de aprendizagem; em cada iteração $\frac{1}{k}$ dos dados é retido como conjunto de teste e os exemplos restantes são usados como treinamento.

Valores populares de k são 5 e 10 — o suficiente para uma estimativa estatisticamente provável que seja precisa a um custo 5-10 vezes maior no tempo de computação. Há também o extremo do $k = n$, também conhecido como **validação cruzada com omissão de um**. O método de validação cruzada permite que o modelo/hipótese seja avaliado uma série de vezes, cada série sendo conhecida como partição (ou *fold*). Ao final, a avaliação pode ser realizada aplicando medidas estatísticas como média, desvio-padrão e intervalo de confiança ao conjunto de k avaliações obtidas ou somando-se os desempenhos obtidos pelos k modelos gerados e dividindo essa soma pelo número de exemplares original (RUSSELL; NORVIG, 2013; SILVA; PERES; BOSCARIOLI, 2017; FERRARI; SILVA, 2017; AMARAL, 2016).

A Figura 5 mostra um exemplo didático de como funciona a validação cruzada. Os dados que não fazem parte do conjunto de teste em cada rodada são utilizados no treinamento.

Figura 5 – Funcionamento da técnica *cross validation*



Fonte: Gufosowa - Distribuída sob Creative Commons em: https://bit.ly/3ziC8x4

Já a técnica de ***Bootstrap*** funciona de forma parecida à estratégia *holdout*. Ela também usa dois conjuntos, um de treinamento e outro para teste, porém durante o processo de formação dos subconjuntos, exemplares que já foram sorteados podem novamente serem contemplados, com probabilidade igual. É uma estratégia que permite, portanto, a reposição.

Neste trabalho, todos os algoritmos de classificação usados foram testados usando as técnicas de resubstituição, *holdout* (com taxas de 70-30, 75-25 e 60-40), além de *cross-validation* com 3, 5 e 10 folds.

Como explicado em Wolpert e Macready (1995) e Wolpert (1996) não existe um algoritmo de aprendizado superior a todos os demais quando considerados todos os problemas de classificação possíveis (teorema **NFL**, ou *No Free Lunch*), portanto, variações foram executadas nos experimentos em todas técnicas avaliadas, alterando-se os padrões para chegar a métricas e medidas de avaliação mais eficientes.

2.5.3.1 Medidas de avaliação

Para Castro e Braga (2011), geralmente a métrica usada na avaliação e seleção de modelos de classificação é a acurácia (ou taxa de erro) estimada em relação a um conjunto de teste. Essa metodologia é justificada pela formulação padrão do problema do aprendizado supervisionado que visa a minimização da probabilidade do erro global.

Há, porém, conforme Silva, Peres e Boscarioli (2017), Amaral (2016), Kesavaraj e Sukumaran (2013) e Kumar (2012) diversas medidas usadas na avaliação de classificadores. A que será usada neste trabalho (com o objetivo de minimizar a probabilidade do erro global) é a, já citada, acurácia ou taxa de classificações corretas.

A métrica acurácia é dada, portanto, por:

$$\text{Acurácia} = |y - f(\aleph) = 0|, \tag{2.4}$$

em que $|equação|$ representa a contagem de vezes em que *equação* é verdadeiro, f é o modelo preditivo, $\aleph$ é o subconjunto de dados sob o qual o modelo está sendo avaliado, $f(equação)$ é a classificação fornecida pelo modelo preditivo para cada um dos exemplares (dos dados), e y é a classe esperada como resposta cf. Silva, Peres e Boscarioli (2017, p. 129).

A acurácia de um classificador também pode ser descrita em termos do **erro de generalização** ξ_g, e uma função de perda binária e, portanto, ser interpretada como a probabilidade de ocorrer uma classificação correta. Dessa forma:

$$\text{Acurácia}_g = 1 - \xi_g \tag{2.5}$$

Ou seja, a acurácia é, basicamente o número de acertos (positivos) divido pelo número total de exemplos. Será a métrica mais usada para avaliar os classificadores neste trabalho.

Há, entretanto, em modelos preditivos que trabalham com dados numéricos (como as redes neurais e o SVM), pode-se utilizar uma função de perda contínua, capaz de medir o erro entre a resposta obtida pelo modelo preditivo e a resposta aguardada (SILVA; PERES; BOSCARIOLI, 2017; ACADEMY, 2020).

A função de perda também é conhecida, popularmente, como função de custo. Basicamente ela é a diferença entre os valores de rótulo de treinamento e a previsão feita pelo modelo. Os parâmetros do modelo devem, assim, serem estimados em relação à minimização da função de perda. Classificadores diferentes podem ser configurados com funções de perda distintas (REED; MARKSII, 1999; MINEWISKAN, 2020; SILVA, 2016; SILVA; PERES; BOSCARIOLI, 2017; GOLDSCHMIDT; PASSOS, 2005; GÉRON, 2019; ACADEMY, 2020).

Para Reed e MarksII (1999), A função de custo reduz todos os aspectos bons e ruins de um sistema complexo a um único número, um valor escalar, o que permite ranquear e comparar as soluções candidatas.

Algumas funções de perda (*loss functions*), comuns, neste contexto são, de acordo com Silva, Peres e Boscarioli (2017):

$$Erro\ absoluto = \sum_{<\vec{x}_i, y_i> \in \aleph} |y_i - f(\vec{x}_i)| \tag{2.6}$$

$$Erro\ quadrático = \sum_{<\vec{x}_i, y_i> \in \aleph} (y_i - f(\vec{x}_i))^2 \tag{2.7}$$

$$Erro\ absoluto\ médio = \sum_{<\vec{x}_i, y_i> \in \aleph} |y_i - f(\vec{x}_i)|/m \tag{2.8}$$

$$Erro\ médio\ quadrático = \sum_{<\vec{x}_i, y_i> \in \aleph} (y_i - f(\vec{x}_i))^2/m \tag{2.9}$$

Onde m é a quantidade de instâncias existentes em $\aleph$. Neste trabalho, as principais usadas na avaliação de alguns modelos são o erro qudrático e o erro quadrático médio.

Silva, Peres e Boscarioli (2017), Amaral (2018) além de Deng et al. (2016) e Ruuska et al. (2018) afirmam que é importante analisar o tipo de erro que o modelo está cometendo e, principalmente, no caso de classificadores binários, é possível observar que, mesmo com uma acurácia alta o classificador pode não estar respondendo de forma adequada.

Para realizar esse tipo de análise, os autores supramencionados no princípio deste parágrafo, citam que uma ferramenta adequada é a ***matriz de confusão***. Geralmente, uma matriz de confusão tem dimensões $C \times C$, em que C é o número de classes presentes no problema de classificação que está sendo avaliado. As linhas dessa matriz são indexadas seguindo as "classes esperadas" (y), e as colunas seguindo as "classes preditas" ($f(x) ou \hat{y}$).

Cada célula é um contador que é incrementado a depender do resultado da comparação de $f(x)$ e y. As respostas corretas do modelo geram os valores que entram na diagonal principal da matriz (SILVA; PERES; BOSCARIOLI, 2017, p. 130).

A Figura 6 demonstra o procedimento de preenchimento de uma matriz de confusão conforme a resolução de um problema hipotético de classificação.

Figura 6 – Preenchimento de uma matriz de confusão

Problema sob resolução	
f(x)	y
1	1
1	1
1	1
2	1
2	2
2	2
2	2
3	1
3	2
3	3

Classe Esperada (y)	Classe Predita (f(x))		
	Classe 1	Classe 2	Classe 3
Classe 1	3	1	1
Classe 2		3	1
Classe 3			1

Fonte: Silva, Peres e Boscarioli (2017, p. 130)

Matrizes de confusão são, cf. Ruuska et al. (2018) e Silva, Peres e Boscarioli (2017), particularmente úteis para avaliação de classificadores binários. Neste caso, os valores são atribuídos conforme ilustrado na Figura 6 (neste caso, as duas classes estão definidas como "classe positiva" e "classe negativa"). (HOSSIN; M.N, 2015).

Figura 7 – Matriz de confusão para o caso de um classificador binário

Classe Esperada (y)		***Classe Predita(f(x))***	
		positivo	***negativo***
	positivo	*Verdadeiros Positivos (VP)*	*Falsos Negativos (FN)*
	negativo	*Falsos Positivos (FP)*	*Verdadeiros Negativos (VN)*

Fonte: Silva, Peres e Boscarioli (2017, p. 131)

Cada célula, neste caso, possui o seguinte significado:

- Verdadeiro Positivo (VP): classificação correta na classe positiva. A instância pertence a classe positiva e o modelo classificou na classe positiva.
- Falso Positivo (FP): classificação incorreta na classe negativa. A instância pertence à classe negativa, mas o classificador a classificou como pertencente à classe positiva.
- Verdadeiro Negativos (VN): classificação correta na classe negativa. A instância pertence a classe negativa e o modelo classificou na classe negativa.

- Falso Negativo (FN): classificação incorreta na classe negativa. classificação incorreta na classe negativa. A instância pertence à classe positiva, mas o classificador a classificou como pertencente à classe negativa.

A proporção das amostras previstas como positivas é chamada **taxa de verdadeiros positivos** (**TPR**, ***True Positive Rate***). Então, a TPR (True positive Rate) é: (KLOSTERMAN, 2020) (HOSSIN; M.N, 2015).

$$TPR = \frac{TP}{P} \tag{2.10}$$

Sendo que **P** é o número de amostras da classe positiva. O inverso da taxa de verdadeiros positivos é a ***taxa de falsos negativos*** (**FNR**, ***False Negative Rate***) que é calculada da seguinte forma: (KLOSTERMAN, 2020; HOSSIN; M.N, 2015).

$$FNR = \frac{FN}{P} \tag{2.11}$$

Da mesma forma que a TPR e a FNR, há a ***taxa de verdadeiros negativos*** (**TNR**, ***True Negative Rate***) e a ***taxa de falsos positivos*** (**FPR**, ***False Positive Rate***). Na Equação 2.12 e na Equação 2.13, considera-se que N é o número de amostras negativas. (KLOSTERMAN, 2020) (HOSSIN; M.N, 2015).

$$TNR = \frac{TN}{N} \tag{2.12}$$

$$FPR = \frac{FP}{N} \tag{2.13}$$

Outra métrica usada neste trabalho é a **precisão** que é definida como a taxa entre os verdadeiros positivos (**TP**, ***True Positive***) e a soma de verdadeiros positivos com os falsos positivos (**FP**, ***False Positive***). A precisão é usada para medir os padrões positivos que são previstos corretamente a partir do total de padrões previstos em uma classe positiva. Ela representa, portanto, quantos daquelas instâncias que foram classificados como certas são efetivamente corretos. Mede a proporção de previsões positivas que estão corretas entre todas as previsões positivas. É definida pela Equação 2.14. (HOSSIN; M.N, 2015).

$$\textbf{Precisão (P)} = \frac{TP}{TP + FP} \tag{2.14}$$

Assim como a precisão, este trabalho usa o **Recall (R)**, também chamado de "sensibilidade" que é a proporção de verdadeiros positivos, ou seja, a capacidade do modelo em predizer corretamente a condição para casos que realmente a têm. Representa a fração

de padrões positivos que são classificados corretamente. É indicado na Equação 2.15. (HOSSIN; M.N, 2015).

$$\textbf{Recall (R)} = \frac{TP}{TP + FN} \tag{2.15}$$

Os algoritmos que serão descritos nas próximas seções foram escolhidos neste trabalho porque foram descritos por Wu et al. (2007) como alguns dos principais na área da Mineração de Dados e são reconhecidos por serem utilizados em diversos outros trabalhos.

2.6 Naïve Bayes

Os algoritmo Naïve Bayes é um classificador probabilístico baseado na aplicação do Teorema de Bayes[5]. É um dos mais utilizados em tarefas de classificação por apresentar bom desempenho em problemas de classificação tanto com dados categóricos quanto com dados numéricos. Geralmente é o primeiro classificador utilizado nos testes de classificação. (SILVA; PERES; BOSCARIOLI, 2017; AMARAL, 2018; GOLDSCHMIDT; PASSOS, 2005; BRUCE; BRUCE, 2019).

Recebe o nome de "ingênuo" (naïve) porque ele considera como hipótese que os atributos previsores são *estaticamente independentes* entre si, o que, em muitos casos práticos, não ocorre. Entretanto, a prática mostra que o classificador é efetivo mesmo em casos em que os atributos previsores não são estaticamente independentes. (SILVA; PERES; BOSCARIOLI, 2017; GOLDSCHMIDT; PASSOS, 2005; BRUCE; BRUCE, 2019).

Portanto, o Naïve Bayes usa a probabilidade de observação dos valores previsores, dado um resultado, para estimar a probabilidade de observar o resultado $Y = i$, dado um conjunto de valores preditores. Usa a probabilidade posterior que é a probabilidade de um resultado após a informação preditora ter sido incorporada.

A classificação usando o algoritmo Naïve Bayes funciona da seguinte forma (usando todo o conjunto de dados para prever os resultados) (BRUCE; BRUCE, 2019):

1. Para uma resposta bynária $Y = i$ (i = 0 ou 1), é estimada as probabilidades condicionais individuais para cada preditor $P(X_j|Y) = i$; estas são as probabilidades de o valor previsor estar no registro quando se observa $Y = i$. Essa probabilidade é estimada pela proporção de valores X_j entre os registros $Y = i$ no conjunto de treinamento;

[5] Criado por Thomas Bayes (1701-1761), usa a probabilidade condicional que é a probabilidade de se observar algum evento (por exemplo $X = i$) dado algum outro evento (por exemplo, $Y = i$), escrito como $P(X_i|Y_i)$ (BRUCE; BRUCE, 2019)

2. Essas probabilidades são multiplicadas uma pela outra, e então pela proporção de registros pertencentes a $Y = i$;

3. Os passos 1 e 2 são repetidos para todas as classes;

4. É estimado uma probabilidade para o resultado i assumindo o valor calculado no passo 2 para cada classe i e dividindo-o pela soma de tais valores para todas as classes;

5. É atribuído o registro à classe com a maior probabilidade para esse conjunto de valores preditores.

O algoritmo pode, também, ser escrito como uma equação para a probabilidade de se observar um resultado $Y = i$, dado um conjunto de valores previsores $X_1, \cdots, X_p$:

$$P(X_1, X_2, \cdots, X_p) \tag{2.16}$$

O valor da Equação 2.16 é um fator de escalonamento para a garantia de que a probabilidade esteja entre 0 e 1 e não dependa de:

$$\begin{aligned} P(X_1, X_2, \ldots, X_p) = P(Y &= 0)(P(X_1|Y=0)P(x_2|Y=0)\ldots P(X_p|Y=0)) + P(Y=1) \\ (P(X_1|Y=1)P(X_2|Y=1) &\ \ldots\ P(X_p|Y=1)) \end{aligned} \tag{2.17}$$

Segundo Bruce e Bruce (2019), o Naïve Bayes "é conhecido por produzir estimativas *enviesadas*.No entanto, no que o objetivo seja *ordenar* os registros conforme a probabilidade de $Y = 1$, as estimativas de probabilidade não enviesdas não são necessária" e o classificador produz bons resultados.

2.7 K-NN — K-vizinhos mais próximos

O algoritmo KNN, ou ***k-vizinhos mais próximos*** é um método de classificação supervisionada que serve para estimar a função de densidade $F(x/C_j)$ dos atributos previsores x para cada classe C_j. Geralmente é um dos primeiros classificadores geométricos utilizados. É um método estatístico não-paramétrico[6]. É usado para classificação e regressão. Ele estima o valor da função de densidade de probabilidade diretamente da probabilidade *a posteriori* de que um elemento x pertença a uma classe C_j a partir da informação proporcionada pelo conjunto de treinamento. No processo de aprendizagem, nenhuma

[6] o termo estatística não-paramétrica designa-se a estatísticas que não possuem dados ou população com estruturas ou parâmetros característicos ou é definido como uma função de uma amostra que não tem dependência de parâmetros (BRUCE; BRUCE, 2019)

suposição é feita sobre a distribuição das variáveis previsoras (por isso ele é *não-paramétrico*). (SILVA; PERES; BOSCARIOLI, 2017; GOLDSCHMIDT; PASSOS, 2005; WU et al., 2007; BRUCE; BRUCE, 2019).

É um estilo de processamento conhecido como "avaliação preguiçosa" (*lazy evaluation*) já que não existe um trabalho prévio de análise e construção de um modelo de classificação (pois o mesmo é baseado nas instâncias existentes — o que pode, dispender muito tempo dependendo do tamanho do *dataset* (conjunto de dados) disponível para o treinamento (SILVA; PERES; BOSCARIOLI, 2017). Segundo (GOLDSCHMIDT; PASSOS, 2005), "o K-NN considera que os registros do conjunto de dados correspondem a pontos no R^n, em que cada atributo corresponde a uma dimensão deste espaço".

Ainda cf. Goldschmidt e Passos (2005, p. 117 e 118), no método K-NN quando um novo registro deve ser classificado ele é comparado com todos os registros do conjunto de treinamento para identificar os k vizinhos mais próximos *i.e.*, mais semelhantes, de acordo com a métrica escolhida. A classe desta nova instância é determinada, portanto, pela inspeção das classes dos vizinhos mais próximos conforme a métrica selecionada.

A reposta do método é a classe mais frequente entre estes vizinhos mais próximos. É, desta forma, um aprendizado baseado em instâncias (*instance-based learning*) já que a classificação de um exemplar cuja classe é desconhecida é realizada a partir da comparação desse exemplar com aqueles que possuem uma classe já conhecida. (SILVA; PERES; BOSCARIOLI, 2017, p. 80). A padronização (mostrada em Equação 3.1) pode melhorar consideravelmente a exatidão e precisão do algoritmo. Os exemplos de treinamento são vetores em um espaço de característica multidimensional, cada exemplo é descrito em termos de p atributos considerando q classes para classificação. Os valores de atributo para a i-ésimo instância (onde $1 \leq i \leq n$) são representados pelo vetor p-dimensional mostra na Equação 2.18 (BRUCE; BRUCE, 2019; ALPAYDIN, 2014; KESAVARAJ; SUKUMARAN, 2013).

$$x_i = (x_{1i}, x_{2i}, ..., x_{pi}) \in X \tag{2.18}$$

Embora o processo possa ser custoso, dependendo do tamanho do conjunto de dados de treinamento, a lógica que o implementa é simples: (SILVA; PERES; BOSCARIOLI, 2017) (BRUCE; BRUCE, 2019) (GOLDSCHMIDT; PASSOS, 2005). Considerando um conjunto de dados de um problema de classificação e um novo registro a ser classificado. Considere também que foi definido um valor para a quantidade de vizinhos a ser examinada, ou seja, o valor do parâmetro k. Sendo assim, o algoritmo k-NN segue os passos subsequentes: 1) Cálculo da distância da nova instância (do novo registro) para cada uma das instâncias existentes no conjunto de dados; 2) Identificação daquelas k instâncias do conjunto de dados que apresentaram a menor distância em relação ao novo registro (ou seja, os mais

similares); 3) Verificação da classe mais frequente entre os k registros identificados no passo anterior; 4) Atribuir à nova instância esta classe.

Há, na literatura, várias formas de medir distâncias entre duas instâncias sendo a escolha de qual usar variando de acordo com o problema. As principais métricas de distância são: Euclidiana; Minkowsky; Chebishev e Manhattan (SILVA; PERES; BOSCARIOLI, 2017; ALPAYDIN, 2014; KESAVARAJ; SUKUMARAN, 2013). A distância Euclidiana é dada por:

$$D_E(p,q) = \sqrt{(p_1 - q_1)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{2.19}$$

É a mais utilizada por representar a distância física entre pontos em um espaço d-dimensional. A de Minkowski é dada por:

$$D_M(p,q) = \Big(\sum_{i=1}^{n} |p_i - q_i|^r \Big)^{\frac{1}{r}} \tag{2.20}$$

A distância de Minkowski é uma métrica para o espaço Euclidiano que serve de generalização para outras distâncias, como a própria Euclidiana e a Manhattan. A distância de Chebyshev é dada por:

$$D_C(p,q) = max_i(|p_i, q_i|) \tag{2.21}$$

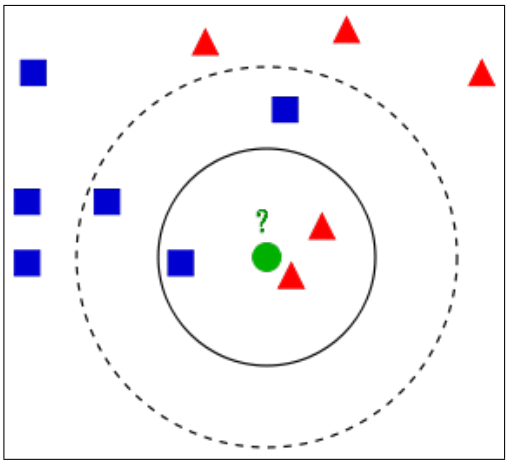
E a de Manhattan é dada por:

$$D_M(p,q) = \sum_{i}^{n} \Big| p_i - q_i \Big| \tag{2.22}$$

Nestes casos, $p = (p_1, \ldots, p_n)$ e $q = (q_1, \ldots, q_n)$ representam dois pontos n-dimensionais e na Equação 2.20, r é uma constante. No exemplo da Figura 8, a instância a ser classificada é o círculo verde. Para $k = 3$, o círculo verde seria classificado com a classe do triângulo, pois há apenas um quadrado e 2 triângulos, dentro do círculo que os contém. Se k = 5 será classificado com a classe quadrada, pois há 2 triângulos e 3 quadrados, dentro do círculo externo. Neste exemplo, as distâncias mostradas na Equação 2.19, na Equação 2.21, na Equação 2.20 e na Equação 2.22 seriam calculados entre o círculo verde e todas as outras figuras (instâncias). Este exemplo é 2D, logo, cada ponto teria seu valor em x e em y. Para dimensões maiores a abordagem é a mesma, contudo, a visualização das amostras no espaço é um pouco mais complicada. (SILVA; PERES; BOSCARIOLI, 2017; GRUS; NASCIMENTO, 2016; GOLDSCHMIDT; PASSOS, 2005).

A escolha de k depende dos dados. Geralmente, grandes valores de k reduzem o efeito do ruído na classificação, mas criam limites entre classes semelhantes. Um bom k pode ser selecionado através de otimização. O caso especial em que se prevê que a classe seja a classe mais próxima do exemplo de treinamento (quando $k = 1$) é denominado Algoritmo do Vizinho Mais Próximo. O K-NN foi utilizado neste trabalho para determinar

Figura 8 – Exemplo do algoritmo k-NN



Fonte: distribuído sob GPL no endereço: https://bit.ly/3oKQLE1

se, de fato, o prolema é linearmente separável. Por ser um método *lazy* ele, de fato, não se ajusta à hipótese desta dissertação sendo utilizado apenas para demonstrar empiricamente a linearidade dos dados e por ser o método geométrico de instâncias mais utilizado. Na próxima seção serão descritos alguns conceitos envolvendo as RNA's, Redes Neurais Artificiais.

2.8 Redes Neurais Artificiais - RNA

As redes neurais instituem um campo da ciência da computação, parte da área da inteligência artificial, que busca efetivar modelos matemáticos que se assemelhem às redes neurais biológicas. Elas apresentam capacidade de adaptar seus parâmetros como resultado da interação com o meio externo (FERNEDA, 2006; RUSSELL; NORVIG, 2013).

2.8.1 Definição

Para Santos (2013, p. 41) uma rede neural artificial contitui-se de um sistema composto por neurônios artificiais dispostos em camadas e conectados através de pesos sinápticos produzindo um sistema que simula o funcionamento do cérebro humano. É uma técnica computacional que representa um modelo inspirado na estrutura neural de organismos inteligentes e que adquire conhecimento através da experiência. De acordo com Lima, Pinheiro e Santos (2016, p. 47), "redes neurais podem ser caracterizadas como modelos computacionais com capacidades de adaptar, aprender, generalizar, agrupar ou organizar dados".

Inicialmente, portanto, se desenvolveram como uma estratégia de simular os processos mentais humanos, como reconhecimento de imagens e sons, e após, como instrumento tecnológico e eficiente para muitas tarefas (JIN; CHEU; SRINIVASAN, 2002).

Para Obaidat e Macchairolo (1994), as redes neurais artificiais podem ser usadas efetivamente para prover soluções para um amplo espectro de aplicações, incluindo mapeamento de padrões e classificação, análise e codificação de imagens, processamento de sinais, otimização, manipulação de grafos, reconhecimento de caracteres, reconhecimento automático de alvo, fusão de dados, processamento de conhecimento, controle de qualidade, mercado de ações, processamento de hipotecas, triagem de créditos para empréstimos entre muitos outros problemas.

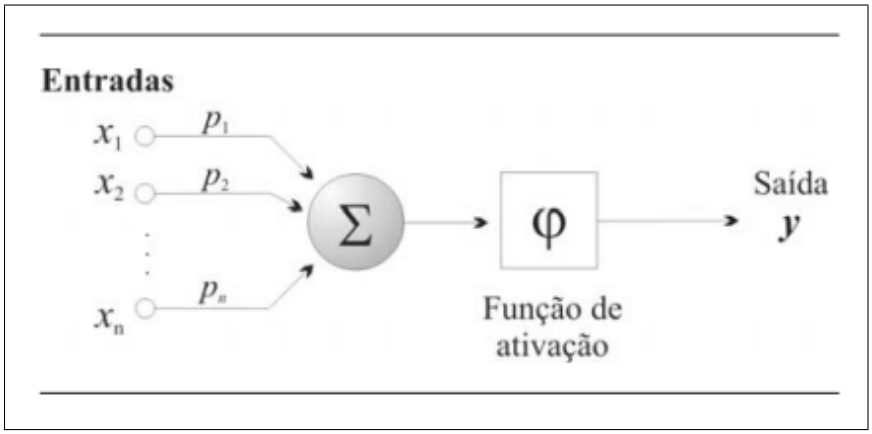
2.8.2 Modelo de neurônio artificial

Desde a década de 1940 com o trabalho de Mcculloch e Pitts (1943) que se busca um modelo computacional que simule o cérebro humano e suas conexões. O interesse pela pesquisa nesta área cresceu e se desenvolveu durante os anos 50 e 60. É dessa época que Rosenblatt (1958) sugeriu um método de aprendizagem para as redes neurais artificiais chamado *perceptron*.

Até o final da década de 1960 muitos trabalhos foram feitos usando o percepton como modelo, mas ao final desta década, Minsky e Papert (1969) apresentaram significativas limitações do perceptron.

A pesquisa diminui consideravelmente nos anos seguintes (o chamado inverno da IA), porém durante os anos 80, a excitação ressurge mediante os avanços metodológicos importantes e, também, ao aumento dos recursos computacionais disponíveis. O modelo de neurônio artificial da Figura 9 é uma simplificação do apresentado por Haykin (2001, p. 36).

Figura 9 – Modelo matemático de um neurônio



Fonte: Haykin (2001, p. 36)

O modelo da Figura 9 é composto por três elementos: um conjunto de n conexões de entrada $(x_1, x_2, \ldots, x_n)$, caracterizadas por pesos $(p_1, p_2, \ldots, p_n)$; um somador $(\sum)$ para acumular os sinais de entrada e uma *função de ativação* (φ) que, no caso específico do

neurônio apresentado por McCullock-Pitts em Mcculloch e Pitts (1943) é uma função de limiar (FERNEDA, 2006; LIMA; PINHEIRO; SANTOS, 2016).

O comportamento das conexões entre os neurônios é simulado através de seus pesos $(p_1, p_2, ..., p_n)$. Os valores podem ser positivos ou negativos (dependendo se a conexão é *inibitiva* ou *excitativa*. O efeito de um sinal proveniente de um neurônio é determinado pela multiplicação do valor do sinal recebido pelo peso da conexão correspondente $(x_i \times p_i)$.

Então é efetuada a soma dos valores $x_i \times p_i$ de todas as conexões e o valor resultante é enviado para a função de ativação que define a saída (y) do neurônio. cf. Russell e Norvig (2013) e Mcculloch e Pitts (1943), além de Minsky e Papert (1969), Ferneda (2006) e Haykin (2001). Matematicamente, a saída y_k do neurônio mostrado na Figura 9 é dada pela expressão abaixo:

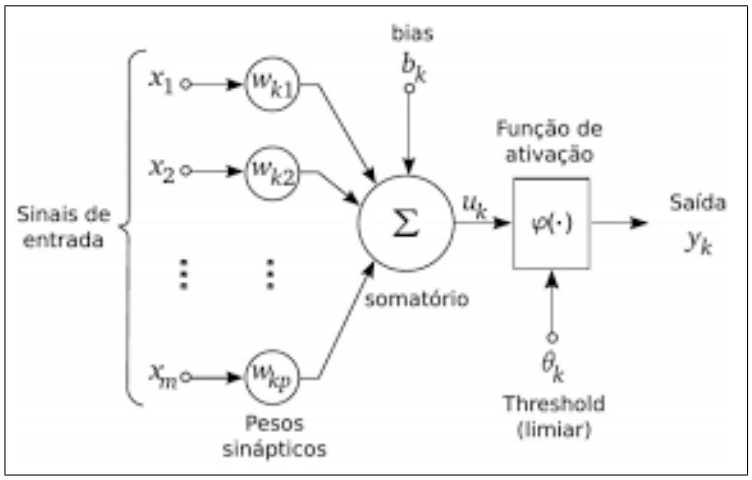
$$Y_k = \varphi(u_k) \qquad \text{onde} \qquad u_k = \sum_{i=1}^{n} P_{ki} X_i \tag{2.23}$$

A função de ativação proposta inicialmente por McCullock e Pitts em seu trabalho seminal, Mcculloch e Pitts (1943) é uma função de limiar:

$$y_k = \varphi(u_k) = \begin{cases} 1, & \text{se } u_k > 0 \\ 0, & \text{se } u_k < 0 \end{cases} \tag{2.24}$$

Uma alteração importante neste modelo foi a introdução de um parâmetro polarizador (*bias* ou *offset*) b_k, conforme a Figura 10 cujo objetivo é deslocar o valor da informação referente à entrada líquida u_k, de forma a verter a função de ativação no eixo correspondente ao valor de u_k. Assim a saída $y_k = \varphi(u_k)$.

Figura 10 – Adição de um *offset* (*bias*) no modelo do neurônio



Fonte: Lima, Pinheiro e Santos (2016, p. 58)

Dessa forma, o polarizador (bias) pode ser tratado como mais um peso da rede. Basta considerar um nova entrada do tipo $x_0 = 1$, com um peso associado $w_0 = b_k$, assim

a representação fica sendo, considerando os pesos como w, de acordo com a Figura 10:

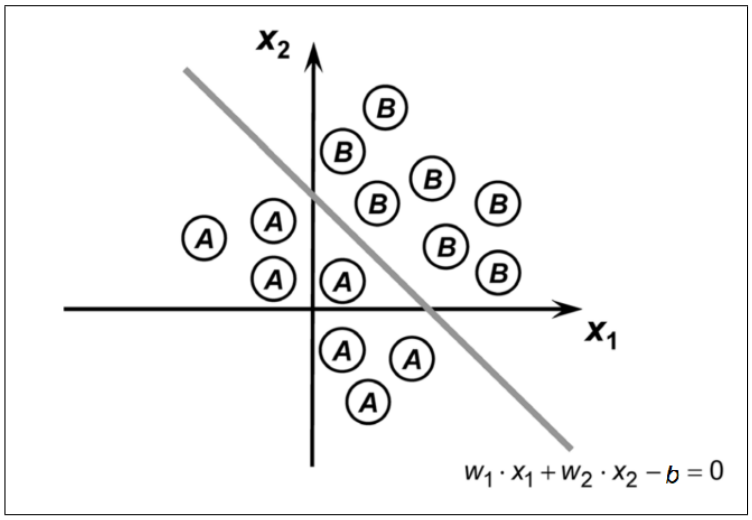
$$u_k = w_0 + w_1 x_1 + w_2 x_2; \qquad y_k = \varphi(u_k) \tag{2.25}$$

Assim, esse neurônio pode ser empregado, segundo, Lima, Pinheiro e Santos (2016, p. 58), para "separar classes distintas de padrões de entradas para aplicações de classificações de padrões". E prossegue: "se a entrada líquida for maior que o limiar, o padrão dessa entrada pertence à classe 1, caso contrário, pertence à classe 0". Analisando matematicamente o modelo *perceptron*, se percebe que ele pode ser considerado um típico caso de discriminador linear.

A fronteira de decisão para um *perceptron* como o da Figura 10, dada duas entradas e a função de ativação mostrada na Equação 2.24 será, então, uma reta cuja equação é definida por:

$$w_1 x_1 + w_2 x_2 - b = 0 \tag{2.26}$$

Figura 11 – Fronteira de separação (perceptron com duas entradas)

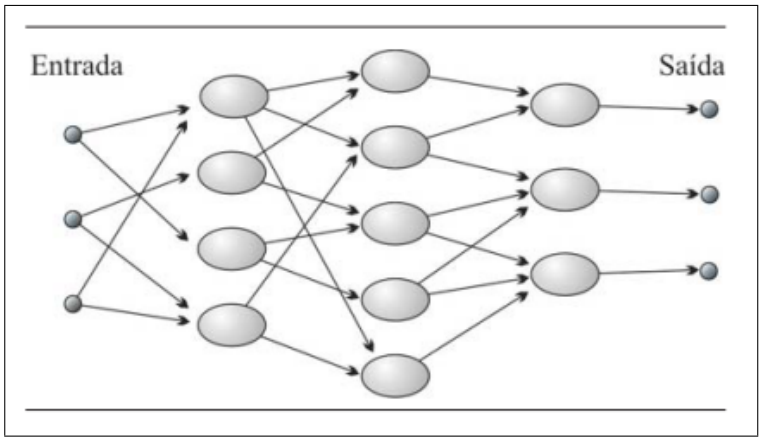


Fonte: Silva (2016, p. 62)

Portanto, cf. Silva (2016), "pode-se concluir que o Perceptron se comporta como um classificador de padrões cuja função é dividir classes que sejam linearmente separáveis". A Figura 11 mostra uma reta posicionada na fronteira de separação entre as classes. Para este tipo de problema o perceptron é um classificador adequado.

As redes neurais artificiais (**RNA**) se formam quando diversos neurônios se combinam. De forma resumida, "uma rede neural artificial (RNA) pode ser vista como um grafo onde os nós são os neurônios e as ligações fazem a função das sinapses". Isto está demonstrado na Figura 12.

Figura 12 – Representação simplificada de uma RNA



Fonte: Ferneda (2006, p.26)

As redes neurais artificiais se diferem pelas suas arquiteturas e pela forma como os pesos associados às conexões são ajustados durante o processo de aprendizado. A *arquitetura* de uma rede neural restringe o tipo de problema no qual a rede poderá ser utilizada, e é definida pelo número de camadas (camada única ou múltiplas camadas), pelo número de nós em cada camada, pelo tipo de conexão entre os nós (*feedforward* ou *feedback*) e por sua topologia (HAYKIN, 2001, p. 46-49).

O desenvolvimento de uma rede neural artificial consiste em determinar sua arquitetura, ou seja, os números de camadas e de neurônios em cada camada, bem como o ajuste dos pesos na fase conhecida como treinamento (HAGAN; DEMUTH; BEALE, 1996; HAYKIN, 2001).

Uma das características mais importantes de uma rede neural artificial é a habilidade de aprender através de exemplos e fazer inferências sobre o que aprendeu, melhorando, assim, o seu desempenho. As RNA's utilizam um algoritmo de aprendizagem que serve, basicamente, para ajustar os pesos de suas conexões (HAYKIN, 2001; FERNEDA, 2006; LIMA; PINHEIRO; SANTOS, 2016; RUSSELL; NORVIG, 2013).

Aqui também há, cf. explicitado na seção 2.4, duas formas básicas de aprendizado, o supervisionado e o não-supervisionado.

2.8.3 Redes do tipo Perceptron de múltiplas camadas

As arquiteturas do tipo perceptron de múltiplas camadas (MLP) são os modelos de redes neurais mais utilizados e conhecidos. Elas, basicamente, consistem de uma camada de entrada e uma ou mais camadas intermediárias (ou ocultas) além, claro da camada de saída (uma ou mais unidades sensoriais - neurônios) (HAYKIN, 2001).

Os sinais de entrada são propagados camada a camada pela rede em uma direção, ou seja, da entrada para a saída (*feedfoward*). Esta arquitetura retrata uma generalização

do perceptron. Portanto, segundo Silva (2016, p. 26), As redes Perceptron de Múltiplas Camadas (PMC) são marcadas pela presença de ao menos uma camada intermediária, oculta, de neurônios, situada entre a camada de entrada e a camada neural de saída. Desta forma, elas possibilitam elevadas possibilidades de aplicações em muitas áreas do conhecimento, entre as principais: aproximação universal de funções, reconhecimento de padrões, identificação e controle de processos, previsões de séries temporais, otimização de sistemas entre muitos outros (HAYKIN, 2001).

A Figura 13 ilustra uma rede do tipo perceptron de multicamadas. O treinamento deste tipo de rede é do tipo supervisionado e, geralmente, se utiliza um algoritmo muito popular chamado *retropropagação* do erro (*error backpropagation*). Este algoritmo é baseado numa regra de aprendizagem que "corrige" o erro durante o treinamento (HAYKIN, 2001).

Substancialmente, o método de *retropropagação* é constituído de duas etapas: uma fase de propagação do sinal no sentido tradicional (*feedforward*) e uma de retropropagação do erro (*backpropagation*) de todas as camadas e seus respectivos pesos. Na fase de ida, os vetores de dados e pesos são aplicados às unidades de entrada, e seu efeito se propaga pela rede, camada por camada (HAGAN; DEMUTH; BEALE, 1996; HAYKIN, 2001).

Após isso, um conjunto de saídas é produzido como resposta da rede. Na retropropagação os pesos são ajustados de acordo com uma regra de correção de erro (normalmente uma função matemática como a Equação 2.6, a Equação 2.8, a Equação 2.7 ou Equação 2.9, sendo esta última uma das mais utilizadas em muitas arquiteturas, na prática) (HAYKIN, 2001; HAGAN; DEMUTH; BEALE, 1996; YEUNG; BOTVINICK; COHEN, 2004).

A resposta da rede em um instante é subtraída da saída desejada (*target*) para produzir um valor de erro. Este valor de erro é propagado da saída para a entrada, camada a camada, de onde vem o nome "*retropropagação* do erro". Os pesos são, então, redefinidos de forma que a distância entre a resposta da rede e a resposta desejada seja reduzida (o erro seja minimizado). O processo é repetido diversas vezes até que uma tolerância global de erro seja assumida. Cada iteração é denominada época (*epoch*) (HAYKIN, 2001; HAGAN; DEMUTH; BEALE, 1996; MINSKY; PAPERT, 1969).

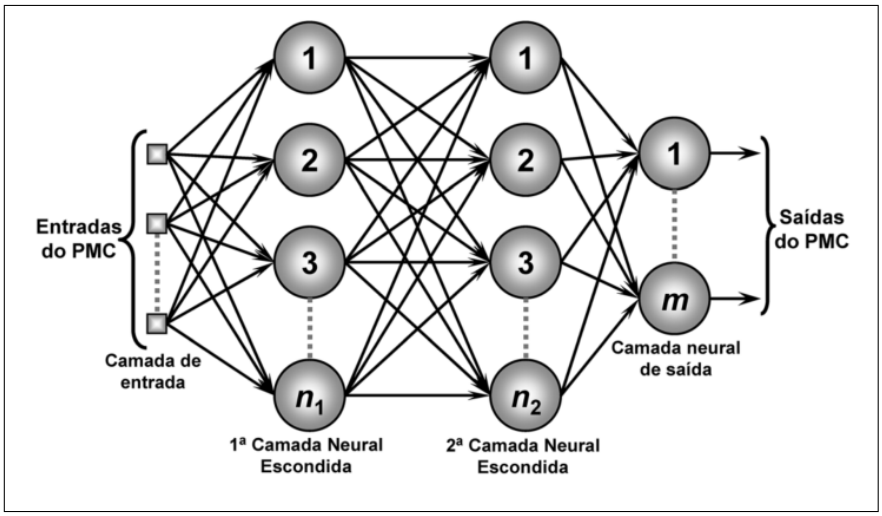
Uma arquitetura de um MLP (*Multi-Layer Perceptron* - Perceptron Multicamadas), possui, portanto três propriedades distintas: a função de ativação, o número de camadas ocultas e forma das conexões (totalmente conectada ou não).

2.8.4 Funções de ativação

Como visto na subseção 2.8.2, o modelo perceptron, lida bem com problemas linearmente separáveis, mas tem problemas ao lidar com problemas não-lineares (HAYKIN, 2001).

Para dar capacidade representativa às redes neurais artificiais, são essenciais as

Figura 13 – Rede Perceptron de multicamadas



Fonte: Silva (2016, p. 92)

diferentes funções de ativação, pois só assim elas conseguirão lidar um componente de não-linearidade, como o são a maioria dos problemas práticos (HAGAN; DEMUTH; BEALE, 1996).

Ao se introduzir ativações não-lineares, a superfície de custo da rede neural deixa de ser convexa fazendo com que a otimização se torne mais difícil (MINSKY; PAPERT, 1969; HAYKIN, 2001).

As principais funções de ativação, utilizadas na literatura ena prática, são, portanto:

2.8.4.1 Função de ativação limiar

Foi proposta na primeira definição de rede com neurônios artificiais (ROSENBLATT, 1958). O modelo de neurônio usado por RosenBlatt foi o mesmo sugerido por Mcculloch e Pitts (1943).

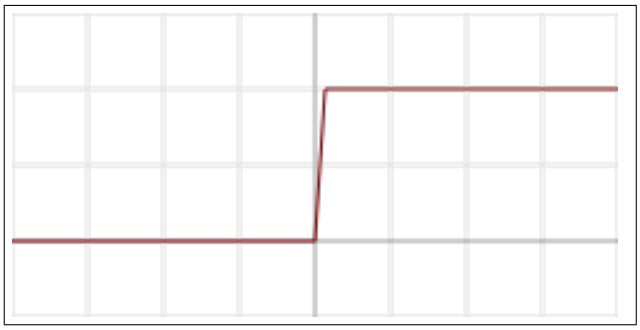
Neste modelo as saídas são binárias, ou seja, assumem, normalmente o valor 0 ou 1. A saída é 1 se o valor da entrada líquida for superior a um determinado valor chamado *thereshold*. Na maioria das vezes, esse valor é zero.

Matematicamente:

$$y_k = y = f_i(a_i(t)) = \begin{cases} 1, & \text{se } a_i(t) >= 0 \\ 0, & \text{se } a_i(t) < 0 \end{cases} \tag{2.27}$$

A Figura 14 mostra o 'funcionamento' da função de ativação limiar. Eventualmente podem ser utilizados os valores -1 e 1. Uma rede de camada simples que utiliza este tipo de ativação é o perceptron simples.

Figura 14 – Função de ativação Limiar



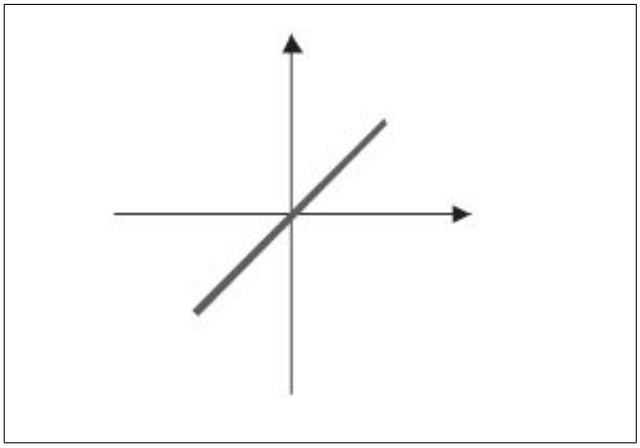
Fonte: Adaptado de Haykin (2001)

2.8.4.2 Função de ativação linear

A saída do neurônio, neste caso, é representada por uma função linear da forma descrita na Figura 15.

Redes que usam este tipo de função de ativação apresentam apenas uma camada de entrada e uma de saída. Possuem, portanto, uma série de limitações quanto ao que são capazes de representar (LIMA; PINHEIRO; SANTOS, 2016).

Figura 15 – Função de ativação Linear



Fonte: Adaptado de Haykin (2001)

2.8.4.3 Funções de ativação semilineares

As funções de ativação mais usadas deste tipo são a função logística e a tangente hiperbólica. Elas são populares por conta de suas derivadas (que são necessárias nas etapas de treinamento, particularmente no *Gradiente Descent* — descida do gradiente) poderem ser expressas a partir das próprias funções.

Estas funções, tanto a logística (ou *sigmoide*) e a tangente hiperbólica, respectivamente, podem ser expressas pelas equações 2.28 abaixo, onde a representa a entrada líquida da unidade (HAYKIN, 2001; LIMA; PINHEIRO; SANTOS, 2016).

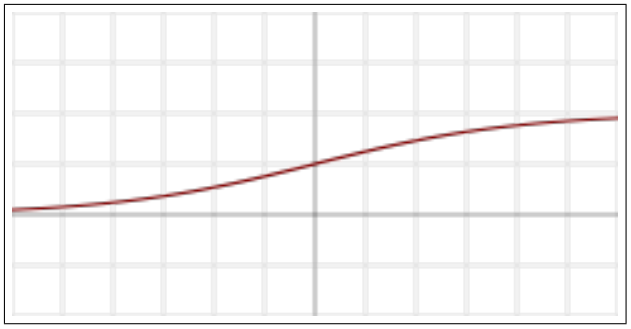
$$f(a) = \frac{1}{1 + e^{(-2\beta a)}} \qquad g(a) = tanh(\beta a) \tag{2.28}$$

E as respectivas derivadas das funções 2.28 acima são dadas por:

$$f'(a) = 2\beta f(1 - f) \qquad g'(a) = \beta(1 - g^2) \tag{2.29}$$

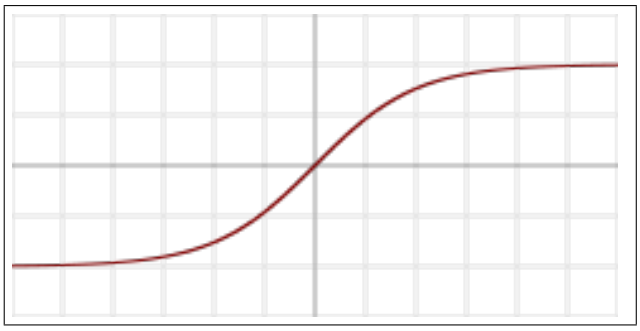
A Figura 16 mostra o comportamento da função sigmoide e a Figura 17 demonstra o da função tangente hiperbólica.

Figura 16 – Função de ativação Logística (sigmoide)



Fonte: Adaptado de Haykin (2001)

Figura 17 – Função de ativação Tangente Hiperbólica



Fonte: Adaptado de Haykin (2001)

2.8.4.4 Função de ativação ReLu

A função ReLU é a unidade linear retificada. É definida como, Academy (2020):

$$ReLU(x) = max(0, x) \tag{2.30}$$

Sua derivada é dada por:

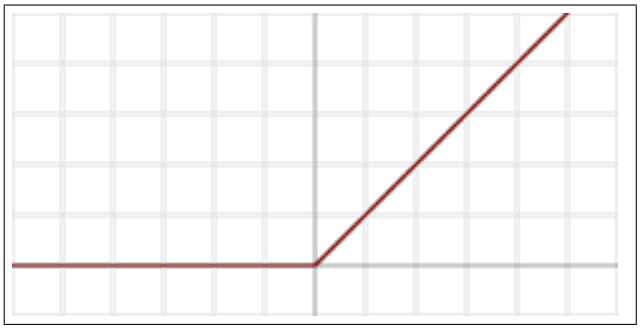
$$ReLU'(x) = \begin{cases} 1, & \text{se } x > 0 \\ 0, & \text{caso contrário} \end{cases} \tag{2.31}$$

Conforme, Academy (2020), ReLU é a função de ativação mais utilizada ao projetar redes neurais hodiernamente. Antes de tudo, a função ReLU é não linear, o que significa que se pode copiar os erros para trás e ter várias camadas de neurônios ativados pela função ReLU.

Ainda, de acordo com Academy (2020), a principal vantagem de utilizar a função ReLU é que ela não ativa todos os neurônios ao mesmo tempo. Se a entrada for negativa, ela será convertida em zero e o neurônio não será ativado. Isso significa que, ao mesmo tempo, apenas alguns neurônios são ativados, tornando a rede esparsa, eficiente e fácil para a computação.

A Figura 18 demonstra o comportamento da função de ativação ReLu em função da entrada

Figura 18 – Função de ativação ReLu - unidade linear retificada



Fonte: Adaptado de Haykin (2001)

Redes com a função ReLU são incomplexas de otimizar porque a função é parecida com a função identidade. A única diferença é que a ReLU produz zero em metade de seu domínio (HAGAN; DEMUTH; BEALE, 1996).

2.8.5 Otimização

Redes Neurais tem uma etapa iterativa em que os pesos são ajustados e se pode realizar otimizações para melhorar o erro e incrementar o modelo (HAYKIN, 2001).

Normalmente, o fluxo de treinamento se resume nos seguintes passos iterativos:

1. Operar a entrada na rede
2. Cálculo da função de perda (*loss function*) ou outra função de cálculo de erro
3. Cálculo do gradiente
4. Atualização dos pesos
5. Volta para o passo 1

Este processo é repetido até que um hiperparâmetro de tolerância seja alcançado para o cálculo dos erros mediante a *loss function* ou outra função de cálculo do erro.

O cálculo do gradiente descrito anteriormente diz respeito, cf. Academy (2020) a um dos mais usados algoritmos para otimizar a tarefa de aprendizagem dos pesos de uma rede neural.

A Descida do Gradiente é uma ferramenta padrão para otimizar funções complexas iterativamente. Seu objetivo é dada alguma função arbitrária, encontrar um mínimo. Para alguns pequenos subconjuntos de funções — aqueles que são convexos — há apenas um único *minimum* que também acontece de ser global. Para as funções mais realistas, pode haver muitos mínimos, então a maioria dos mínimos são locais. É preciso, assim, que a otimização encontre o "melhor" *minimum* e não fique preso em mínimos sub-otimistas (um problema comum durante o treinamento do algoritmo) (ACADEMY, 2020)

O algoritmo consiste basicamente de subtrair o valor do gradiente ∇f dos pesos w da rede, assim:

$$w_i = w_i - \alpha \times \nabla f_i \tag{2.32}$$

Sendo α o multiplicador que nos permite controlar o tamanho do passo de otimização. ∇f é a derivada da função de ativação no ponto específico. α é um hiperparâmetro conhecido como taxa de aprendizado e repreenta a *velocidade* em que a rede neural "aprende" os melhores pesos para o problema específico (HAYKIN, 2001).

Uma iteração consiste em um passo de otimização como o descrito em 2.8.5 e corresponde a uma ligação sináptica de *forward* na rede e uma de *backpropagation*. Isto constitui uma **época**. Normalmente, muitas épocas são necessárias, visto que o aprendizado em uma rede neural é, geralmente, lento para que a otimização evite os mínimos locais.

Na próxima seção, conceitos sobre SVM (Support Vector Machines) serão descritos e explorados. Este será um algoritmos específicos utilizado neste trabalho para sustentar a hipótese, juntamente com as RNA's e as Florestas Aleatórias (Random Forest).

2.8.5.1 TensorFlow e Pytorch

De acordo com Géron (2019, p. 233), o *TensorFlow* é uma biblioteca de software para cálculo numérico de código aberto especialmente adequada e ajustada para o Aprendizado de Máquina em larga escala. Foi desenvolvido pela *Google Brain Team* para uso intensivo de redes neurais profundas. O código fonte está à disposição no link: <https://github.com/tensorflow/tensorflow>.

É possível usar computação paralela em várias CPU's ou GPU's além de suportar computação distribuída para que seja possível o treinamento e o uso de redes neurais em grandes conjuntos de treinamento dividindo os cálculos por centenas de servidores em um período de tempo razoável (GÉRON, 2019).

Entre suas características importantes destacam-se: rodar em diversos sistemas operacionais (e inclusive na nuvem); diversas APIs públicas para criação, treinamento e avaliação de arquiteturas de diferentes tipos de redes neurais; diversas outras API's de alto nível foram construídas com base no TensorFlow como o *Keras* e o *Pretty Tensor*; implementações em C++ altamente eficientes para operações de Aprendizado de Máquina; nós de otimização avançados para procura por parâmetros que minimizem uma função de custo; usa extensões CUDA, uma API destinada a computação paralela, GPGPU, e computação heterogênea, criada pela Nvidia que dá acesso ao conjunto de instruções virtuais da GPU e a elementos de computação paralela (GÉRON, 2019).

Os cálculos no *TensorFlow* são expressos como grafos de fluxo de dados , seu nome deriva das operações que as redes neurais realizam em arranjos de dados multidimensionais, chamados de "tensores"(generalização matemática de escalares, vetores e matrizes) (KADIMISETTY, 2018).

2.9 SVM - Support Vector Machines

Segundo Cortes e Vapnik (1995), o algoritmo SVM (*Support Vector Machines*) é um dos mais efetivos para a tarefa de classificação.

De acordo com Goldschmidt e Passos (2005), no algoritmo SVM, o os dados de entrada são usados para construir uma *função de decisão* $f(x)$, tal que:

$$\begin{array}{llll} Se & f(x_i) \geq 0, & \text{então} & y_i = 1 \\ Se & f(x_i) < 0, & \text{então} & y_i = -1 \end{array} \tag{2.33}$$

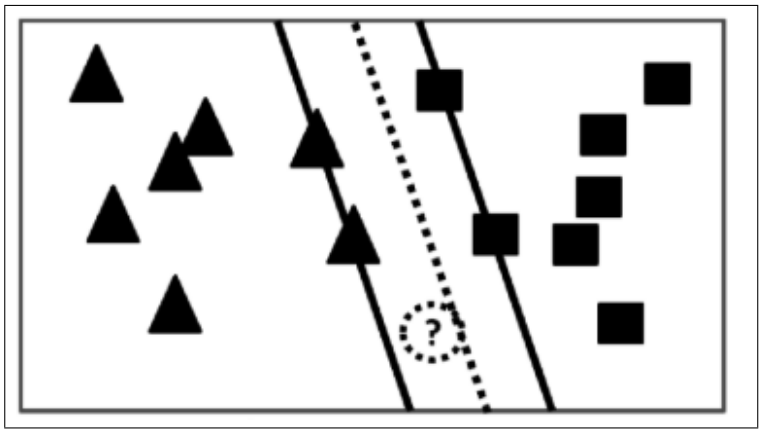
O algoritmo SVM está incluso nos denominados classificadores lineares, que separam o conjunto de dados por meio de um *hiperplano* que é a generalização do conceito de *plano* para dimensões maiores que três (GOLDSCHMIDT; PASSOS, 2005).

Assim, SVM, cf. Amaral (2016, p. 45) "é um algoritmo de classificação que maximiza as margens entre instâncias mais próximas, dessa forma, é criado um vetor otimizado que é então utilizado para classificar novas instâncias".

Conforme se vê na Figura 19, os dois vetores *não pontilhados* são as margens otimizadas. As instâncias por onde as margens otimizadas passam são os vetores de suporte. O vetor pontilhado é a referência para classificar novas instâncias. Assim, a nova instância, na Figura 19 é classificada como triângulo.

Seguindo o estudo de Mukkamala, Janoski e Sung (2002) há duas razões principais que levaram os autores do artigo citado de usarem SVMs para detecção de intrusão: a velocidade, pois a performance é prioritária para sistemas de detecção de intrusos e a escalabilidade, tendo em vista que as SVMs são indiferentes ao número de *data*

Figura 19 – Vetores de Suporte



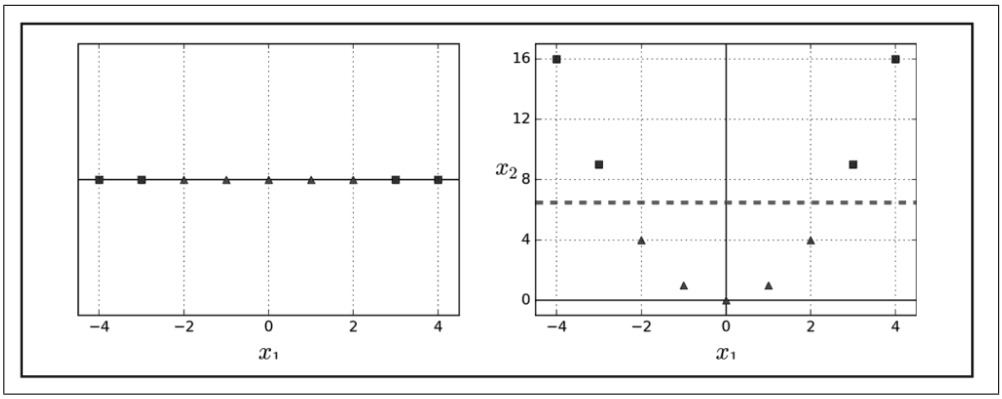
Fonte: Amaral (2016, p. 45)

points e a complexidade da classificação não depende da dimensionalidade do espaço de características. Em algumas aplicações podem ser usados, também, algoritmos de otimização para identificar mais características.

Embora os classificadores lineares SVM sejam eficientes e funcionem surpreendentemente bem em muitos casos, alguns conjuntos de dados não são linearmente separáveis. Adicionar mais características (dimensões), como as polinomiais é uma abordagem válida para lidar com os conjuntos de dados não lineares; em alguns casos, isso pode resultar em um conjunto de dados linearmente separável (GÉRON, 2019, p. 153)

Na Figura 20 observa-se que na plotagem à esquerda um conjunto de dados simples com apenas uma característica x_1. Esse conjunto não é linearmente separável (como a imagem mostra), mas se for adicionada uma segunda característica $x_2 = (x_1)^2$, o conjunto de dados resultante será linearmente separável.

Figura 20 – Adição de características para tornar um conjunto linearmente separável

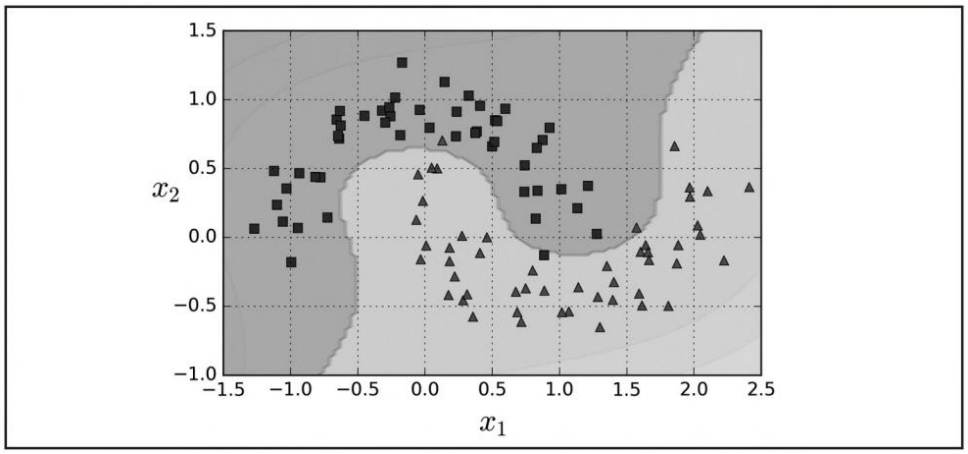


Fonte: Géron (2019, p. 153)

Isto pode ser visualizado na Figura 21 que também mostra a adição de características para tornar um problema linearmente separável.

O modelo de classificador SVM faz a previsão de uma nova instância $\mathbf{x}$, calculando

Figura 21 – Classificador Linear SVM com o uso de características polinomiais



Fonte: Géron (2019, p. 154)

uma função de decisão $w^T \cdot x + b = w_1 x_1 + \cdots + w_n x_n + b$: e se o resultado for positivo, a classe prevista $\hat{y}$ será a classe positiva (1), ou então será a classe negativa(0), caso contrário. (SOMAN; LOGANATHAN; AJAY, 2009).

$$\hat{y} = \begin{cases} 0, & \text{se } w^T \cdot x + b < 0, \\ 1, & \text{se } w^T \cdot x + b \geq 0 \end{cases} \tag{2.34}$$

2.9.1 Kernel

No aprendizado de máquina, um kernel é, em resumo, uma função capaz de calcular o produto escalar $\phi(a)^T \cdot \phi(a)$ com base somente nos vetores originais **a** e **b** sem a necessidade de calcular (ou mesmo saber) a transformação ϕ.

Diferentes kernels ajustam diferentes tipos de modelos e mudam a acurácia da classificação baseado na dimensionalidade dos dados analisados. A ideia básica é que quando um conjunto de dados for inseparável nas dimensões correntes, adicionar outra dimensão para que talvez assim os dados sejam separáveis (GÉRON, 2019; SOMAN; LOGANATHAN; AJAY, 2009).

Os kernels mais comuns são: (GÉRON, 2019)

$$Linear : K(a, b) = a^T \cdot b \tag{2.35}$$

$$Polinomial : K(a, b) = (\gamma a^T \cdot b + r)^d \tag{2.36}$$

$$RBFGaussiano : K(a, b) = exp(-\gamma ||a - b||^2) \tag{2.37}$$

$$Sigmóide : K(a, b) = tanh(\gamma a^T \cdot b + r) \tag{2.38}$$

Para que o classificador SVM seja utilizado com sucesso é necessário escolher uma função de kernel adequada, assim como os parâmetros dessa função, pois mesmo sendo um classificador com bom poder de generalização o seu desempenho depende da seleção dos parâmetros do kernel (SOMAN; LOGANATHAN; AJAY, 2009).

Neste trabalho, também foi usado, com eficácia que suportasse a hipótese (cf. se vê no Capítulo 3) o algoritmo SVM.

2.10 Random Forest - Florestas Aleatórias

O algoritmo de classificação ***Random Forest*** se baseia na ideia de ***ensemble*** para realizar diversas classificações e escolher qual é a "melhor". Ele usa um conjunto de treino com classificadores de árvores de decisão, cada um com um subconjunto aleatório diferente do conjunto de treinamento.

Para realizar as previsões, todas as árvores geradas individualmente são usadas e então a classe que obtém a maioria dos votos (ou pela média de todas as árvores individuais) é a escolhida (ALPAYDIN, 2014; BRUCE; BRUCE, 2019; GÉRON, 2019).

No Random Forest, são criadas várias árvores de decisão[7] (a quantidade é um hiperparâmetro). O Random Forest se baseia na aplicação de *bagging*[8] em árvores de decisão e além de reamostrar os registros (as instâncias), o algoritmo também amostra as variáveis (BRUCE; BRUCE, 2019, p. 237 e 238).

As árvores de decisão são propensas ao *overfitting* e está é uma das principais críticas a este algoritmo. Mesmo assim elas são base para outros modelos mais complexos, como o random forest. Isto porque com a combinação ds previsões de muitos modelos é possível melhorar as deficiências de cada um deles separadamente.

As Florestas Aleatórias (Random Forests) são, portanto, combinações de muitas árvores de decisão. Geralmente, à medida que a quantidade de árvores geradas aumenta a variância do conjunto geral diminui, isto facilita a generalização para novos dados (instâncias novas).

[7] As Árvores de Decisão, ou Decision Trees, determinam normas para tomada de decisão. O algoritmo cria uma estrutura similar a um fluxograma, com "nós" onde uma condição é verificada, e se atendida o fluxo segue por um ramo, caso contrário, por outro, sempre levando ao próximo nó, até a finalização da árvore que é a classificação desejada. (KESAVARAJ; SUKUMARAN, 2013)

[8] "Bagging funciona como o algoritmo básico de agrupamentos, exceto pelo fato de que, em vez de ajustar diversos modelos aos mesmos dados, cada modelo é ajustado a uma reamostra *bootstrap*". (BRUCE; BRUCE, 2019, p.237). A ideia geral do método de *bagging* é que uma combinação de modelos de aprendizagem aumenta o resultado geral

2.11 Trabalhos Relacionados

Esta seção expõe alguns dos principais trabalhos encontrados na literatura que estão, de alguma forma, relacionados a esta dissertação. Foram estudadas e analisados trabalhos acerca das políticas de segurança da informação, abordando a detecção de conflitos em diversos contextos como identificação e resolução de conflitos aéreos, rodoviários e de *normas* além de trabalhos sobre mineração de dados e técnicas de aprendizagem de máquina tanto em conjunturas distintas como regressão e classificação quanto para localização e solução de conflitos, no caso específico das políticas computacionais, principalmente sendo usadas em registros de patentes americanas fechadas — o que, de certa forma, evidencia o quanto este tema está na vanguarda da pesquisa na área.

Todos os trabalhos descritos foram base para o estudo, amadurecimento bibliográfico e aprofundamento teórico sobre o problema e as soluções propostas neste trabalho, principalmente aqueles relacionados a intrusões, detecção de conflitos aéreos e as soluções baseadas em aprendizado de máquina, usando classificadores utilizados neste trabalho, como k-NN, Naïve Bayes, Random Forest, redes neurais e máquinas de vetores de suporte (SVM) pois serviram de inspiração e ofereceram estímulo para que as técnicas pudessem ser extrapoladas para o uso na detecção de conflitos em políticas, tema desta de dissertação.

Como forma de organização, os trabalhos serão divididos em 4 seções:

- Verificação Detecção de conflitos;
- Mineração de Políticas;
- Resolução de outros tipos conflitos e
- mineração de dados e aprendizagem de máquina em geral.

2.11.1 Verificação e Detecção de conflitos

Esta seção aborda os artigos e trabalhos relacionados a respeito de verificação e detecção de conflitos. Estes são os trabalhos-base no estudo da detecção de conflitos, da determinação do modelo das políticas utilizadas e analisadas além de serem a fonte de algumas das principais definições empregadas nesta dissertação.

2.11.1.1 (KUANG et al., 2020)

Este artigo aborda o conflito de modalidade e os atributos de condição complexos tais como restrições espaciais e temporais afirmando que artigos e abordagens anteriores não tratam destas condições específicas. Os autores propuseram um modelo de detecção de conflito de modalidade (o conflito direto) aplicável durante a avaliação da política (em tempo de execução) e que ainda suporta uma regra de propagação de autorização para

investigar relacionamentos de classe-subclasse de um sujeito,recurso, ação e localização de uma política (ou seja, políticas hierárquicas com atualização entre as subpolíticas). No modelo eles usam políticas reais do tipo XACML — *eXtensible Access Control Markup Language*, que é uma arquitetura para políticas baseada em XML para uma universidade, gerenciamento de conferências e o domínio da saúde.

Para avaliar a eficiência do modelo os autores rodaram vários testes avaliando a Precisão (que traduz daqueles que o classificador indicou como corretos, quantos efetivamente eram), o Recall (a sensibilidade, ou seja, a capacidade do modelo em predizer corretamente a condição para casos que realmente a têm) e o F-Measure (ou F1 que combina precisão e recall de modo a trazer um número único que indique a qualidade geral do seu modelo e trabalha bem até com conjuntos de dados que possuem classes desproporcionais).

Este trabalho relaciona-se com esta dissertação pois *aborda detecção de conflitos de modalidade, os mesmos abordados nesta dissertação* (onde eles são chamados de **conflitos diretos**). Este artigo também usa, assim como esta dissertação, políticas geradas randomicamente (e também requisições geradas aleatoriamente), um número alto para cada um dos domínios analisados (saúde, universidade e gerenciamento de conferências) para realizar os testes embora os autores afirmem que a maioria dos sistemas do mundo real usa muito menos do que a quantidade de políticas geradas por eles em cada domínio (10 mil).

A justificativa que os autores usam para a geração randômica de políticas é devido ao fato de que há uma falta distinta de conjuntos de dados de solicitação real em ambiente distribuído. Ao contrário desta dissertação este artigo usa ontologias para cada domínio de aplicação como fonte para gerar as solicitações aleatórias (citado como limitação na seção 1.6).

Como já citado, o modelo de políticas é baseado em XML, o XACML que embora baseado no RBAC e no ARBAC difere bastante do usado nesta dissertação. Neste artigo é usada a precisão, o recall e f-measure como métricas, mas ele não realiza testes estatísticos que se justificam nesta dissertação, pois são realizados, em batch, muito mais experimentos por algoritmo (150 para um dos 3 algoritmos). Os autores realizam uma análise em que alteram a quantidade de atributos em seus experimentos — Em um ele usa 3 atributos (sujeito, recurso e ação) e em outro eles usam 4 atributos (sujeito, recurso, ação e condição).

As contribuições deste trabalho para esta dissertação incluem: i) o conflito estudado que é o mesmo (conflitos de modalidade, i.e., conflitos diretos); ii) políticas geradas randomicamente; iii) métricas de avaliação e validação.

2.11.1.2 Sarkis (2017)

Em Sarkis (2017) se propõe também uma abordagem para detectar conflitos indiretos entre políticas, algo não explorado nesta dissertação. Assim, as contribuições

deste trabalho incluem: a definição da política utilizada; o conjunto de relacionamentos que podem ocorrer entre os atributos que compõem uma política (organização, sujeitos, papéis, objetos, visões, ação e atividades); o conjunto de regras de propagação e de regras de conflitos (utilizados nos conflitos indiretos). Uma ferramenta foi desenvolvida, o "Conflict Detector" que recebe como entrada um conjunto de políticas de controle de acesso, um conjunto de relacionamentos entre os atributos destas políticas e fornece como saída, no caso da existência de conflitos, a identificação do conflito e as políticas conflitantes.

2.11.1.3 Vijayalakshmi e Jayalakshmi (2020)

Este artigo analisa e detecta as anomalias importantes nas políticas de controle de acesso baseadas em atributos—ABAC. Apresenta uma abordagem que usa o nível de prioridade para evitar o conflito nas políticas. Esta abordagem agrupa as regras das políticas ABAC com base no nível de prioridade e semelhança com a técnica de agrupamento, e detecta as anomalias em cada cluster em vez de todas as regras.

2.11.1.4 Zheng e Xiao (2019)

Aborda o controle de acesso baseado em atributos (ABAC) nos contextos da computação em nuvem e Internet das coisas. Cita que os conflitos entre as de políticas ocorrem com frequência devido à sua grande escala e expressão complexa. O trabalho propõe, então, um novo método para detectar conflitos entre regras ABAC com o auxílio da conversão de regras em um conjunto de sequências binárias. Em seguida, a detecção de conflito é alcançada através da operação bit a bit AND da sequência binária, o que melhora a eficiência da detecção.

2.11.1.5 Yahiaoui, Zinedine e Harti (2018)

Também aborda o controle de acesso baseado em atributos (ABAC) afirmando que este pode trazer grandes benefícios no contexto de compartilhamento e proteção de dados confidenciais. Declara também que esses sistemas são complexos de analisar e geralmente sofrem de anomalias e conflitos entre as políticas de segurança. Neste artigo, os autores lidam com o problema de várias políticas fornecerem respostas conflitantes (permitir / negar) para uma mesma solicitação de acesso e é proposta uma estrutura que pode apoiar os administradores de segurança com ferramentas para analisar, detectar e resolver continuamente quaisquer possíveis conflitos nos sistemas ABAC.

2.11.1.6 Silvestre (2017)

Este trabalho estuda a verificação de conflitos entre múltiplas normas em sistemas multiagentes (SMA). As normas, nesta tese, são semelhantes às políticas, inclusive em suas definições, mas restritas a um contexto de agentes autônomos. Para a resolução de conflitos,

o autor usa uma estratégia de aplicação de filtros para suavizar o custo computacional e utiliza transformação deôntica para análise de diversas normas ao mesmo tempo.

2.11.1.7 Sun et al. (2011)

Este artigo propõe uma estrutura para políticas e mecanismos de controle de acesso que preservam a privacidade e descreve algoritmos para problemas conflitantes de políticas de acesso. O mecanismo reforça a política de acesso aos dados que contêm informações de identificação pessoal. O principal componente da estrutura é o *propósito* envolvido nos modelos de controle de acesso (PAC) que fornecem suporte para expressar políticas complexas relacionadas à privacidade, levando em consideração recursos como objetivos, condições e obrigações. Os autores comentam que podem surgir problemas de conflito de políticas quando novas políticas de acesso são geradas e podem entrar em conflito com as políticas existentes. A estrutura da política de controle de acesso, incluindo condições e obrigações, é estudada. Com base na política de acesso, os modelos de autorização e as operações de política são analisados. E, também os conflitos entre as políticas.

2.11.1.8 Hwang et al. (2010)

Como Sarkis (2017), os autores também desenvolvem uma ferramenta para resolver o problema da detecção de conflitos, chamada ACPT (Teste de Política de Controle de Acesso), que ajuda a modelar e implementar políticas corretamente durante a modelagem, implementação e verificação de políticas.

2.11.1.9 Shu, Yang e Arenas (2009)

Este artigo propõe um método otimizado para detectar os conflitos entre regras estatisticamente conflitantes em uma política ABAC. O método inclui duas técnicas de otimização: redução de regras e busca binária. As políticas ABAC são fartamente explicadas e exploradas neste trabalho.

2.11.2 Mineração de políticas

2.11.2.1 Bui, Stoller e Li (2019)

O contexto do estudo é o controle de acesso baseado em relacionamento (ReBAC) que, segundo os autores, fornece um alto nível de expressividade e flexibilidade que promove a segurança e o compartilhamento de informações. O ReBAC é como uma extensão orientada a objetos do controle de acesso baseado em atributos (ABAC) em que os relacionamentos são expressos usando campos que se referem a outros objetos e as expressões de caminho são usadas para seguir cadeias de relacionamentos entre objetos. Este artigo apresenta dois algoritmos para mineração de políticas ReBAC: um algoritmo

guloso guiado por heurísticas e um algoritmo evolutivo baseado em gramática. Os autores afirmam que os algoritmos de mineração de políticas ReBAC têm potencial para reduzir significativamente o custo de migração de sistemas de controle de acesso legados para ReBAC, automatizando parcialmente o desenvolvimento de uma política ReBAC a partir de uma política de controle de acesso existente.

2.11.2.2 Chakraborty, Sandhu e Krishnan (2019)

O artigo aborda a tecnologia de controle de acesso baseado em atributo (ABAC) focando no problema de converter políticas de sistemas de controle de acesso já implantados para ABAC. Diversas variações desse problema geral podem ser definidas, em particular, o problema de mineração de política ABAC. Neste artigo, os autores formalizam o problema de existência do ABAC RuleSet neste contexto e desenvolvem um algoritmo e uma análise de complexidade para sua solução. É introdu zida, neste artigo, ainda a noção de Correção de inviabilidade do conjunto de regras ABAC junto com um algoritmo para a solução.

2.11.2.3 Kalaskar e Barkade (2018)

Neste artigo, os autores propõem uma técnica conhecida como algoritmo de FP-Growth para minerar regras de associação no contexto de uso desse algoritmo ter a capacidade de reduzir o custo de migração para ABAC. O artigo mostra o algoritmo na mineração de políticas ABAC.

2.11.2.4 Xu e Stoller (2013)

O controle de acesso baseado em funções (RBAC) é amplamente usado, mas tem limitações notáveis, o que leva a uma mudança para o controle de acesso baseado em atributos (ABAC). No entanto, o custo de desenvolver uma política ABAC pode ser um obstáculo significativo para a migração do RBAC para o ABAC. Este artigo apresenta a primeira definição formal do problema de mineração de políticas ABAC a partir de políticas RBAC e dados dos atributos, e o primeiro algoritmo projetado especificamente para extrair uma política ABAC de uma política RBAC e dados dos atributos.

2.11.2.5 Hachana, Cuppens-Boulahia e Cuppens (2015)

Neste artigo, os autores fazem uma abordagem de mineração de políticas por meio de um processamento adicional para uma mineração de políticas de controle de acesso à rede. Desenvolvem o problema de integração de políticas Net-RBAC resultantes da mineração de políticas em vários firewalls. Além disso, mostram como verificar as propriedades de segurança relacionadas à consistência da implantação nos firewalls. Usam, portanto, a mineração de dados em políticas de controle de acesso em um contexto de regras de um firewall (na realidade, de vários firewalls).

2.11.2.6 Martin e Tao Xie (2006)

Segundo os autores, para aliviar a carga de implementação e manutenção de aspectos de controle de acesso em um sistema, uma tendência crescente entre os desenvolvedores é escrever políticas de controle de acesso em uma linguagem de especificação, como XACML e integrar as políticas com aplicativos através do uso de um ponto de decisão de política (PDP). Para garantir que as políticas especificadas reflitam as esperadas, pesquisas desenvolveram ferramentas de verificação de políticas; no entanto, seus aplicativos na prática ainda são limitados, sendo restringidos pelo conjunto limitado de recursos de linguagem de política suportados e a indisponibilidade de propriedades de política. Este artigo apresenta uma abordagem de mineração de dados para o problema de verificar se as políticas de controle de acesso expressas refletem os verdadeiros desejos do autor da política. Foi desenvolvida uma ferramenta para investigar essa abordagem gerando solicitações automaticamente, avaliando essas solicitações para obter respostas e aplicando o aprendizado de máquina aos pares de solicitação-resposta para inferir propriedades de política. Essas propriedades inferidas facilitam a inspeção do comportamento da política. Os resultados mostram que algoritmos de aprendizado de máquina podem fornecer informações valiosas sobre as propriedades básicas da política e ajudar a identificar solicitações específicas de exposição a bugs.

2.11.3 Detecção e Resolução de outros tipos conflitos

2.11.3.1 Lupu e Sloman (1999)

Os autores deste trabalho discorrem sobre conflitos no gerenciamento de sistemas distribuídos com base em políticas de controle de acesso. O artigo analisa os conflitos de políticas, concentrando-se nos problemas de detecção e resolução dos mesmos. Discute-se, no artigo, os vários relacionamentos de precedência que podem ser estabelecidos entre as políticas e é apresentado uma ferramenta de análise de conflitos que faz parte de uma estrutura de gerenciamento baseada em funções, porém, sem usar mineração de dados ou aprendizagem de máquina.

2.11.4 Mineração de dados e aprendizagem de máquina

2.11.4.1 Guerrero-Higueras, DeCastro-Garcia e Matellan (2018)

Guerrero-Higueras, DeCastro-Garcia e Matellan (2018) aborda um método para construir modelos de *aprendizagem de máquina* com o objetivo de detectar ataques cibernéticos em RTLSs (*Real Time Location Systems* - sistemas de localização em tempo real) em um ambiente de cibersegurança para sistemas robóticos usando técnicas de *Machine Learning* (aprendizagem de máquina). O artigo mostra que os ciberataques nos sistemas de localização em tempo real para sistemas robóticos podem ser detectados por

um sistema criado usando o aprendizado supervisionado. Além disso, mostra que alguns tipos de ciberataques em sistemas de localização em tempo real, especificamente negação de serviço e falsificação(*DoS* e *Spoofing*), podem ser detectados por um sistema construído usando técnicas de aprendizado de máquina. Oito classificadores e algoritmos preditores conhecidos foram avaliados neste artigo e a análise de validação cruzada mostrou que os classificadores MLP (*Multi Layer Perceptron*) funcionam melhor que os outros obtendo maior acurácia e menor erro, sendo também o modelo com menor *overfitting* e maior *sensibilidade*[9].

2.11.4.2 Bui, Stoller e Le (2019)

Bui, Stoller e Le (2019) versa acerca de mineração de dados em políticas de controle de acesso, especificamente do modelo ReBAC (*Relationship-Based Access Control* - controle de acesso baseado em *relacionamento*), mas não foca na detecção de conflitos entre as políticas, e sim, na propagação de seus relacionamentos na criação de novas políticas. No artigo, os algoritmos de mineração de política do ReBAC propostos puderam reduzir significativamente o custo da migração dos sistemas de controle de acesso legados para o ReBAC, automatizando parcialmente o desenvolvimento de uma nova política do ReBAC.

2.11.4.3 Obaidat e Macchairolo (1994) e Mukkamala, Janoski e Sung (2002)

Obaidat e Macchairolo (1994) aborda um sistema de rede neural multicamadas para segurança de acesso a computadores com o objetivo de identificar usuários do mesmo. Os vetores de entrada foram compostos pelos intervalos de tempo entre pressionamentos de teclas sucessivos criados pelos usuários ao digitar uma sequência conhecida de caracteres. Usando aprendizado supervisionado, cada vetor de entrada foi classificado em uma das várias classes, identificando assim o usuário que digitou a sequência de caracteres. Neste artigo, uma precisão máxima de classificação de 97,5% foi alcançada usando um classificador de padrões baseado em rede neural multicamadas *feedforward* treinada usando *backpropagation*, o algoritmo de retropropagação. Essa abordagem visa melhorar a segurança do acesso ao computador. Este artigo trouxe contribuições para esta proposta de dissertação na maneira de construir a arquitetura da rede neural e também no ajuste dos *hiperparâmetros*. Na mesma linha, o artigo de Mukkamala, Janoski e Sung (2002) estuda a detecção de invasões usando redes neurais e máquinas de vetores de suporte e a ideia central é descobrir padrões úteis ou características que descrevam o comportamento intrusivo de um usuário em um sistema e os autores usam este conjunto de características para construir classificadores que puderam reconhecer anomalias e intrusões conhecidas em tempo real. É usado um conjunto de dados de referência de uma competição de KDD (*Knowledge Discovery in Databases* — Descoberta de Conhecimento em Bases de Dados)

[9] Sensibilidade é a proporção de verdadeiros positivos: a capacidade do sistema em predizer corretamente a condição para casos que realmente a têm - É também conhecida como ***Recall***.

projetada pela DARPA (Defense Advanced Research Projects Agency), e é demonstrado que classificadores eficientes e precisos podem ser construídos para detectar invasões. Ao final é comparado o desempenho de redes neurais e máquinas de vetores de suporte para a detecção de intrusões. Nesta dissertação, baseando-se neste artigo de Mukkamala, Janoski e Sung (2002) também serão avaliados os desempenhos de redes neurais e máquinas de vetores de suporte, mas para detecção de conflitos em políticas.

2.11.4.4 Jin, Cheu e Srinivasan (2002) e Debar, Becker e Siboni (1992)

O trabalho de Jin, Cheu e Srinivasan (2002) aborda o desenvolvimento e uso de uma rede neural probabilística construtiva (**CPNN** - *Constructive Probabilistic Neural Network*) na detecção de incidentes em rodovias, incluindo a construção e adaptação dos modelos. Esta CPNN foi estruturada com base no modelo Gaussiano de mistura e treinada por um algoritmo de ajuste dinâmico de decaimento (para correção dos erros). Os incidentes em rodovias, como colisões de veículos, podem ser extrapolados para um modelo de conflito entre agentes de um sistema. O modelo foi treinado e avaliado sobre um banco de dados de incidentes simulados em Singapura e adaptado para a rodovia I-880 na Califórnia sendo então investigada em ambientes on-line e off-line. Este artigo citado compara o desempenho do modelo CPNN e um modelo de rede neural probabilística básica (BPNN - Basic Probabilistic Neural Network). O artigo de Debar, Becker e Siboni (1992) estuda um componente de rede neural para um sistema de detecção de intrusão. O modelo *aprende* os hábitos que um usuário tem enquanto trabalha com o computador e emite avisos quando o comportamento atual não é consistente com os padrões *aprendidos* anteriormente. O modelo de de rede neural é usado para modelar o comportamento do usuário como uma característica componente para o sistema de detecção de intrusão.

2.11.5 Relacionamento dos trabalhos com a dissertação

Na detecção de conflitos em políticas, usam-se, comumente, abordagens como as de Sarkis (2017) e Mohan et al. (2012) — estritamente analíticas e formais, baseadas em análise de ontologias entre os atributos que compõem uma política e os relacionamentos e regras de propagação destas políticas ou mesmo estratégias como as de Wang et al. (2010) e Ferraiolo, Atluri e Gavrila (2011) que usam uma arquitetura que pode ser empregada como uma máquina de proteção de propósito geral usando métodos matemáticos formais para atingir seus objetivos. Dentro do conjunto de modelos formais, envolvendo relacionamentos, na detecção de conflitos, há também o trabalho de Sun et al. (2011), que usa o "propósito envolvido nos modelos de controle de acesso" para atingir seu fim. *Estes, fatalmente, analisam as políticas em pares ou em grupos sem filtros de agrupamentos.*

Existem os métodos baseados em requisitos como os propostos por He e Antón (2009) que integram a especificação de política no processo de desenvolvimento de software,

garantindo, assim, a consistência entre os artefatos de software e fornecendo orientação prescritiva sobre como especificar ACPs (*Access Control Policies)* — políticas de controle de acesso, mas que *tem o inconveniente de ocorrer durante o desenvolvimento e a fase de análise do sistema.*

Na literatura especializada são encontrados procedimentos como os descritos em Silvestre (2017) que utilizam lógica deôntica[10] para encontrar os conflitos. Estas abordagens citadas definem tipos de conflitos que podem ocorrer entre políticas computacionais e os tentam detectá-los, cada uma usando uma perspectiva específica.

Para detectar os conflitos entre políticas, nestes trabalhos citados anteriormente, estas foram analisadas, geralmente, em pares (e sem filtros para agrupamentos — quando os há) e mesmo quando foram verificadas múltiplas normas ou políticas, como em Silvestre (2017), *toda a base precisou ser "consultada"* ou *"varrida"* novamente a cada conjunto de novas instâncias de políticas inseridas ou analisadas no sistema (para que o conflito seja ou não detectado).

Hwang et al. (2010), assim como Sarkis (2017), desenvolvem uma ferramenta, chamada ACPT (Access Control Test Policy), que modelam e implementam políticas durante a modelagem, implementação e verificação de políticas e tem a desvantagem (em termos de custo computacional) de, também verificar as políticas em pares (ou em grupos maiores) e não contemplar essa análise no momento da inserção da política.

As técnicas e algoritmos de aprendizagem de máquina juntamente com as de mineração de dados foram utilizadas com resultados promissores na detecção de conflitos, principalmente em Obaidat e Macchairolo (1994), Chen (2011), bem como em Christodoulou e Kontogeorgou (2008) e Jin, Cheu e Srinivasan (2002). Estes estudos abordam problemas variados como detecção de colisões em voos, segurança de acesso computacional, incidentes em rodovias e intrusão de sistemas — todos de alguma forma relacionados à conflitos entre normas, regras, políticas ou direção e que são problemas que podem, tomadas as devidas precauções, serem extrapolados para o problema descrito e estudado neste trabalho.

Já os trabalhos de Bui, Stoller e Li (2019), Hachana, Cuppens-Boulahia e Cuppens (2015), Kalaskar e Barkade (2018), Martin e Tao Xie (2006), Chakraborty, Sandhu e Krishnan (2019) e os de Xu e Stoller (2013) e Xu e Stoller (2014) que analisam, de formas diferentes, aspectos relacionados com a mineração de dados em políticas de controle de acesso. Entretanto, a maioria dos trabalhos citados tem a desvantagem de abordarem o conflito de políticas, em si, de forma mais periférica e secundária (até acidental), sendo que em nenhum destes trabalhos, o foco principal é a detecção de conflitos entre políticas propriamente ditas (sejam eles quais forem), concentrando-se primordialmente seja em

[10] A lógica *deôntica* é um tipo de lógica usada para analisar de modo formal as normas e as proposições que tratam dessas normas (SILVESTRE, 2017)

migração/conversão de políticas de um modelo para outro; ou no problema de políticas refletirem fielmente o desejo do autor e até mesmo em outros contextos (muito específicos), como o gerenciamento de firewalls ou mesmo usando abordagens particulares demais, com o foco em regras de associação (que não se aplicam ou resolvem diretamente o problema investigado nesta dissertação).

3 Experimentos

Neste capítulo serão descritas as abordagens diferentes que foram utilizadas e em cada uma delas, os principais algoritmos de classificação que foram utilizados. A primeira parte dos experimentos focaram em determinar quais os classificadores apresentaram melhor acurácia para a resolução do problema descrito na subseção 1.1.3. A base para experimentos foi pequena, de apenas 68 políticas. Em seguida, dois outros datasets foram utilizados, um com 276 e outro com 392 políticas, o definitivo, em que a maioria dos experimentos foi realizada.

A descoberta do melhor algoritmo para a detecção de conflitos, deu-se através da execução dos algoritmos no último experimento em um processo de batch em um total de 30 vezes. Para avaliar o desempenho definiu-se o método cross-validation com 5 folds para cada uma das 30 execuções, pois alterando-se o *random state* a cada execução, garante-se que os valores de cada fold sejam diferentes a cada performance, totalizando, assim, 150 testes para cada um dos algoritmos.

As acurácias foram armazenadas em uma lista, em seguida, as médias aritméticas, médias quadráticas e outros testes estatísticos – como o teste Shappiro-Wilk, cf. Razali e Yap (2011) e um teste de Tukey, cf. Tukey (1970) — foram realizados para determinar o algoritmo com as melhores acurácias médias e melhores performances nas simulações executadas.

3.1 Visão geral dos experimentos

A forma geral de como a estrutura dos experimentos foram realizados para o problema proposto na subseção 1.1.3 é a seguinte:

1. Definição do problema;
2. Coleta de dados;
3. Pré-processamento dos dados;
4. Engenharia e seleção de atributos;
5. Modelagem: definição, configuração e arquitetura do modelo (ou modelagem dos hiperparâmetros);
6. Treinamento (Aprendizagem);
7. Testes e validação do modelo;

8. Avaliação e ajuste do modelo;
9. Apresentação dos resultados;

A figura 22 demonstra a forma geral dos experimentos realizados neste trabalho.

Figura 22 – Forma Geral dos experimentos



Fonte: compilação do autor. Adaptado dos modelos de Lima, Pinheiro e Santos (2016), Silva (2016) e Haykin (2001)

Este modelo de método experimental foi adaptado daqueles propostos em Lima, Pinheiro e Santos (2016), Silva (2016) e Haykin (2001). O item 1 já foi descrito na subseção 1.1.3. As próximas seções trazem os passos 2 a 9.

3.2 Base de dados, pré-processamento e recursos computacionais

3.2.1 Datasets

Esta seção detalha os pormenores dos conjunto de dados (*datasets*) utilizados nos experimentos do Capítulo 3.

De acordo com Chwif, Medina e Simulate (2014, p. 34), há situações em que não é possível conseguir dados reais para simulações e estudos acadêmicos, geralmente quando envolve confidencialidade e segurança de informações sensíveis dos clientes, organizações ou atores envolvidos no processo pois, como explicitado na seção 2.1, essas informações são valiosas e por questões estritas de segurança esses dados, geralmente, não são compartilhados. Este trabalho usa meios de geração automática e randomizada das políticas de segurança[1]. Trabalhos relacionados, como os de Sarkis (2017), Kuang et al. (2020) e Silvestre (2017), da mesma maneira, usaram formas de geração automática de políticas ou normas. Esta dissertação usou a mesma abordagem. Todos os *datasets* utilizados neste trabalho foram gerados com o mesmo programa, escrito em Python, chamado **`geracao_politicas.py`** e disponibilizado em <https://bit.ly/3qkJKKs>. A Figura 23 mostra o início do programa de geração automática de políticas. Já a Figura 24 e a Figura 25 demonstra a segunda e a terceira partes do programa de geração de políticas.

Para o primeiro experimento, um arquivo de políticas foi gerado randomicamente a partir do proposto em Sarkis (2017) e, de acordo com o exposto na subseção 2.2.3. O arquivo gerado possui, inicialmente 68 políticas nomeadas, constituindo a *fase de seleção*[2] da Mineração de Dados e da fase coleta de dado do método proposto anteriormente. Este arquivo foi usado nos testes preliminares da hipótese descrita na subseção 1.1.4 deste trabalho. Para o problema da detecção de conflitos diretos, como já citado, foram usadas técnicas de aprendizagem supervisionada. Para tanto, ao arquivo com as políticas, no *pré-processamento* foi acrescentada uma coluna rotulando os conflitos da seguinte forma: **1**: *conflito direto* e **0**: *sem conflito*. Esta estratégia é a mesma utilizada no trabalho de Davy, Jennings e Strassner (2008) onde ele usa um modelo de matriz de controle de acesso usando operações lógicas `AND` e `OR` para identificação de conflitos. A figura 26 demonstra o aspecto do arquivo das políticas geradas para os experimentos deste trabalho. Na imagem, pode-se notar a *classe* (coluna) criada para guiar o aprendizado supervisionado dos algoritmos utilizados no estudo.

Ainda na fase de *pré-processamento*, a coluna 9 (Conflito) foi transformada do tipo de dado *Numérico para Nominal*. Para isso foi usado o softwate WEKA, descrito em Witten et al. (2016), aplicado o filtro *NumericToNominal* do software. Além disso, tanto o primeiro atributo quanto as datas de ativação e desativação da política foram removidos, pois, dentro do escopo estudado neste trabalho, eles não influenciariam nos resultados finais. O primeiro atributo é o "nome" da política, na forma "PolicyXY", onde XY é um número inteiro sequencial, iniciando em 1. O dataset, ao final, ficou com seis atributos, 5 previsores (Acesso, Organizacao, Sujeito, Acao e Objeto) e uma classe (o atributo 'Conflito').

[1] Assim como em Sarkis (2017), Kuang et al. (2020) e Silvestre (2017)
[2] cf. seção 2.3 deste trabalho.

Figura 23 – Parte do programa para geração de políticas automaticamente

```
#Geração das políticas
"""
Autor   :  Edkallenn Lima
Data    :  18/04/2019
Função  :  Gera automaticamente as políticas de controle
           de acesso. As políticas são geradas a partir
           de listas com os valores das mesmas.
           As instâncias geradas são todas randômicas.
Obs.    :
"""
#Geração das políticas
from random import randint
import pandas as pd
import numpy as np
import time
import random
from time import sleep


def get_aleatorio(max):
  return random.randint(0,max-1)

def gera_uma_politica(num_politica):
  #str = '';
  milisegundos = int(round(time.time() * 1000))
  random.seed(milisegundos)

  data1 = random_date("1/1/2017 1:30 PM", "1/1/2018 4:50 AM", random.random())
  data2 = random_date("1/1/2019 1:30 PM", "1/1/2020 4:50 AM", random.random())
  numero = repr(num_politica+1)
  string = 'Policy' + numero + ', ' \
       + acesso[get_aleatorio(tamAcesso)] + ', ' \
       + org[get_aleatorio(tamOrg)] + ', ' \
       + sujeito[get_aleatorio(tamSujeito)] + ',' \
       + acao[get_aleatorio(tamAcao)] + ', ' \
       + objeto[get_aleatorio(tamObjeto)] + ', ' \
       + data1  + ', ' + data2
  #print(string)
  return string
```

Fonte: compilação do autor

Figura 24 – Segunda parte do programa para geração de políticas automaticamente

```
41  def exibe_politicas(array_politicas):
42    for i in range(0,len(array_politicas)):
43      linha = array_politicas[i]
44      print(linha)
45
46  def gera_politicas(politicas, quant):
47    for i in range(0,quant):
48      politicas.append(gera_uma_politica(i))
49      sleep(0.01)
50
51  def str_time_prop(data_inicial, data_final, formato, prop):
52      """Retorna uma data na proporção de um intervalo de dois tempos formatados:
53      - data_inicial e data_final devem ser strings especificando os tempos formatados
54        fornecendo um intervalo [data_inicial, data_final].
55      - prop especifica como uma proporção do intervalo a ser tomada após data_inicial.
56        O horário de retorno será no formato especificado.
57
58      Exemplo de uso:
59        data1 = random_date("1/1/2017 1:30 PM", "1/1/2018 4:50 AM", random.random())
60        data2 = random_date("1/1/2019 1:30 PM", "1/1/2020 4:50 AM", random.random())
61        print('Data1 = {}\nData2 = {}'.format(data1, data2))
62      """
63
64      stime = time.mktime(time.strptime(data_inicial, formato))
65      etime = time.mktime(time.strptime(data_final, formato))
66
67      ptime = stime + prop * (etime - stime)
68
69      return time.strftime(formato, time.localtime(ptime))
70
71  def random_date(data_inicial, data_final, prop):
72      return str_time_prop(data_inicial, data_final, '%d/%m/%Y %I:%M %p', prop)
```

Fonte: compilação do autor

3.2.2 Recursos computacionais

Dois **ambientes computacionais** foram utilizados para as tarefas de mineração: um **notebook** Intel Core i5 vPro-8350U (8ª Geração de 64 bits com 1.70GHz e 8 GB de RAM, com SSD de 256 GB rodando Windows 10 Pro. O outro ambiente foi uma **Estação de Trabalho** Intel Core i7 vPro-6700 de 8ª geração de 64 bits com 3.40 Ghz e 20 GB de RAM, com HD de 1 TB rodando o Windows 10 Pro. Foram usados dois ambientes para refletir as diferenças computacionais de ambos verificando como os experimentos rodavam em termos de desempenho.

3.2.3 Experimentos iniciais - arquivo com 68 políticas

O arquivo de 68 políticas criado com o programa descrito na Figura 23 está disponível no endereço: <https://bit.ly/2QUnqec>. A Figura 27 mostra como ficou o arquivo com 68 políticas após a execução do pré-processamento.

Figura 25 – Terceira Parte do programa para geração de políticas automaticamente

```
115  tamAcesso = len(acesso)
116  tamOrg = len(org)
117  tamSujeito = len(sujeito)
118  tamAcao = len(acao)
119  tamObjeto = len(objeto)
120
121  politicas = []
122  quantidade_gerada = 200
123  gera_politicas(politicas, quantidade_gerada)
124
125  teste = pd.DataFrame(np.array(politicas), columns=['politica'])
126  teste.to_csv('politicas_geradas.csv', index=False)
127
128  #retira as aspas
129  with open("politicas_geradas.csv", "rt") as fin:
130      with open("politicas_geradas_out.csv", "wt") as fout:
131          for line in fin:
132              fout.write(line.replace("\"" , ""))
133
134  #altera a primeira linha pelas colunas corretas
135  from io import StringIO
136  buffer = StringIO()
137
138  with open('politicas_geradas_out.csv', 'r') as stream:
139      for index, line in enumerate(stream):
140          # index == 0 representa a primeira linha do arquivo:
141          buffer.write('Politica,Acesso,Organizacao,Sujeito,Acao,Objeto\n' if index == 0 else line)
142
143  with open('politicas_geradas_final.csv', 'w') as stream:
144      stream.write(buffer.getvalue())
```

Fonte: compilação do autor

Figura 26 – Aspecto do arquivo das políticas geradas para os experimentos

```
Politica,Acesso,Organizacao,Sujeito,Acao,Objeto,DataAtivacao,DataDesativacao,Conflito
Policy01, Permitted,UFAC,Secretario_de_Curso_Academico,Abertura, Documentos, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy02, Forbidden,UFAC, Sandra_Maria_Soares_da_Rocha,Abertura, Documentos, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy03, Permitted,UFAC, null, Solicitar, Produtos, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy04, Forbidden,UFAC, Secretario_de_Centros_Academicos, Solicitar, Produtos, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy05, Permitted,UFAC, null,Acessar,Almoxarifado, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy06, Forbidden,UFAC, Jaider_Moreira_de_Almeida,Acessar,Almoxarifado, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy07, Permitted,UFAC, Secretarios, Solicitar, Materiais, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy08, Forbidden,UFAC, Secretario_de_Centros_Academicos, Solicitar, Materiais, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy09, Permitted,UFAC, Grupo_IPTU, Gerar, Planilhas_de_Calculo, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy10, Permitted,UFAC, Grupo_IPTU, Calcular, IPTU, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,1
Policy11, Forbidden,UFAC, Grupo_IPTU, Calcular, IPTU, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,1
Policy12, Permitted,UFAC, Socorro_Pontes,Acessar, Portal_do_Aluno, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy13, Permitted,UFAC, Socorro_Pontes, Matricular,Aluno, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy14, Permitted,UFAC, Jose_Rodrigues_Bardalles, Solicitacao, Central_de_Copias, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy15, Permitted,UFAC, Jose_Rodrigues_Bardalles, Analise, Central_de_Copias_Analise, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy16, Permitted,CCET, Jose_Rodrigues_Bardalles, Solicitacao, Central_de_Copias, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy17, Permitted,CCSJA, Jose_Rodrigues_Bardalles, Analise, Central_de_Copias, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy18, Permitted,UFAC, Grupo_Almoxarifado, Requisitar, Material, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy19, Forbidden,UFAC, Grupo_Almoxarifado, Criar, Guia_de_Requisicao, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy20, Forbidden,UFAC, Grupo_Almoxarifado, Inserir, Guia_de_Requisicao, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy21, Permitted,UFAC, Usuario_Qualquer, Cadastrar, Convencao, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,1
Policy22, Forbidden,UFAC, Usuario_Qualquer, Nova, Convencao, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,1
Policy23, Forbidden,UFAC, Usuario_Qualquer, Alterar, Convencao, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,1
Policy24, Forbidden,UFAC, Usuario_Qualquer, Cadastrar, Convencao, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,1
Policy25, Permitted,UFAC, Usuario_Qualquer, Nova, Convencao, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,1
Policy26, Permitted,UFAC, Usuario_Qualquer, Alterar, Convencao, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,1
Policy27, Permitted,UFAC, Socorro_Pontes, Solicitar, MatriculaAluno, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy28, Permitted,UFAC, Socorro_Pontes, Efetivar, MatriculaAluno, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy29, Permitted,UFAC, Joao_Josino, LancarMedia, Notas, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,1
Policy30, Permitted,UFAC, Joao_Josino, VoltarLancamento, Notas, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy31, Forbidden,UFAC, Joao_Josino, LancarMedia, Notas, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,1
Policy32, Permitted,UFAC, Joao_Josino, VoltarLancamento, Notas, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy33, Permitted,UFAC, Secretario_de_Unidade_Administrativa, SolicitacaoAbertura, Documentos, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
Policy34, Forbidden,UFAC, Secretario_de_Unidade_Administrativa, Cancelamento, Documentos, Tue Mar 24 16:23:59 GMT-05:00 2015, Thu Sep 24 16:23:59 GMT-05:00 2020,0
```

Fonte: compilação do autor

3.2.4 Resultados - Arquivo com 68 políticas

Logo após, experimentos foram realizados de forma preliminar no *dataset* envolvendo os diversos algoritmos com alguns parâmetros sendo alterados (a maioria com pequena ou nenhuma variação) para se chegar às técnicas finais que foram utilizadas nos posteriores experimentos e que serão explicitadas a seguir.

Figura 27 – Aspecto do arquivo com 68 políticas após o pré-processamento

	Acesso	Organizacao	Sujeito	Acao	Objeto	Conflito
0	Permitted	UFAC	Secretario_de_Curso_Academico	Abertura	Documentos	0
1	Forbidden	UFAC	Daenerys_Targaryen	Abertura	Documentos	0
2	Permitted	UFAC	Outro_Usuario	Solicitar	Produtos	0
3	Forbidden	UFAC	Secretario_de_Centros_Academicos	Solicitar	Produtos	0
4	Permitted	UFAC	Outro_Usuario	Acessar	Almoxarifado	0
5	Forbidden	UFAC	Tyrion_Lannister	Acessar	Almoxarifado	0
6	Permitted	UFAC	Secretarios	Solicitar	Materiais	0
7	Forbidden	UFAC	Secretario_de_Centros_Academicos	Solicitar	Materiais	0
8	Permitted	UFAC	Grupo_IPTU	Gerar	Planilhas_de_Calculo	0
9	Permitted	UFAC	Grupo_IPTU	Calcular	IPTU	1

Fonte: compilação do autor

Utilizando-se a ferramenta WEKA descrita em (WITTEN et al., 2016) para as últimas fases da Mineração de Dados, foram utilizados alguns algoritmos de classificação que segundo Wu et al. (2007) são alguns dos mais utilizados na Mineração de Dados. Para avaliar o desempenho definiu-se o método *cross-validatio*n com 10 folds. Em seguida suas acurácias foram comparadas.

A tabela 1 mostra o resultado destes experimentos:

Tabela 1 – Acurácia dos classificadores

Classificador/Algoritmo	Acurácia
Multi Layer Perceptron	0.9705
SVM kernel linear	0.9705
Random Forest	0.9542
J48	0.9411
K* (K-star)	0.9411
Trees LMT	0.9117
IBk (KNN, com k =1)	0.8970
JRip	0.8970
Nayve Bayes	0.8674
Random Tree	0.7794

Fonte: Elaborada pelo autor mediante experimentos

As figuras 28 e 29 mostram os resultados das classificações do arquivo de políticas usando, respectivamente, os classificadores/algoritmos: *SVM* e o *MultiLayer Perceptron* (que foram os principais citados nos trabalhos relacionados, cf. descrito na seção 2.11).

Figura 28 – Saída do software WEKA. Classificador: SVM

```
=== Classifier model (full training set) ===

LibSVM wrapper, original code by Yasser EL-Manzalawy (= WLSVM)

Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          66               97.0588 %
Incorrectly Classified Instances         2                2.9412 %
Kappa statistic                          0.9344
Mean absolute error                      0.0294
Root mean squared error                  0.1715
Relative absolute error                  6.6784 %
Root relative squared error             36.5941 %
Total Number of Instances               68

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,957    0,000    1,000      0,957    0,978      0,936    0,978     0,986     0
                 1,000    0,043    0,917      1,000    0,957      0,936    0,978     0,917     1
Weighted Avg.    0,971    0,014    0,973      0,971    0,971      0,936    0,978     0,964

=== Confusion Matrix ===

  a  b   <-- classified as
 44  2 |  a = 0
  0 22 |  b = 1
```

Fonte: compilação do autor

Cf. mostrado na Figura 28, na Figura 29 e como demonstrado na Tabela 1, os classificadores SVM e MultiLayerPerceptron ficaram com os mesmos valores em relação à acurácia.

Figura 29 – Saída do software WEKA. Classificador: *MultiLayer Perceptron*

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          66               97.0588 %
Incorrectly Classified Instances         2                2.9412 %
Kappa statistic                          0.9344
Mean absolute error                      0.0574
Root mean squared error                  0.1664
Relative absolute error                 13.029  %
Root relative squared error             35.5106 %
Total Number of Instances               68

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,957    0,000    1,000      0,957    0,978      0,936    0,996     0,998     0
                 1,000    0,043    0,917      1,000    0,957      0,936    0,996     0,992     1
Weighted Avg.    0,971    0,014    0,973      0,971    0,971      0,936    0,996     0,996

=== Confusion Matrix ===

  a  b   <-- classified as
 44  2 |  a = 0
  0 22 |  b = 1
```

Fonte: compilação do autor

Assim, com uma acurácia de 97,05% na classificação dos conflitos diretos, tanto o algoritmo Multilayer Perceptron (que implementa uma rede neural sigmoide multicamadas) quanto o SVM tiveram a maior acurácia, com 95,7% de *TP rate*(taxa de *True Positives* ou verdadeiros positivos) para a classe 0 (não há conflito) e, somente, 4,3% de *FP rate*(taxa de Falsos Positivos) para a classe 1 (quando há conflito direto). Nos experimentos realizados (assim como se esperava inicialmente na hipótese deste trabalho — baseado em evidências

da literatura), estes modelos algorítmicos foram os mais eficientes para a detecção de conflitos diretos.

A interface visual do WEKA é interessante para observar o comportamento inicial dos algoritmos, mas para as arquiteturas de redes neurais e seus diversos parâmetros, configurações e quantidade de camadas ocultas, entre outras configurações levaram a outros experimentos com outras abordagens. Para isso e com o objetivo de evitar o *overfitting*, como explicado na subseção 2.5.3.1 novos procedimentos e ferramentas foram adotados, incluindo, a alteração da estratégia de validação.

3.3 Outros experimentos - arquivos com 276 e 392 políticas

Foi gerado, então, dois novos arquivos de políticas, também randomicamente, e, ainda, a partir do proposto em Sarkis (2017) e, cf. o exposto na subseção 2.2.3 usando o software demonstrado na Figura 23. Os arquivos gerados agora, possuem, 281 e 430 políticas nomeadas. O arquivo de 281 políticas está disponível em <https://bit.ly/2MW1GwA>https://bit.ly/2MW1GwA. Arquivos maiores, com uma quantidade maior de políticas, se justificam para entender o comportamento dos modelos de mineração em cenários mais próximos da realidade. Ambos os arquivos como são criados com o mesmo software mantém características randomizadas semelhantes.

Figura 30 – Parte do arquivo com 281 políticas após o pré-processamento

265	Permitted	Unidade_Administrativa	Secretarios	Record	BuildDispatch	0
266	Permitted	CPA	Joao_Paulo	Analise	Central_de_Copias	0
267	Permitted	CFET	Joao_Calvino	Requisitar	Material	0
268	Forbidden	CFET	Joao_Calvino	Criar	Guia_de_Requisicao	0
269	Forbidden	CFET	Joao_Calvino	Inserir	Guia_de_Requisicao	0
270	Forbidden	IFAC	Ana_Ester	Generate	Process	0
271	Obliged	IFAC	Ana_Ester	Open	Process	1
272	Permitted	Unidade_Administrativa	Professor	Solicitacao	Central_de_Copias	0
273	Permitted	Unidade_Administrativa	Professor	Analise	Central_de_Copias_Analise	0
274	Permitted	Comite_Etica_Em_Pesquisa	Professor	Solicitacao	Central_de_Copias	0
275	Permitted	CPA	Professor	Analise	Central_de_Copias	0
276	Permitted	IFAC	Professor	Requisitar	Material	0
277	Permitted	IFAC	Coordenador	Requisitar	Material	0
278	Permitted	IFAC	Coordenador	Solicitacao	Central_de_Copias	0

Fonte: compilação do autor

Os mesmos filtros anteriores foram aplicados:

- a coluna 9 (Conflito) foi transformada do tipo de dado *Numérico para Nominal*. Este atributo é a classe;
- O primeiro atributo, o "nome" da política, na forma "PolicyXY", onde XY é um número inteiro sequencial, iniciando em 1 foi removido;
- Tanto as datas de ativação quanto de desativação da política foram removidas;

- O dataset, ao final, ficou com seis atributos, 5 previsores (Acesso, Organizacao, Sujeito, Acao e Objeto) e uma classe (o atributo 'Conflito').

Os resultados para os algoritmos (ainda usando a ferramenta Weka) foram os seguintes:

Para o MultiLayer Perceptron, a Tabela 2 mostra como as acurácias ficaram com os arquivos com quantidades diferentes de políticas.

Tabela 2 – Acurácia do MLP

Qtd. de Políticas	Acurácia
68	97.05
276	98.73
392	99.28

Fonte: Elaborada pelo autor mediante experimentos

Para o classificador SVM, a Tabela 3 mostra como as acurácias ficaram com os arquivos com quantidades diferentes de políticas.

Tabela 3 – Acurácia do SVM

Qtd. de Políticas	Acurácia
68	97.05
276	98.40
392	99.28

Fonte: Elaborada pelo autor mediante experimentos

Percebe-se que quanto maior o número de políticas, é crescente a acurácia do classificador e, portanto, melhor é a classificação. A Tabela 4 mostra os valores de acurácia para o classificador Random Forest.

Tabela 4 – Acurácia do Random Forest

Qtd. de Políticas	Acurácia
68	95.42
276	98.57
392	98.97

Fonte: Elaborada pelo autor mediante experimentos

3.3.1 Experimentos com Pandas, NumPy e sklearn

Outros experimentos foram realizados utilizando as bibliotecas Pandas, descrita em McKinney (2010), NumPy, pormenorizada em Oliphant (2006) e sklearn, detalhado em Pedregosa et al. (2011) além do Notebook Jupyter, caracterizado em Kluyver et al. (2016).

O notebook criado demonstra o treinamento dos 3 classificadores que tiveram melhor acurácia nos experimentos anteriores, uma rede neural Rede Neural Multicamadas usando o classificador **`MLPClassifier`** da biblioteca **`sklearn`**, o SVM com Kernel Linear e o Random Forest.

No arquivo, é demonstrado os experimentos na construção de uma arquitetura de uma rede neural com a estratégia de testes *holdout* sendo a divisão da base (split) em atributos previsores e classe com cerca de 75% da base sendo usada para treinamento da rede e 25% para teste, além de experimentos usando cross-validation com 5 folds e um teste de hipótese ao final de um processamento em batch.

Há também um pré-processamento importante focado em um tratamento dos dados categóricos do *dataset* e sua conversão para dados numéricos e divisão da base original em dois conjuntos de dados (previsores e classe).

Todos os atributos categóricos do dataset de políticas (o maior, com 392 instâncias) foram transformados para numéricos sendo um dicionário de dados construído para isso utilizando uma técnica chamada codificação, semelhante à da classe LabelEncoder da biblioteca `sklearn`. Basicamente, o processo codifica rótulos de destino com valores entre 0 e n-classes -1. Isto é importante, pois alguns algoritmos só trabalham com valores numéricos (como o MultiLayerPerceptron, o SVM entre outros).

Primeiramente a base foi importada da seguinte forma, como demonstrado na Figura 31 mostrando o aspecto inicial do dataset.

Figura 31 – Aspecto do dataset importado

```
In [4]: base.shape
Out[4]: (281, 6)

In [5]: base.head()
Out[5]:
```

	Acesso	Organizacao	Sujeito	Acao	Objeto	Conflito
0	Permitted	UFAC	Secretario_de_Curso_Academico	Abertura	Documentos	0
1	Forbidden	UFAC	Sandra_Maria_Soares_da_Rocha	Abertura	Documentos	0
2	Permitted	UFAC	Outro_Usuario	Solicitar	Produtos	0
3	Forbidden	UFAC	Secretario_de_Centros_Academicos	Solicitar	Produtos	0
4	Permitted	UFAC	Outro_Usuario	Acessar	Almoxarifado	0

Fonte: compilação do autor

A partir daí, foram selecionados da base todas as linhas dos atributos do *dataset* que estavam com os tipos *object*. Foram procurados valores nulos e não foram encontrados.

Foi utilizada uma técnica que transforma um atributo categórico com k valores em uma representação numérica com valores inteiros para cada k valor chamada *label encoder*, ou seja, ela codifica os rótulos de destino com valores entre 1 e n-classes−1. Há vantagens e desvantagens nessa abordagem. Elas estão discutidas em detalhes em Sarkar, Bali e Sharma (2017).

A Figura 32 demonstra como este procedimento foi realizado para o atributo que representa o acesso (Permitido, Proibido e Obrigatório).

Figura 32 – Engenharia de atributos - dados categóricos textuais

```
In [15]: #Usando dicionarios para trocar os valores das colunas
         acesso = {"Acesso": {"Permitted":1, "Forbidden": 2, "Obliged": 3}}
         obj_df.replace(acesso, inplace=True)
         obj_df.head()
```

Out[15]:

	Acesso	Organizacao	Sujeito	Acao	Objeto
0	1	UFAC	Secretario_de_Curso_Academico	Abertura	Documentos
1	2	UFAC	Sandra_Maria_Soares_da_Rocha	Abertura	Documentos
2	1	UFAC	Outro_Usuario	Solicitar	Produtos
3	2	UFAC	Secretario_de_Centros_Academicos	Solicitar	Produtos
4	1	UFAC	Outro_Usuario	Acessar	Almoxarifado

Fonte: compilação do autor

Uma função para realizar o encode automático foi criada. Ela está disponível em <https://bit.ly/3qoILsX>.

Em seguida a base foi dividida em atributos previsores e a classe. Os previsores são as colunas que representam a política em si e a classe é a representação binária do conflito.

Figura 33 – Aspecto dos atributos previsores

```
In [39]: previsores
```

Out[39]:

	Acesso	Organizacao	Sujeito	Acao	Objeto
0	1	5	25	13	5
1	2	5	38	13	5
2	1	5	2	4	11
3	2	5	33	4	11
4	1	5	2	8	12
...	...	...	...	...	...
276	1	2	6	1	2
277	1	8	6	1	2
278	2	2	16	8	16
279	1	2	5	1	20
280	2	2	29	9	1

281 rows × 5 columns

Fonte: compilação do autor

O aspecto dos atributos previsores ficou como o mostrado na Figura 33. Já a classe ficou com o aspecto da Figura 34.

Em seguida os atributos foram transformados usando padronização para manter as variáveis na mesma ordem de grandeza. Na padronização, a média se iguala a 0 e o desvio-padrão se mantém em 1. A fórmula da padronização é a seguinte:

$$z = \frac{x - \mu}{\sigma} \tag{3.1}$$

onde μ é a média aritmética e σ é o desvio-padrão dos dados. Como todos os dados estão agora em formato numérico, é um passo importante. O código para a padronização foi o Código 3.1:

Código 3.1 – Código da Padronização

```
10  from sklearn.preprocessing import StandardScaler
11  scaler = StandardScaler()
12  previsores_transformados = scaler.fit_transform(previsores)
```

Assim os dados foram padronizados e ficaram na mesma escala. Todos os algoritmos que foram padronizados utilizaram este mesmo algoritmo. Foi gerado, então um novo conjunto de dados de treinamento com 75% do *dataset* (sendo escolhidos randomicamente). Deixando, assim, 25% da base para validação.

Figura 34 – Aspecto do atributo classe

```
In [ ]: classe

Out[ ]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
               1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
               1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
               0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
               0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
               1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0,
               0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
               0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
               0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
               0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
               1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
               0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
               0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0])
```

Fonte: compilação do autor

Logo em seguida, um modelo de Multi-Layer Perceptron foi criado usando o classificador **`MLPClassifier`** da biblioteca **`sklearn`** com os seguintes hiperparâmetros, mostrados no Código 3.2:

Código 3.2 – Hiperparâmetros do MLPClassifier

```
    verbose = True,
    max_iter=10000,
    tol = 0.000010,
    solver = 'adam',
    hidden_layer_sizes=(32),
    shuffle=False,
    activation='relu' ,
    batch_size=40
```

Nos quais, respectivamente estão configuradas, o número máximo de épocas de treinamento (10000), a tolerância (0.000010), a função de otimização de peso ('adam' refere-se a um otimizador estocástico baseado em gradiente descendente), a quantidade de neurônios na única camada oculta, se as amostras devem ser embaralhadas em cada iteração (marcado como falso) e a função de ativação da camada oculta (função de ativação ReLU).

A Figura 35 mostra o código no notebook e as iterações finais onde o classificador indica que a função de perda no treinamento não melhorou mais do que a tolerância, $tol = 0.000010$ por 10 épocas consecutivas e, portanto, ele encerrou as iterações.

Figura 35 – Código do **MLPClassifier** com as primeiras iterações

```
In [ ]: import random
        random.seed(42)
        from sklearn.neural_network import MLPClassifier
        classificador = MLPClassifier(
                                       verbose = True,
                                       max_iter=10000,
                                       tol = 0.000010,
                                       solver = 'adam',
                                       hidden_layer_sizes=(32),
                                       shuffle=False,
                                       activation='relu',
                                       batch_size=40
                                       )
        classificador.fit(previsores_treinamento, classe_treinamento)

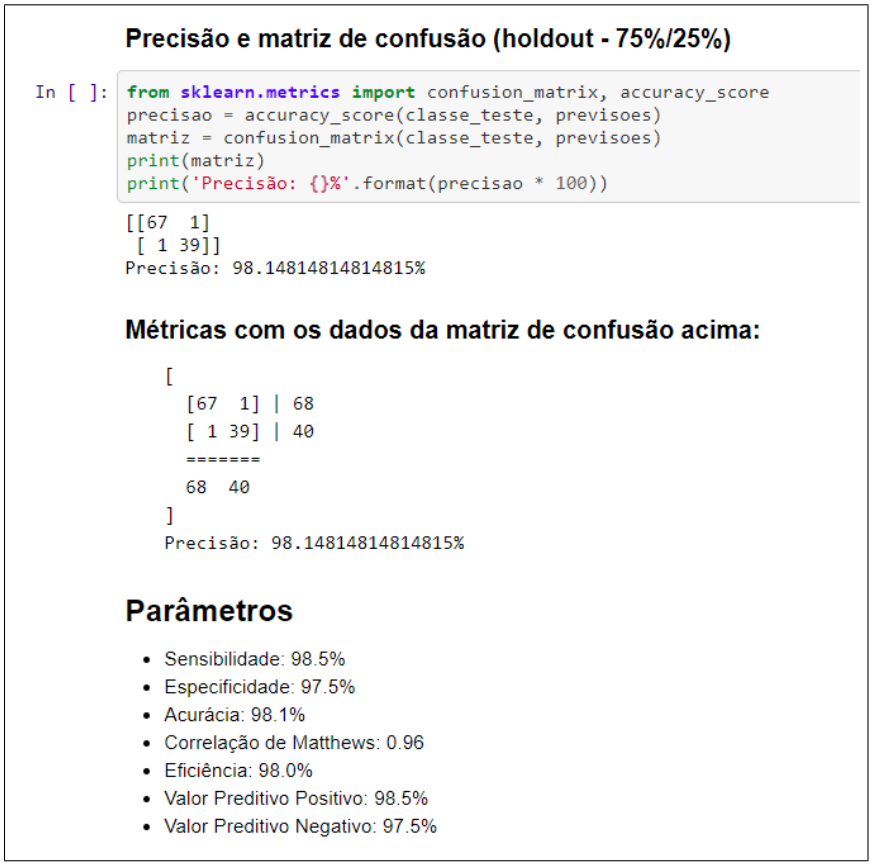
        Iteration 1, loss = 4.34947175
        Iteration 2, loss = 3.23204901
        Iteration 3, loss = 2.22792699
        Iteration 4, loss = 1.46255093
        Iteration 5, loss = 1.08155719
        Iteration 6, loss = 0.96505404
        Iteration 7, loss = 0.92027858
        Iteration 8, loss = 0.87434168
        Iteration 9, loss = 0.82498906
        Iteration 10, loss = 0.78195139
        Iteration 11, loss = 0.74830926
        Iteration 12, loss = 0.72129424
```

Fonte: compilação do autor

A Figura 36 mostra algumas métricas de validação após o modelo criado realizar as predições na base de teste (25% do dataset ou 71 políticas). Nela, pode-se perceber a acurácia do classificador em 95.77%, classificando, conforme a matriz de confusão

mostrada na mesma figura, somente 3 previsões incorretas. Estes resultados apresentam uma sensibilidade (*recall*) de 97.7%. Levando-se em conta que o modelo se aprimora conforme a quantidade de instâncias aumenta, de acordo com o pressuposto no *teorema da aproximação universal* Hagan, Demuth e Beale (1996), pode se considerar este como um modelo satisfatório.

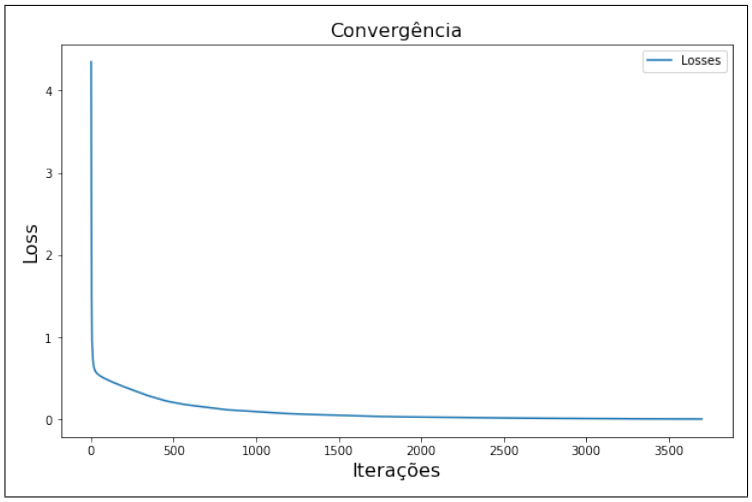
Figura 36 – Validações para o modelo **MLPClassifier**



Fonte: compilação do autor

Já a figura Figura 37 mostra a convergência da função de perda.

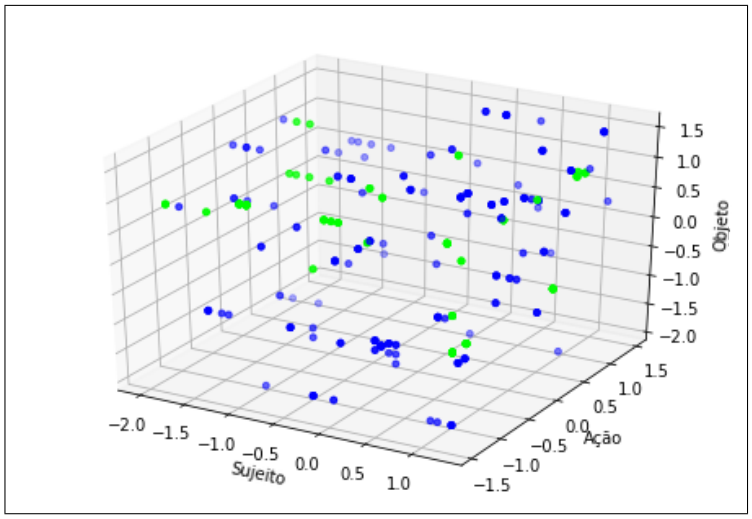
Figura 37 – Convergência da Loss Function - **MLPClassifier**



Fonte: compilação do autor

A Figura 38 mostra a dimensionalidade dos dados após transformados pelo processo de padronização e dá uma visão geral de 3 atributos, ***sujeito, ação e objeto*** e o impacto na difusão dos conflitos no espaço após os atributos categóricos terem sido, todos, transformados para uma representação numérica. A dimensionalidade da Figura 38 mostra a diferença de proporção e extensão do atributo classe além de mostrar que o problema provavelmente é linearmente separável. As fronteiras de decisão são mostradas na Figura 39 para diversos classificadores.

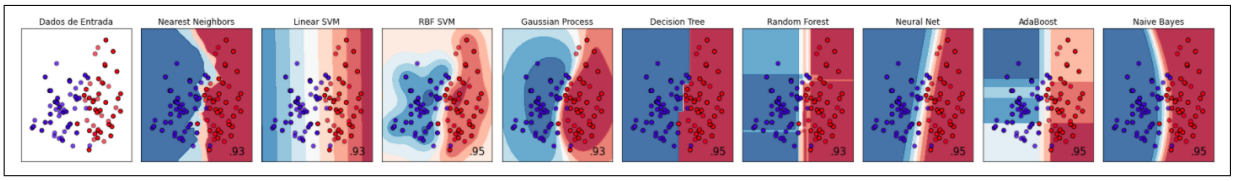
Figura 38 – Dimensionalidade dos dados: atributos sujeito, ação e objeto



Fonte: compilação do autor

Foi criado um script que, com parâmetros padrão dos classificadores, roda um experimento sobre a base para verificar a fronteira de decisão dos dados em relação a diferentes classificadores com seus parâmetros padrão. O script está disponível em <https://bit.ly/3bFJaTM>. O resultado pode ser visualizado na Figura 39.

Figura 39 – Comparação visual entre alguns classificadores



Fonte: compilação do autor

O notebook com todos os experimentos restantes (Naïve Bayes, Keras, K-NN, SVM, Random Forest e Árvores de Decisão) pode ser visualizado em <https://bit.ly/3nJHYky>. Na Tabela 5 com as acurácias destes classificadores conforme ficou demonstrada nos experimentos.

Tabela 5 – Acurácia dos classificadores - dataset 392 políticas

Classificador/Algoritmo	Acurácia
MLPCLassifier	0.9900
SVM kernel linear	0.9900
Random Forest	0.9814
SVM RBF	0.9814
Keras (MLP)	0.9629
k-NN (com k=5	0.8796
Árvore de Decisão	0.8055
Naïve Bayes	0.7314

Fonte: Elaborada pelo autor mediante experimentos

3.3.2 Experimentos com Redes Neurais usando TensorFlow e Pytorch

Foi realizado, assim, no escopo deste trabalho experimentos com a base de dados de com 281 políticas já citada anteriormente. O notebook completo está disponível online no endereço: <https://bit.ly/33BmCzt>. Foi todo construído no ambiente Google Collab que é um serviço de nuvem gratuito hospedado pelo Google para incentivar a pesquisa de Aprendizado de Máquina e Inteligência Artificial, similar ao Jupyter Notebook, é uma lista de células que podem conter textos explicativos ou códigos executáveis e suas saídas (SANTOS, 2020).

Neste modelo os hiperparâmetros principais, cf. Thenmozhi, Kalaivani e Aravindan (2018) foram ajustados como: tamanho do batch em 20 (analisa, computa e reajusta os pesos de 20 instâncias de cada vez, por época), o número de GPU's trabalhando em conjunto para 4, a taxa de aprendizado em 0.00001, o decaimento dos pesos em 0.000005 e o número de épocas padrão em 30 (apenas para testes inciais).

Figura 40 – Separação dos dados de teste e treino



```
[78]  1 torch.manual_seed(1)
      2 indices = torch.randperm(len(dados_com_rotulos)).tolist()   # a
      3
      4 #separando os dados de teste e treino
      5 train_size = int(0.7*len(dados_com_rotulos))
      6 df_train   = dados_com_rotulos.iloc[indices[:train_size]]
      7 df_test    = dados_com_rotulos.iloc[indices[train_size:]]
      8
      9 print('Treino : ', len(df_train), ' - Teste: ', len(df_test))
     10 display(df_test.head())
     11
     12 df_train.to_csv('politicas_train.csv', index=False)
     13 df_test.to_csv('politicas_test.csv', index=False)
     14
     15 !ls

    Treino :  196  - Teste:  85
```

Fonte: compilação do autor

Para o processo de validação, usou-se o *holdout* com a separação em 75% das instâncias para o treinamento da rede neural e 25% para teste, predição e validação. Na figura 40 pode-se visualizar como o processo foi realizado,inclusive com a quantidade de instâncias em cada conjunto de dados (na linha 2 é feita a randomização do *dataset* original para evitar o overfitting e balancear a probabilidade da distribuição).

O pacote `torch.util.data` do PyTorch possui a classe abstrata `Dataset`. Ela permite que seja implementado o próprio dataset reescrevendo os métodos:

- `__init__(self)`: Define a lista de amostras do dataset
- `__getitem__(self, idx)`: Carrega uma amostra, aplica as devidas transformações e retorna uma tupla (dado, rótulo)
- `__len__(self)`: Retorna a quantidade de amostras do dataset

Dessa forma, a Figura 41 mostra a elaboração de uma classe chamada `Politicas` que implementa uma classe-filha que herda da superclasse, *Dataset* descrita acima.

Em seguida, dois objetos DataLoader são criados, um para a base de treinamento e um para a base de teste. Em seguida, a Rede Neural Multicamadas é instanciada mediante a criação de uma classe chamada MLP que herda da classe `nn.Module` que representa um módulo genérico de uma rede neural.

A arquitetura da rede é configurada dentro da classe que a cria sendo: uma *camada linear* de entrada, duas *camadas lineares ocultas com 32 neurônios* em cada camada usando a função de ativação ReLU e *uma camada linear de saída com dois neurônios*, representando os dois rótulos do atributo que é a classe, neste caso específico, o atributo binário **conflito** que será predito.

Figura 41 – Implementação da classe Politicas

```
class Politicas(Dataset):
  def __init__(self, csv_path):
    self.dados = pd.read_csv(csv_path).to_numpy()

  def __getitem__(self, idx):

    sample = self.dados[idx][0:5]    # [2:14] são as colunas do dataset
    label  = self.dados[idx][-1:]     # [-1:]

    # converte pra tensor
    sample = torch.from_numpy(sample.astype(np.float32))
    label  = torch.from_numpy(label.astype(np.float32))

    return sample, label

  def __len__(self):
    return len(self.dados)
```

Fonte: compilação do autor

É criada, no mesmo código da classe MLP, uma função que faz o avanço (fed-forward) das computações na rede e, ao final, é instanciada uma variável chamada **`net`** com as variáveis descritas:

- 5 atributos/neurônios na camada linear de entrada;
- 2 camadas ocultas com 32 neurônios cada; e
- 2 camadas de saída representando as variáveis preditas

Na mesma linha que cria o objeto `net` é feito o *cast* da rede na GPU para que ela possa, ao ser treinada, fazer uso dos poderes computacionais em paralelo da API, CUDA. A Figura 42 mostra o código descrito aqui.

Em seguida é definida uma *loss function*(função de perda ou de custo). É criado um critério que mede o erro médio quadrático (norma matricial ao quadrado) entre cada elemento na entrada x e o destino y, cf. Equação 2.9 da página página 47.

O otimizador utilizado é o Adam, um algoritmo para otimização estocástica descrito em Kingma e Ba (2017) passando para o algoritmo os valores da taxa de aprendizado e do decaimento de pesos mostrado anteriormente nesta seção.

Figura 42 – Implementação da classe que modela a arquitetura da rede

```
class MLP(nn.Module):

  def __init__(self, input_size, hidden_size, out_size):
    super(MLP, self).__init__()

    self.features  = nn.Sequential(
                       nn.Linear(input_size, hidden_size),
                       nn.ReLU(),
                       nn.Linear(hidden_size, hidden_size),
                       nn.ReLU()
                     )
    self.out     = nn.Linear(hidden_size, out_size)

  def forward(self, X):

    feature = self.features(X)
    output  = self.out(feature)

    return output

input_size  = len(train_set[0][0])    # quantidade de atributos *importantes*
hidden_size = 32
out_size    = 2                       # variaveis que serão preditas

net = MLP(input_size, hidden_size, out_size).to(args['device']) #cast na GPU
```

Fonte: compilação do autor

O fluxo de treinamento desta arquitetura de rede neural multicamadas proposta neste experimento segue o algoritmo iterativo:

- Iterar nas épocas
- Iterar nos batches (a quantidade de instâncias simultâneas)
- Cast dos dados no dispositivo de hardware (GPU)
- Forward na rede e cálculo da *loss function*
- Cálculo do gradiente e atualização dos pesos

Figura 43 – Implementação da função de treino da rede

```
def train(train_loader, net, epoch):
  net.train()

  epoch_loss = []
  for batch in train_loader:

    dado, rotulo = batch

    # Cast na GPU
    dado   = dado.to(args['device'])
    rotulo = rotulo.to(args['device'])

    # Forward
    pred = net(dado)
    loss = criterion(pred, rotulo)
    epoch_loss.append(loss.cpu().data)

    # Backward
    loss.backward()
    optimizer.step()

  epoch_loss = np.asarray(epoch_loss)

  print("Epoca %d, Loss: %.4f +\- %.4f" % (epoch, epoch_loss.mean(), epoch_loss.std()) )

  return epoch_loss.mean()
```

Fonte: compilação do autor

Esse conjunto de passos é responsável pelo processo iterativo de otimização de uma rede. A validação, entretanto, é apenas a aplicação da rede em dados nunca antes vistos para estimar a qualidade do modelo no mundo real.

Três funções, portanto, são criadas, uma para modelar o treinamento da rede neural, uma para o teste e validação e outra que agrega as duas primeiras em uma só para executar o algoritmo descrito anteriormente. Então, para finalizar o experimento com o TensorFlow, o PyTorch, CUDA e o Google Collaboratory, *foi realizado um treinamento com 500 épocas da rede neural* explanada nesta seção e os resultados tanto do treino quando do teste e validação foram armazenados. A Figura 43 mostra a função de treino da rede.

A Figura 44 mostra a função de teste da rede (fase de aprendizado).

Figura 44 – Implementação da função de teste da rede

```
def test(test_loader, net, epoch):

  net.eval()
  with torch.no_grad():
    epoch_loss = []
    for batch in test_loader:

      dado, rotulo = batch

      # Cast na GPU
      dado   = dado.to(args['device'])
      rotulo = rotulo.to(args['device'])

      # Forward
      pred = net(dado)
      loss = criterion(pred, rotulo)
      epoch_loss.append(loss.cpu().data)

    epoch_loss = np.asarray(epoch_loss)

    print('********** Validate **********')
    print("Epoca %d, Loss: %.4f +\- %.4f" % (epoch, epoch_loss.mean(), epoch_loss.std()) )

    return epoch_loss.mean()
```

Fonte: compilação do autor

E a Figura 45 exibe a função que mescla o treino e o teste em uma só.

Figura 45 – Implementação da função que mescla o treino e o teste

```
def forward(loader, net, epoch, mode):
  if mode == "train":
    net.train()
  else:
    net.eval()

  epoch_loss = []
  for batch in loader:
    dado, rotulo = batch

    # Cast na GPU
    dado   = dado.to(args['device'])
    rotulo = rotulo.to(args['device'])

    # Forward
    pred = net(dado)
    loss = criterion(pred, rotulo)
    epoch_loss.append(loss.cpu().data)

    if mode == "train":
      # Backward
      loss.backward()
      optimizer.step()

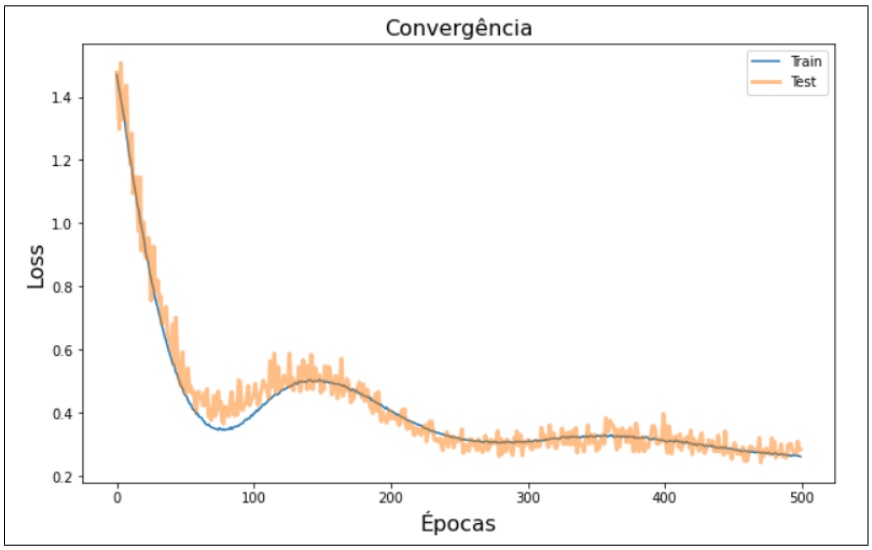
  epoch_loss = np.asarray(epoch_loss)
  print("Epoca %d, Loss: %.4f +\- %.4f" % (epoch, epoch_loss.mean(), epoch_loss.std()) )
```

Fonte: compilação do autor

Como esclarecimento e interpretação visual foi construída, então, a Figura 46 com

os dados de armazenados de treino e teste (média da *loss function* ou função de custo de cada iteração dentro da época) que mostra um comparativo das épocas de teste e treino da rede neural e a convergência de ambas, retratando a acurácia e a validade do modelo deste experimento.

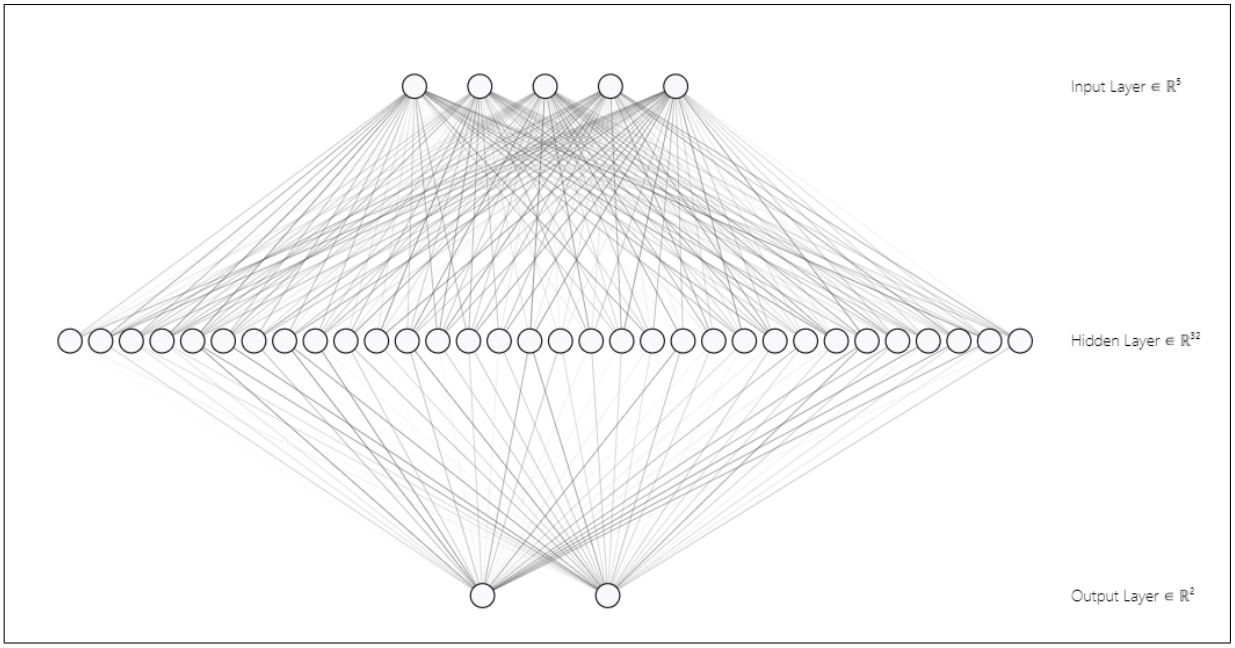
Figura 46 – Convergência das épocas entre o treino e o teste da MLP



Fonte: compilação do autor

A arquitetura da rede neural construída e mostrada na Figura 42 pode ser visualizada na Figura 47 abaixo:

Figura 47 – Arquitetura da rede neural



Fonte: compilação do autor

3.3.3 Cross-validation, OneHotEncoder e Testes Estatísticos

Nos experimentos anteriores foi usada a técnica chamada de LabelEncoder para lidar com as variáveis categóricas. Este procedimento foi mostrado na subseção 3.3.1, exemplificado na Figura 32 e na Figura 33. Esta técnica tem alguns inconvenientes em relação aos dados categóricos, principalmente quando estes não são ordinais (SHEN; SHAFIQ, 2019).

Para sanar essas limitações há a técnica chamada de esquema de codificação *one-hot*, também conhecida como *'one-of-K'* ou *'dummy'* que cria uma coluna binária para cada categoria e retorna uma matriz esparsa ou matriz densa, dependendo do parâmetro esparso. Cada valor único na categoria será adicionado como um atributo novo. Essa codificação é necessária para alimentar dados categóricos para muitos estimadores do scikit-learn, principalmente modelos lineares e SVMs com os kernels padrão. (PEDREGOSA et al., 2011) (SHEN; SHAFIQ, 2019).

Os próximos experimentos usam essa técnica para codificar o dataset e utilizar os dados nos algoritmos selecionados nos dois primeiros experimentos.

Um experimento **usando cross-validation com 5 folds foi executado com os 3 algoritmos que tiveram maior acurácia nos dois primeiros experimentos** , referenciado em subseção 3.2.3 e na seção 3.3, mas desta vez usando ainda o dataset com 392 políticas. O notebook do experimento está disponível em <https://bit.ly/3fIsnAV>. A matriz de confusão e a precisão do **MLPClassifier**, que implementa uma rede neural multicamadas, cf. Pedregosa et al. (2011) e Crabbé et al. (2020), são exibidas na Figura 48

Figura 48 – MLPClassifier - resultado dataset 392 políticas

```
[89]  1 #acurácia
      2 precisao_cross = accuracy_score(classe_treinamento, pred)
      3 matriz_cross = confusion_matrix(classe_treinamento, pred)
      4 print(matriz_cross)
      5 print('Precisão: {}%'.format(precisao_cross * 100))

     [[188   5]
      [  0 101]]
     Precisão: 98.29931972789116%
```

Fonte: compilação do autor

Algumas métricas importantes deste modelo, MLPClassifier com os hiperparâmetros da Figura 49 foram o **recall** que ficou em 100.0%, a **acurácia** em 98.2%, a **correlação de Mathews** em 0.96 e a **eficiência** em 97.0%.

Já para o classificador SVM, com kernel linear, a matriz de confusão e a precisão do **SVC** do scikit-learn cf. Pedregosa et al. (2011) são exibidas na Figura 50.

Figura 49 – MLPClassifier - hiperparâmetros do modelo

```
classificador = MLPClassifier(verbose = True,
                              max_iter=10000,
                              tol = 0.000010,
                              solver = 'adam',
                              hidden_layer_sizes=(100),
                              batch_size=10,
                              learning_rate='constant',
                              learning_rate_init=0.0001,
                              activation='relu')
```

Fonte: compilação do autor

Figura 50 – SVM kernel linear - resultado dataset 392 políticas

```
[111]  1 print(matriz_cross)
       2 print('Precisão: {}%'.format(precisao_cross * 100))

     [[190   3]
      [  1 100]]
     Precisão: 98.63945578231292%
```

Fonte: compilação do autor

Algumas métricas deste modelo SVM com kernel linear com os hiperparâmetros da Figura 51 foram o **recall** que ficou em 99.5%, a **acurácia** em 98.6%, a **correlação de Mathews** em 0.97 e a **eficiência** em 98.3%.

Figura 51 – SVM kernel linear - hiperparâmetros do modelo

```
[102]  1 from sklearn.svm import SVC

[103]  1 svm_conflito = SVC(kernel='linear', random_state=1, C = 4.0, verbose=True)
```

Fonte: compilação do autor

E para finalizar, com o classificador Random Forest a matriz de confusão e a precisão do **RandomForestClassifier** do scikit-learn descrito em Pedregosa et al. (2011) são exibidas na Figura 52.

Já as métricas deste modelo Random Forest com os hiperparâmetros da Figura 53 foram o **recall** que ficou em 97.7%, a **acurácia** em 97.2%, a **correlação de Mathews** em 0.94 e a **eficiência** em 97.0%.

Figura 52 – Random Forest - resultado dataset 392 políticas

```
[116]  1 print(matriz_cross)
       2 print('Precisão: {}%'.format(precisao_cross * 100))

      [[251   5]
       [  6 130]]
      Precisão: 97.1938775510204%
```

Fonte: compilação do autor

Figura 53 – Random Forest - resultado dataset 392 políticas

```
[112]  1 from sklearn.ensemble import RandomForestClassifier
       2 classificador = RandomForestClassifier(n_estimators=40, criterion='entropy', random_state=0)
```

Fonte: compilação do autor

3.3.3.1 Tuning dos parâmetros

Hiperparâmetros são importantes pois deles dependem a eficiência do modelo. (KUMAR; JAIN, 2020, pp. 31-48). Neste experimento foi usada uma função de *tuning*, a partir do módulo *GridSearchCV* disponível com a biblioteca scikit-learn Pedregosa et al. (2011) que é uma ferramenta utilizada para automatizar o processo de ajuste dos hiperparâmetros de um algoritmo já que executa de forma sistemática diversas combinações dos parâmetros avaliando os melhores. Os resultados do ajuste de alguns dos hiperparâmetros importantes encontrados pela função automática de *tuning* foram os seguintes:

1. Para o algoritmo Multi-Layer Perceptron (MLPClassifier):
   - **activation** (Função de ativação): **relu**;
   - **batch_size** (tamanho dos minibatches): **10**;
   - **solver** (solver para otimização dos pesos): **adam**;
   - **max_iter** (máximo de iterações em cada época): **10000**;
   - **tol** (Tolerância para a otimização): **0.000010**.
2. Para o algoritmo SVM, kernel linear:
   - **C** (paramêtro de regularização/custo): **4**;
   - **kernel** (tipo de kernel): **linear**;
   - **tol** (tolerância do critério de parada):**0.001**.

3. Para o algoritmo Random Forest:

   - **criterion** (qualidade da divisão): **gini** (impureza de gini);
   - **min_samples_leaf** (mínimo de amostras em um nó-folha): **1**;
   - **min_samples_split** (mínimo de amostras para dividir um nó):**2**;
   - **n_estimators** (número de árvores na floresta):**40**.

Estes valores serão utilizados na subseção 3.3.3.2 em que testes estatísticos serão realizados para a descoberta de qual é o melhor algoritmo para a detecção de conflitos de modalidade entre políticas de controle de acesso.

3.3.3.2 Testes Estatísticos e descoberta do melhor algoritmo

A descoberta do melhor algoritmo para a detecção de conflitos de modalidade, se deu através da execução dos 3 algoritmos em um processo de batch em um total de 30 vezes para cada um deles. Para avaliar o desempenho definiu-se o método cross-validation com 5 folds para cada uma das 30 execuções alterando-se o *random state* a cada execução, para assegurar que os valores de cada *fold* sejam diferentes a cada performance, totalizando, portanto, 150 execuções para cada um dos algoritmos, usando os hiperparâmetros descobertos pela função de *tuning* da subseção 3.3.3.1 cf. se observa na Figura 54.

As acurácias foram armazenadas em uma lista, em seguida, diversos cálculos estatísticos como as médias aritméticas, médias quadráticas, variâncias, desvio padrão e outras estatísticas foram calculadas para determinar o algoritmo com as melhores acurácias médias e melhores performances na simulação executada. O notebook deste experimento está disponível para consulta em <https://bit.ly/3fIsnAV>. O coeficiente de variação das acurácias dos 150 testes realizados na execução do experimento com os algoritmos é mostrado na Tabela 6. São apresentados também os valores máximo e mínimo (ou seja, a amplitude) das acurácias de todas as 150 execuções:

Tabela 6 – Coeficiente de variação e valores mínimos/máximos das execuções

Classificador/Algoritmo	Coef. de Variação (%)	Valores min-máx
Random Forest	0.499	0.9719-0.9923
SVM kernel Linear	0.386	0.9795-0.9923
MLPClassifier	0.362	0.9796-0.9949

Fonte: Elaborada pelo autor mediante experimentos

Na Tabela 6 verifica-se, portanto, que a execução de **menor acurácia** do algoritmo Random Forest foi 0.9719, ou seja, 97.19% e a de **maior acurácia** foi de 0.9923, ou seja

Figura 54 – Batch - código das 150 execuções distintas por algoritmo

```
[ ]  resultados_random_forest = []
     resultados_svm = []
     resultados_rede_neural = []

     for i in range(30): #quantidade de testes
       print(i)
       kfold = KFold(n_splits=5, shuffle=True, random_state=i)

       random_forest = RandomForestClassifier(criterion = 'gini',
                                              min_samples_leaf = 1,
                                              min_samples_split=2,
                                              n_estimators = 40)
       scores = cross_val_score(random_forest, previsores_onehot, classe_onehot,
                                cv = kfold)
       resultados_random_forest.append(scores.mean())

       svm = SVC(kernel = 'linear', C = 4.0, tol=0.001)
       scores = cross_val_score(svm, previsores_onehot, classe_onehot, cv = kfold)
       resultados_svm.append(scores.mean())

       rede_neural = MLPClassifier(activation = 'relu', batch_size = 10,
                                   solver = 'adam',
                                   tol=0.000010)
       scores = cross_val_score(rede_neural, previsores_onehot, classe_onehot,
                                cv = kfold)
       resultados_rede_neural.append(scores.mean())
```

Fonte: compilação do autor

99.23% — e este foi o algoritmo que teve a *menor* acurácia nesta simulação. Todos os 150 experimentos rodados em batch mostram que as acurácias dos 3 algoritmos ficaram entre 97.19% e 99.49%. Como os valores das acurácias ficaram próximos dos 3 algoritmos nas execuções em batch, a Tabela 7 mostra a média e o desvio-padrão das acurácias de cada algoritmo.

Tabela 7 – Média e desvio-padrão da distribuição das acurácias por algoritmo

Algoritmo	Random Forest	SVM kernel Linear	MLPClassifier
Média	0.9824	0.9863	0.9904
Desvio-padrão	0.0049	0.0038	0.0035

Fonte: Elaborada pelo autor mediante experimentos

A Tabela 7 mostra que o algoritmo que teve a menor variação entre as execuções, ou seja, o menor desvio-padrão é o MLPClassifier que implementa uma rede neural multicamadas. A Tabela 8 mostra a média das 150 execuções de cada algoritmo sendo assim um retrato do desempenho destes classificadores já que a variação entre as execuções

é baixa, como a Tabela 7 mostra. Especificamente a melhor das execuções do MLP Classifier (MultiLayer Perceptron) apresentou recall de 100%, eficiência de 99,5% sendo que o coeficiente de correlação de Mathews (ϕ) de 0,99 demonstra que há eficiência entre as predições observadas e as previstas.

Tabela 8 – Médias da distribuição dos resultados dos classificadores

Classificador/Algoritmo	Média das acurácias
Multi Layer Perceptron	0.9904
SVM kernel linear	0.9863
Random Forest	0.9824

Fonte: Elaborada pelo autor mediante experimentos

O teste estatístico de normalidade de Shapiro-Wilk cf. Razali e Yap (2011) foi, então, executado com $\alpha = 0.05$ (confiança de 95% e nível de significância, *alfa* de 5%) e os resultados mostram um *p-value* de 0.6640, 0.008 e 0.0059 respectivamente para os algoritmos Random Forest, SVM e MLPClassifier indicando que somente a execução do Random Forest seguiu uma distribuição normal nos experimentos realizados já que foi o único que teve um p-value maior que o alpha de 0.05.

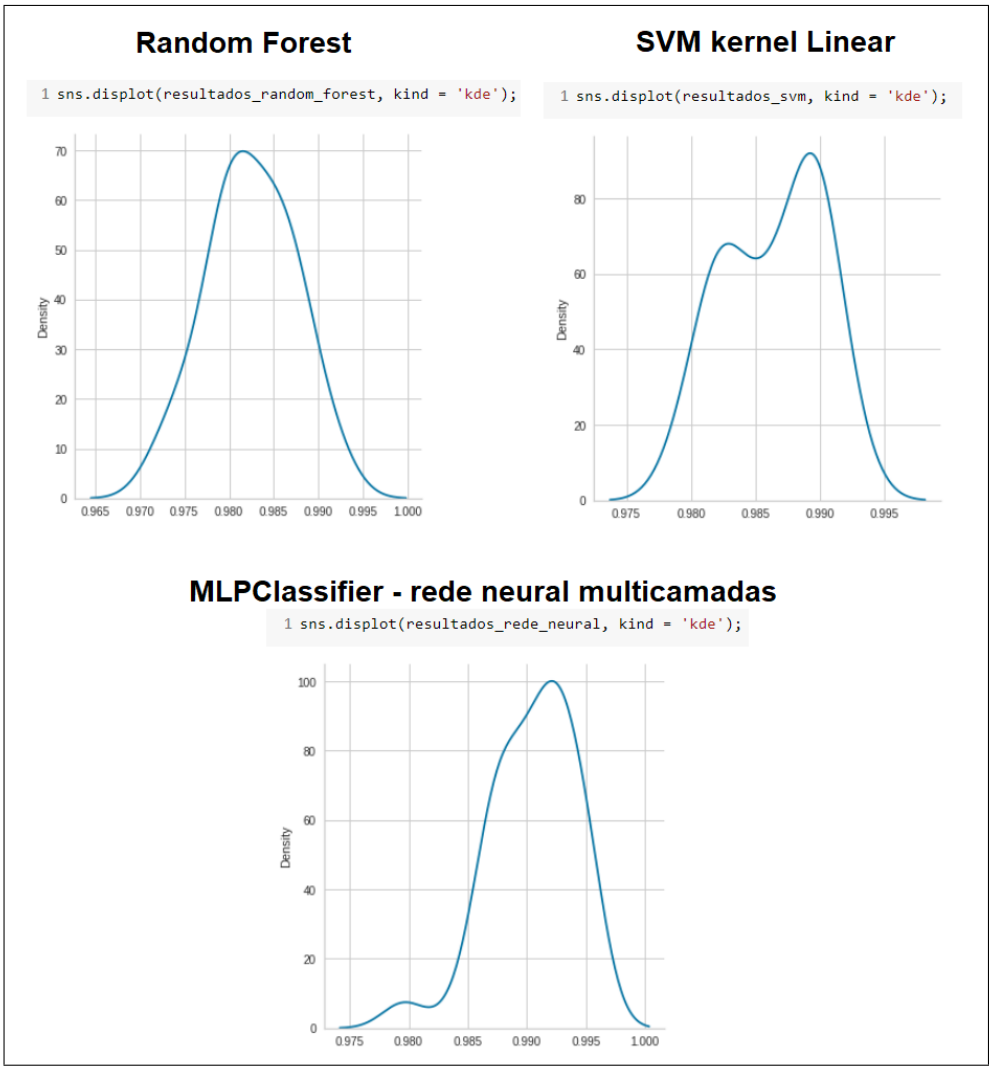
A hipótese nula, neste caso, indica que os resultados dos algoritmos estão na distribuição normal e a hipótese alternativa indica que os dados não estão em uma distribuição normal, algo que pode ser facilmente verificado observando-se as curvas das distribuições das acurácias de cada classificador que estão na Figura 55.

Para comparar as distribuições das acurácias dos classificadores gerada na simulação e, para determinar qual é, de fato, *o melhor algoritmo* no experimento executado levando em consideração como determinante a métrica "acurácia", foi executada, então, um **análise ANOVA** e um **teste de Tukey** cf. Yandell (1997) e Driscoll (1996).

O teste de Tukey compara todos os possíveis pares de médias e se baseia na diferença mínima significativa. A hipótese-nula, neste teste específico, é os algoritmos não terem diferença significativa entre eles — caso em que não faria diferença na detecção de conflitos usar um ou outro. É um teste exato em que para o grupo de todos os $c = \frac{1}{2}k(k-1)$ comparações duas a duas, a taxa de erro familiar[3] é exatamente α (alfa) e o intervalo de confiança do teste é $1 - \alpha$. Os resultados o teste estatístico estão demonstrados na Figura 56.

[3] (FWER) é a probabilidade de fazer uma ou mais descobertas falsas ou erros do tipo I ao realizar vários testes de hipóteses (DRISCOLL, 1996).

Figura 55 – Curvas de distribuição das acurácias dos algoritmos



Fonte: compilação do autor

Figura 56 – Resultados do teste de Tukey

```
[158]  1 compara_algoritmos = MultiComparison(resultados_df['accuracy'],
       2                                      resultados_df['algoritmo'])

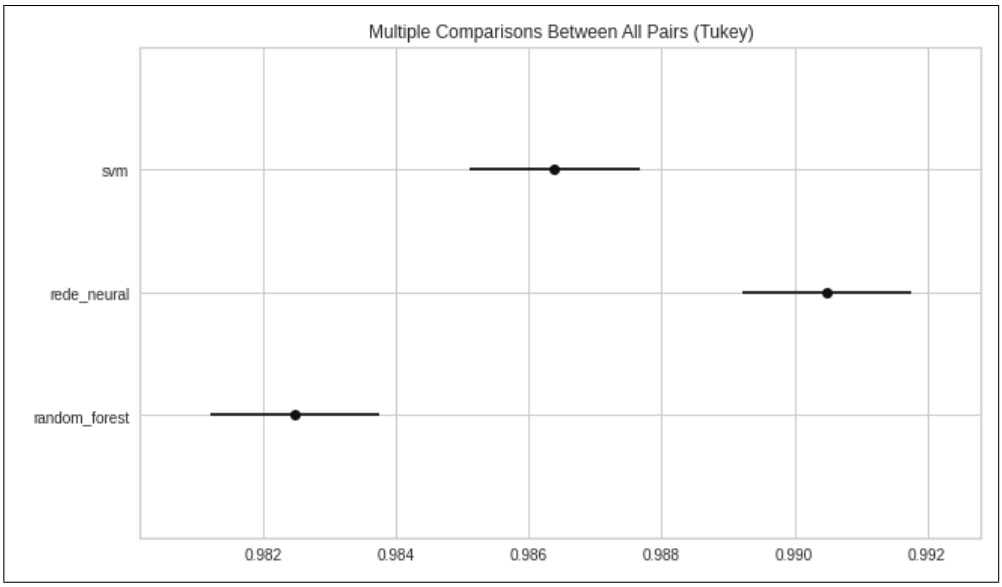
[159]  1 teste_estatistico = compara_algoritmos.tukeyhsd()
       2 print(teste_estatistico)

     Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================================
    group1       group2   meandiff p-adj   lower   upper  reject
---------------------------------------------------------------------
random_forest rede_neural    0.008  0.001  0.0054  0.0105   True
random_forest         svm   0.0039 0.0013  0.0014  0.0065   True
  rede_neural         svm  -0.0041  0.001 -0.0066 -0.0015   True
---------------------------------------------------------------------
```

Fonte: compilação do autor.

O teste estatístico na Figura 56 mostra que existe diferença significativa entre os algoritmos pela rejeição da hipótese-nula[4]) e cf. a Tabela 7 e pela Tabela 8 podemos concluir que o algoritmo MLPClassifier (Multi Layer Perceptron), que implementa uma rede neural, é estatisticamente superior aos outros dois quando se leva em conta a acurácia. Isto também é demonstrado na Figura 57 que mostra uma comparação visual dos classificadores e apresenta aquele que oferece a melhor classificação nas simulações realizadas.

Figura 57 – Comparação visual do resultado do teste estatístico dos classificadores



Fonte: compilação do autor.

3.4 Experimento com novas instâncias

Após a análise estatística da seção anterior em que foi possível, baseado nos experimentos, encontrar o melhor algoritmo para a detecção dos conflitos mediante o *dataset* utilizado e com o intuito de demonstrar que a hipótese deste trabalho é corroborada pelos experimentos empíricos na construção de modelos que possibilitem a detecção de conflitos de modalidade (os diretos), um teste com instâncias novas e desconhecidas pelo modelo foi realizado. Para isso foram criados 3 modelos de Aprendizagem de Máquina com os hiperparâmetros utilizados e discutidos na seção anterior.

[4] No teste executado é mostrada a rejeição da hipótese-nula que indica os algoritmos não terem diferença significativa entre si. Há uma comparação entre os algoritmos contendo dados relevantes e alguns cálculos post-hoc, incluindo o valor p ajustado (***p-adj***), a média entre os grupos (***meandiff***) entre outras métricas.

Figura 58 – Instâncias geradas em formato dataframe

	Acesso	Organizacao	Sujeito	Acao	Objeto
0	Obliged	UFAC	Secretario_de_Curso_Academico	Acessar	Produtos
1	Permitted	UFAC	Secretario_de_Curso_Academico	Acessar	Documentos
2	Obliged	CCSJA	Outro_Usuario	Abertura	Almoxarifado
3	Permitted	Uninorte	Coordenador	Solicitacao	Central_de_Copias
4	Forbidden	IFAC	Grupo_Almoxarifado	Inserir	Guia_de_Requisicao
5	Forbidden	UFAC	Grupo_Almoxarifado	Inserir	Guia_de_Requisicao
6	Permitted	IFAC	Estagiario3	Acess	Process
7	Permitted	Administrative_Unit	Secretarios	Create	ProcessDispatch
8	Permitted	Unidade_Administrativa	Professor	Solicitacao	Central_de_Copias
9	Permitted	Unidade_Administrativa	Professor	Acess	Central_de_Copias
10	Permitted	UFAC	Outro_Usuario	Acess	Process
11	Permitted	IFAC	Usuario_Qualquer	Alterar	Convencao

Fonte: compilação do autor

Todos os modelos para este teste foram gerados na ferramenta Weka, usando o mesmo dataset anterior (com 392 instâncias usado nos testes estatísticos) e estão disponíveis, junto com o arquivo com as novas instâncias em <https://bit.ly/3hYrRjH>.

Este arquivo, baseado no modelo anterior, também foi gerado pelo software escrito em Python, chamado **`geracao_politicas.py`** e disponibilizado em <https://bit.ly/3qkJKKs>.

A Figura 58 mostra, respectivamente, as instâncias novas geradas pelo programa **`geracao_politicas.py`** em um formato de um DataFrame Pandas, biblioteca de mineração de dados descrita em McKinney (2010).

Além da ferramenta Weka, as demais ferramentas utilizadas neste último experimento foram a bibliotca NumPy, pormenorizada em Oliphant (2006) e sklearn, detalhado em Pedregosa et al. (2011) além do Notebook Jupyter, caracterizado em Kluyver et al. (2016).

Ambos arquivos, *sem nenhuma alteração ou pré-processamento*, foram salvos com o formato do Weka (ARFF) para uso efetivo neste ferramenta. Assim, 12 novas instâncias foram geradas, cf. Figura 58, apenas para fins de teste e o resultado, usando o modelo MultiLayer Perceptron com os hiperparâmetros encontrados na função de Tuning, podem ser observados na Figura 59.

Após a simulação ter sido realizada, os modelos construídos e as previsões processadas foi usada uma classe para as políticas escrita em Python (disponível em: <https://bit.ly/3wWj9qH>) que possui um método para mostrar exatamente em qual política do dataset que surge o conflito detectado.

Na Figura 60 verifica-se o conflito na tentativa de inserção da política e em qual

Figura 59 – Resultado da predição

```
=== Predictions on user test set ===

    inst#     actual  predicted error prediction
        1        1:0        1:0       0.999
        2        1:0        1:0       1
        3        1:0        1:0       1
        4        1:0        1:0       1
        5        1:0        1:0       0.997
        6        1:0        1:0       1
        7        2:1        2:1       1
        8        2:1        1:0   +   1
        9        1:0        1:0       1
       10        1:0        1:0       1
       11        2:1        2:1       0.951
       12        2:1        2:1       1
```

Fonte: compilação do autor

instância específica o conflito ocorre.

Figura 60 – Resultado 1 da detecção de conflito

```
Conflitos!
        Acesso Organizacao      Sujeito   Acao   Objeto
33   Forbidden        IFAC  Estagiario3  Acess  Process
116  Forbidden        IFAC  Estagiario3  Acess  Process

Política: [Permitted, IFAC, Estagiario3, Acess, Process]
```

Fonte: compilação do autor

Na Figura 61, observa-se o segundo conflito detectado e qual a instância conflitante

Figura 61 – Resultado 2 da detecção de conflito

```
Conflitos!
        Acesso Organizacao        Sujeito   Acao   Objeto
1    Forbidden        UFAC  Outro_Usuario  Acess  Process
217  Forbidden        UFAC  Outro_Usuario  Acess  Process

Política: [Permitted, UFAC, Outro_Usuario, Acess, Process]
```

Fonte: compilação do autor

Por último, na Figura 62 verifica-se quais instâncias estão em conflito com a política gerada.

Assim, com uma precisão de 100% para a detecção de quando há conflito de modalidade, ou seja, direto, (indicando a instância onde ocorre — neste exemplo, as

Figura 62 – Resultado 3 da detecção de conflito

```
Conflitos!
       Acesso Organizacao           Sujeito     Acao     Objeto
10  Forbidden        IFAC  Usuario_Qualquer  Alterar  Convencao

Política: [Permitted, IFAC, Usuario_Qualquer, Alterar, Convencao]
```

Fonte: compilação do autor

instâncias 7, 11 e 12), um *recall* de 100% quando não há conflito e um *f-measure* de 94,1% quando não existe conflito e uma acurácia geral de 91,7%, e um erro quadrado médio de 0,2, considerado baixo, pode-se afirmar, levando-se em consideração que, em aplicações reais, poucas políticas são inseridas e avaliadas por vez, cf. Fugini e Bellettini (2004), portanto que usando o modelo construído dos experimentos, é possível detectar conflitos usando aprendizagem de máquina seguindo os hiparâmetros indicados neste trabalho.

3.5 Uma abordagem para os conflitos indiretos

Embora não seja o foco desta dissertação, uma maneira de tratar os conflitos indiretos, explicitados na subseção 1.1.1.1 é mostrado nesta seção. Os conflitos indiretos como são chamados, nesta dissertação, os conflitos de aplicação e os de modalidade com ações diferentes mas relacionadas, em objetos/sujeitos/organizações diferentes mas relacionados, segundo Cuppens, Cuppens-Boulahia e Ghorbel (2007) e Lupu e Sloman (1999) e mesmo quando os conflitos de modalidade não tem especificamente modalidades contraditórias ou contrárias mas seus relacionamentos o são. Para Sarkis (2017) os estudos da detecção de conflitos, mesmo os indiretos, pouco exploram os conflitos que ocorrem entre ações, entre eles, os relacionamentos de ortogonalidade e dependência entre ações. Assim como Kamoda, Yamaoka e Matsuda (2005) e Sarkis (2017), uma proposta de detecção de conflitos indiretos é utilizar uma estrutura de semântica de papel/sujeito e a propagação de políticas. Uma política de propagação define como uma política de autorização se propaga de acordo com a ordem parcial definida em uma semântica ou ontologia (KAMODA; YAMAOKA; MATSUDA, 2005, p. 4).

Usando os conceitos de Kamoda, Yamaoka e Matsuda (2005) um exemplo foi construído para ilustrar como esta detecção de conflitos indiretos pode ser realizada usando o que já foi construído pelos modelos de aprendizagem de máquina para os conflitos diretos. A estratégia é propagar a política baseado em sua semântica de papeis/ações/organizações em objetos para que um conflito indireto seja transformado em direto mediante sua política e, assim, possa ser detectado pelos algoritmos de aprendizagem de máquina.

Neste experimento, então, para exemplificar a propagação de políticas um novo arquivo de políticas foi randomicamente gerado pelo programa **`geracao_politicas2.py`**, disponível em <https://bit.ly/3jpYW8P>, baseado em um ambiente hospitalar. O arquivo,

com as políticas geradas, está disponível em <https://bit.ly/3h2HBBv>. Uma parte do programa está exibido na Figura 63.

Figura 63 – Geração políticas 2

```
#listas com as opções das políticas
acesso = ["Permitted", "Forbidden", "Obliged"] #KP

org = ['Hospital_Florence_Nightingale', 'Ortopedia', 'Enfermaria', 'Hospital_Florence_Nightingale', 'Unidade_Administrativa',
       'Radiologia', 'Unidade_Administrativa'] #ORG

sujeito = ['Médico', 'Dra. Daenerys_Targaryen', 'Arwen_Undomiel','Funcionário', 'Jaime_Lannister', 'Enfermeiros', 'Secretário',
       'Ada_Lovelace', 'Carl_Sagan', 'Grupo_Almoxarifado','Usuario_Qualquer', 'Italo_Calvino','Secretario_de_Unidade_Administrativa',
       'PROTOCOLIZADOR3', 'PROTOCOLIZADOR1', 'Elrond', 'Tyrion_Lannister','Eddard_Stark', 'Neurologista', 'Coordenador'] #SR

acao = ['Abertura', 'Solicitar', 'Acessar', 'Gerar', 'Calcular','Registrar', 'Solicitacao', 'Analise', 'Requisitar', 'Criar',
       'Inserir', 'Cadastrar', 'Incluir', 'Alterar', 'Efetivar','LancarRegistroPaciente', 'SolicitacaoAbertura', 'Cancelamento',
       'Acessar', 'Record', 'Create', 'Open', 'Close'] #AA

objeto = ['Documentos', 'Produtos', 'Almoxarifado', 'Materiais','Planilhas_de_Calculo', 'Imposto', 'Sistema_Hospitalar',
       'Paciente', 'Central_de_Copias', 'Central_de_Copias_Analise','Material', 'Guia_de_Requisicao_Medicamento',
       'PedidoMedicamento', 'RegistraPaciente', 'HistoricoPaciente','Processo', 'ProcessoCompra', 'DespacharDocumentos',
       'Despachar'] #OV

tamAcesso = len(acesso)
tamOrg = len(org)
tamSujeito = len(sujeito)
tamAcao = len(acao)
tamObjeto = len(objeto)

politicas = []
quantidade_gerada = 200
gera_politicas(politicas, quantidade_gerada)

teste = pd.DataFrame(np.array(politicas), columns=['politica'])
teste.to_csv('politicas_geradas-2.csv', index=False)

#retira as aspas
with open("politicas_geradas-2.csv", "rt") as fin:
    with open("politicas_geradas_out-2.csv", "wt") as fout:
        for line in fin:
            fout.write(line.replace("\"" , ""))

#altera a primeira linha pelas colunas corretas
from io import StringIO
buffer = StringIO()
```

Fonte: compilação do autor

O arquivo gerado como arquivo **`geracao_politicas2.py`** ficou com o aspecto da Figura 64.

O exemplo que será abordado, por fins de simplificação, será a propagação de Sujeitos/Papéis e Ações. Para tanto, uma análise semântica das políticas geradas é necessária (visto que não há ontologia para o domínio abordado, pois este não é foco do trabalho).

E, claro, o conflito detectado na política propagada é o de modalidade, descrito em Lupu e Sloman (1999) como sendo o conjunto das seguintes combinações de políticas de autorização e obrigação: $Aut + /Aut-$, $Obli + /Obli-$, $Obli + /Aut-$, sendo Aut como Autorização (*Permitted*) e $Obli$ como Obrigação (*Obligation*).

Para os objetivos do experimento, verifica-se que em relação aos sujeito/papel há os seguintes relacionamentos baseados na semântica do domínio do sistema, ilustrado na Figura 65.

Em relação à organização e as ações, os relacionamentos da Figura 66 foram definidos. Todos essas relações semânticos usam uma estrutura de dados conhecida como dicionário, nativa da linguagem Python descrita em Rossum e Drake (2009).

Figura 64 – Arquivo gerado

```
base_nova
```

	Acesso	Organizacao	Sujeito	Acao	Objeto
0	Permitted	Hospital_Florence_Nightingale	Médico	Abertura	Documentos
1	Forbidden	Hospital_Florence_Nightingale	Dra. Daenerys_Targaryen	Abertura	Documentos
2	Permitted	Hospital_Florence_Nightingale	Arwen_Undomiel	Solicitar	Produtos
3	Forbidden	Hospital_Florence_Nightingale	Funcionário	Solicsujeioitar	Produtos
4	Permitted	Hospital_Florence_Nightingale	Arwen_Undomiel	Acessar	Almoxarifado
...	...	...	...	...	...
59	Permitted	Enfermaria	Neurologista	Analise	Central_de_Copias
60	Permitted	Hospital_Florence_Nightingale	Neurologista	Requisitar	Material
61	Permitted	Hospital_Florence_Nightingale	Coordenador	Requisitar	Material
62	Permitted	Hospital_Florence_Nightingale	Coordenador	Solicitacao	Central_de_Copias
63	Permitted	Ortopedia	Coordenador	Solicitacao	Central_de_Copias

Fonte: compilação do autor

Figura 65 – Semântica Sujeito

```
semantica_sujeito = {
    'Médico'              : [' Dra. Daenerys_Targaryen',' Neurologista'],
    ' Enfermeiros'        : [' Arwen_Undomiel', ],
    ' Secretário'         : [' Ada_Lovelace', ' Carl_Sagan'],
    ' Coordenador'        : [' Eddard_Stark'],
    ' Grupo_Almoxarifado' : [' Elrond', ' Tyrion_Lannister']
}
```

Fonte: compilação do autor

Figura 66 – Semântica organização e ação

```
semantica_ORG = {
    'Hospital_Florence_Nightingale': ['Ortopedia', 'Enfermaria', 'Radiologia']
}

semantica_acao = {
    ' Solicitar' : [' Solicitacao'],
    ' Cadastrar' : [' Incluir', ' Inserir'],
    ' Registrar' : [' Efetivar']
}
```

Fonte: compilação do autor

Alguns artefatos de software para apoiar a propagação das políticas, todos desenvolvidos em Python no notebook disponível em <https://bit.ly/3x7EFJe> são mostrados no Código 3.3, no Código 3.4, na Figura 67 e na Figura 68. Nestes é exibido uma **`Classe Politica`** que será o contêiner das políticas, com alguns métodos padrões para exibição da política e de um possível conflito direto além de duas funções **`retorna_politica_por_sujeito`** e **`retorna_politica`** que monta duas listas

que fazem a leitura do dicionário de semântica dos sujeitos e realiza uma busca na base em todas as políticas que tenham os valores hierárquicos das respectivas chaves. Com isso uma nova política é criada refletindo a hierarquia de Acesso dentro da semântica.

Código 3.3 – Classse Política

```
class Politica(object):
    """Classe Politica: responsavel em armazenar todos os dados de uma politica"""

    def __init__(self, acesso,organizacao,sujeito,acao,objeto, conflito = 0):
        self.acesso = acesso
        self.organizacao = organizacao
        self.sujeito = sujeito
        self.acao = acao
        self.objeto = objeto
        self.conflito = conflito

    def __repr__(self):
        return '{}, {}, {}, {}, {}, {}\n'.format(self.acesso , self.organizacao, \
        self.sujeito, self.acao, self.objeto, self.conflito)

    def __str__(self):
        return '[{}, {}, {}, {}, {}, {}]'.format(self.acesso , self.organizacao, \
        self.sujeito, self.acao, self.objeto, self.conflito);

    def mostra_conflito(self, dataset):
        if not dataset.query("Acesso!='{}' & Organizacao=='{}' & Sujeito=='{}'& \
        Acao=='{}'".format(self.acesso, self.organizacao, self.sujeito, self.acao)).empty:
            df = dataset.query("Acesso!='{}' & Organizacao=='{}' & Sujeito=='{}'& \
            Acao=='{}'".format(self.acesso, self.organizacao, self.sujeito, self.acao))
            print('Conflitos!')
            print(df.iloc[:,0:5])
            linhas_em_conflito = df.index.to_list()
            print()

            print('Politica: {}'.format(self.__str__()))

    def propaga_politica(self, politica):
        pass
```

Código 3.4 – Função que retorna políticas por sujeito

```
def retorna_politica_por_sujeito(sujeito):
    lista = []
    #for chave, valor in semantica_sujeito.items():
    #print("Para a chave {}".format(sujeito))
    for linha in base.itertuples():
        for i, j in enumerate(semantica_sujeito.get(sujeito)):

            if linha.Sujeito == j:

                p = '{}, {}, {}, {}, {}, {}'.format(linha.Acesso, linha.Organizacao, \
                linha.Sujeito, linha.Acao, linha.Objeto, linha.Conflito)

                lista.append(p)
    return lista
```

Figura 67 – Função retorna-politica

```
def retorna_politica(sujeito):
  lista_politicas = []
  for linha in base.itertuples():
    if linha.Sujeito == str(sujeito):
      p = '{}, {}, {}, {}, {}, {}'.format(linha.Acesso, linha.Organizacao, \
                                          linha.Sujeito, linha.Acao, linha.Objeto, linha.Conflito)

      lista_politicas.append(p)
  return lista_politicas
```

Fonte: compilação do autor

Figura 68 – Função que propaga as politicas

```
for x in semantica_sujeito.keys():
  #print("Para a chave {}".format(x))
  #print(retorna_politica_por_sujeito(x))
  lista_keys = retorna_politica(x)
  #print(lista_keys)
  #print("")
  lista_values = retorna_politica_por_sujeito(x)
  #print(lista_values)
  for i, j in enumerate(lista_keys):
    p = lista_keys[i]
    string = p.split(',')
    politica = Politica(string[0],string[1], string[2], string[3], string[4], string[5])
    #print(politica.sujeito) ---> esse que vai ser comparado
    for k, y in enumerate(lista_values):
      p2 = lista_values[k]
      string2 = p2.split(',')
      politica2 = Politica(string2[0],string2[1], string2[2], string2[3], string2[4], string2[5])
      #print(politica2)  -----> esse é o outro que será comparado
      if (politica.organizacao == politica2.organizacao) and \
      (politica.acesso != politica2.acesso) and \
      (politica.acao == politica2.acao) and (politica.objeto == politica2.objeto):
        politica3 = Politica(string[0],string2[1], string2[2], string2[3], string2[4], string2[5])
        return politica3
```

Fonte: compilação do autor

As novas políticas geradas, após a propagação foram as seguintes mostradas no Código 3.5. Como a base é reduzida e a semântica também simplificada, somente uma política foi propagada já que atuava mesma organização, sobre sujeitos semanticamente e hierarquicamente relacionados e sobre o mesmo objeto e possuíam, cf. Kamoda, Yamaoka e Matsuda (2005), combinações de autorização/proibição que entravam em conflito por conta dos relacionamentos o que configura um conflito indireto, como mostra a seção 3.5.

Código 3.5 – Políticas propagadas

```
[ Permitted,  Hospital_Florence_Nightingale,   Dra. Daenerys_Targaryen,  Abertura,
    Documentos]
```

Esta política é então incluída no dataset de políticas para refletir a propagação dos sujeitos/ações sobre um objeto, cf. Kamoda, Yamaoka e Matsuda (2005). Desta forma, os

conflitos de modalidade (diretos) podem ser detectados da forma que foi mostrada nesta dissertação nas subseção 3.2.3, na seção 3.3 e na seção 3.4. Seguindo os mesmos passos descritos nestas seções, um modelo foi gerado usando o algoritmo MLP para o dataset criado nesta seção e novas instâncias foram geradas randomicamente com o programa **`geracao_politicas2.py`**. O modelo foi então aplicado para detectar os conflitos após a propagação das políticas para detectar também os conflitos indiretos. A Figura 69 mostra as instâncias geradas e o resultado da detecção é mostrado na Figura 70.

Figura 69 – Novas instâncias geradas

```
geradas = [
        'Permitted, Unidade_Administrativa, Arwen_Undomiel, Create, ProcessoCompra',
        'Forbidden, Unidade_Administrativa, Arwen_Undomiel, Gerar, Processo',
        'Obliged, Unidade_Administrativa, Arwen_Undomiel, Open, Processo',
        'Permitted, Unidade_Administrativa, Enfermeiros, Close, DespacharDocumentos',
        'Permitted, Unidade_Administrativa, Enfermeiros, Create, DespacharDocumentos',
        'Permitted, Unidade_Administrativa, Enfermeiros, Record, DespacharDocumentos',
        'Permitted, Unidade_Administrativa, Enfermeiros, Close, Despachar',
        'Permitted, Unidade_Administrativa, Enfermeiros, Create, Despachar',
        'Permitted, Unidade_Administrativa, Enfermeiros, Record, Despachar',
        'Permitted,Enfermaria, Elrond, Analise, Central_de_Copias',
        'Permitted,Radiologia, Tyrion_Lannister, Requisitar, Material'
]
```

Fonte: compilação do autor

Figura 70 – Resultados

```
1 resultado

[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Fonte: compilação do autor

Assim, neste caso, com um dataset menor e após a propagação das políticas, a acurácia ficou em 100.0% já que todas as instâncias foram corretamente classificadas. Esta forma de detecção, entretanto, não foi foco principal deste trabalho, porque a propagação de políticas, como parte do pré-processamento analisa-as em pares (e com mais de uma consulta com laços aninhados à base e às listas de políticas analisadas) como mostram a Figura 67 e a Figura 68. De qualquer forma é uma abordagem válida para atacar a detecção de conflitos indiretos. Neste caso, as semânticas de organização, ação e até mesmo de objetos devem ser consideradas sendo, portanto, recomendada a construção de uma ontologia para o conjunto de conceitos, relações, restrições e axiomas de cada domínio onde os conflitos entre políticas serão detectados como foi realizado em Sarkis (2017) seguindo os modelos de Clark (2020) e usando a API Web Ontology Language descrita em Horridge e Bechhofer (2011).

3.6 Análise dos resultados

Os experimentos realizados no Capítulo 3 corroboram a hipótese descrita na subseção 1.1.4 de que a detecção de conflitos pode ser transformada em um problema da tarefa de classificação da mineração de dados e do aprendizado de máquina com acurácia suficiente para que esta técnica seja utilizada junto com as demais da literatura além de ter a condição de detecção dos conflitos em tempo de execução já que o modelo de mineração de dados já foi gerado e melhora com o tempo, como são características destes modelos cf. Alpaydin (2014). As vantagens sobre as outras abordagens relatadas na literatura são:

- A detecção do conflito pode ser realizada em tempo de execução, pois os modelos já estão treinados;
- As políticas são verificadas e analisadas em conjunto, "em lote" e não em pares já que a análise em pares é um problema computacionalmente custoso da classe NP-Completo como demonstrado por Shoham e Tennenholtz (1995);
- As três técnicas analisadas neste trabalho mostraram-se eficazes com acurácias acima da proposta inicialmente (97%) cf. o mostrado na seção 3.3, seção 3.3, subseção 3.3.1 e subseção 3.3.2;
- os modelos de aprendizagem de máquina desenvolvidos nos experimentos podem ser aplicados em outros contextos outros contextos de conflitos que se assemelhem aos conflitos de modalidade observados neste estudo e podem servir de base para estudos futuros de detecção de conflitos usando-se aprendizagem de máquina com aprofundamentos ainda mais amplos do que o escopo deste trabalho;
- Ao analisar uma nova política, não é necessário "varrer" ou consultar todas as instâncias do *dataset* novamente já que os modelos já estão treinados;
- os modelos de aprendizagem de máquina tem a tendência a melhorarem a eficácia à medida que a quantidade de instâncias cresce.

OS modelos e algoritmos demonstrados nos experimentos tiveram acurácia acima de 97% o que mostra que a detecção de conflitos em políticas (ou normas) pode ser colocada como uma classe de problemas a serem resolvidos de forma eficiente por técnicas de aprendizagem de máquina.

As limitações e ameaças à validade do estudo concentram-se no fato de que as políticas, embora suficientemente genéricas, não são produzidas por sistemas de organizações do mundo real e sim geradas randomicamente. Outra limitação é o fato de que o estudo concentra apenas os conflitos de modalidade (os diretos) além de que a indicação de quais políticas conflitantes não é iterativa e sim mediante consulta à base (algo que pode ser automatizado como um trabalho futuro).

4 Considerações finais

Resultados Alcançados

Ao fim deste trabalho os seguintes resultados foram alcançados:

- Mostrou-se que o problema da detecção de conflitos em políticas pode ser convertido em um problema de *data mining* (mineração de dados) resolvido pela tarefa da classificação;
- Demonstrou-se que o problema da detecção de conflitos diretos é provavelmente linearmente separável, sendo, portanto resolvido com eficiência por classificadores lineares (que tem como característica a resolução deste tipo de problemas);
- Mostrou-se que a política nova (*instância inédita*) é ou não conflitante imediatamente após a criação da mesma usando como base o treinamento dos modelos de mineração e aprendizagem de máquina no *dataset* de políticas existente;
- Modelou-se uma forma de detectar estes conflitos mediante o uso de diferentes algoritmos e técnicas da aprendizagem de máquina que consigam acurácias que suportem a verificação de conflitos diretos;
- Estabeleceu-se uma relação entre aprendizagem de máquina, técnicas de mineração de dados e a resolução do problema do conflito entre políticas;
- Determinou-se e comparou-se quais algoritmos e técnicas são mais adequados para cada tipo de conflito nas políticas usando as suas acurácias;
- Usou-se e comparou-se estatisticamente o desempenho, a precisão e taxa de acertos das principais técnicas usadas no aprendizado de máquina: redes neurais artificiais (RNA), Support Vector Machines (SVM) e Random Forest.

Com acurácias médias acima de 97% na classificação/detecção dos conflitos diretos, os algoritmos Multilayer Perceptron, SVM e Random Forest tiveram, todos, um *recall* na faixa média de 99,4% e correlação de Matthews (ϕ) média de 0.98, o que mostra que se pode considerar que os algoritmos são eficazes para detectar conflitos diretos em políticas de controle de acesso. Desta forma conclui-se que:

- Esta pesquisa mostrou a possibilidade de *converter a detecção de conflitos a um problema de classificação* especificamente, para os conflitos ***diretos***;

- O classificador mais acurado nos experimentos, cf. testes estatísticos da subseção 3.3.3.2, o *MultiLayer Perceptron* que é um classificador que modela uma rede neural de várias camadas para classificar instâncias desconhecidas (WITTEN et al., 2016);
- Testes estatísticos podem ser usados para determinar a eficiência das soluções propostas;
- Os modelos e algoritmos demonstrados nos experimentos tiveram acurácias médias acima de 97% o que demonstra, portanto, que a detecção de conflitos pode, *de facto*, ser transformada em um problema da tarefa de classificação da mineração de dados e do aprendizado de máquina.

As vantagens deste estudo sobre outras abordagens relatadas na literatura são: A detecção do conflito *é passível de ser processada em tempo de execução*, pelo fato dos modelos já estarem antecipadamente treinadas; as políticas são checadas "em lote" e não em pares — uma importante característica dos modelos de aprendizagem de máquina e mineração de dados cf. Alpaydin (2014) — pois a análise em pares é um problema da classe NP-Completo conforme Shoham e Tennenholtz (1995); outro adicional é que os modelos de aprendizagem de máquina desenvolvidos nos experimentos juntamente com os hiperparâmetros encontrados podem ser empregados em outros contextos de conflitos, pois são genéricos para tal.

E, finalmente, ao se analisar uma nova política, não é necessário "percorrer" ou consultar todas as instâncias do *dataset* repetidamente já que os modelos estão treinados e estes modelos de aprendizagem de máquina tem a tendência a melhorarem a eficácia à medida que a quantidade de instâncias cresce.

Como indicação de trabalhos futuros apontamos a detecção utilizando modelos de aprendizagem de máquina para conflitos indiretos e políticas dinâmicas que tem, por exemplo, a data de ativação como um limitador temporal além de construção de uma aplicação em alguma linguagem de programação (como Java) que faça uso dos modelos e seus hiperparâmetros para a detecção de conflitos, indicando quais as políticas, de forma iterativa, estão conflitando.

Referências

ACADEMY, D. S. *Deep Learning Book, 2019*. Acesso em: 06 de março de 2020., 2020. Disponível em: <http://deeplearningbook.com.br/>.

AHO, A. V.; HOPCROFT, J. E.; UIIMAN, J. D. *The Design and Analysis of computer algorithms*. 1st. ed. Boston, MA, United States: Addison-Wesley Longman Publishing Co., Inc., 1974. ISBN 0-201-00029-6.

AL-RADAIDEH, Q. A.; NAGI, E. A. Using Data Mining Techniques to Build a Classification Model for Predicting Employees Performance. 2012.

ALECRIM, E. *O que é Big Data?* Série Infowester: INFOWESTER. Disponível em: https://www.infowester.com/big-data.php. Acesso em 12/12/2019, 2019.

ALPAYDIN, E. *Introduction to Machine Learning, Third Edition*. 3ª. ed. The MIT Press, 2014. (Adaptive Computation and Machine Learning). ISBN 978-0-262-02818-9. Disponível em: <https://mitpress.ublish.com/ereader/26/?preview#page/Cover>.

AMARAL, F. *Aprenda Mineração de Dados: Teoria e prática*. Rio de Janeiro, Brasil: Alta Books Editora, 2016. 220 p. (Volume 1 da Série: Autoria Nacional). ISBN 9788576089889.

AMARAL, F. *Introdução à Ciência de Dados: Mineração de dados e big data*. Rio de Janeiro, Brasil: Alta Books, 2018. ISBN 9788550804163.

AUTREL, F.; COMA, C.; AL, e. MotOrBAC 2: a security policy tool. 2008.

BAEZA-YATES, R.; RIBEIRO-NETO, B. *Recuperação de Informação: Conceitos e Tecnologia das Máquinas de Busca*. 2a. edição. ed. Porto Alegre, RS, Brasil: Bookman, 2013.

BELLETTINI, C.; BERTINO, E.; FERRARI, E. Role based access control models. v. 6, n. 2, p. 21–29, 2001. ISSN 13634127. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S1363412701002047>.

BROWN, L.; STALLINGS, W. *Segurança de Computadores: Princípios e Práticas*. Rio de Janeiro, RJ, Brasil: Elsevier Brasil, 2017. ISBN 978-85-352-6450-0. Disponível em: <https://books.google.com.br/books?id=y2DcAwAAQBAJ>.

BRUCE, A.; BRUCE, P. *Estatística Prática para Cientistas de Dados*. Rio de Janeiro, Brasil: Alta Books, 2019. ISBN 978-85-508-1080-5.

BUI, T.; STOLLER, S. D.; LE, H. Efficient and Extensible Policy Mining for Relationship-Based Access Control. In: *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*. New York, NY, USA: ACM, 2019. (SACMAT '19), p. 161–172. ISBN 978-1-4503-6753-0. Disponível em: <http://doi.acm.org/10.1145/3322431.3325106>.

BUI, T.; STOLLER, S. D.; LI, J. Greedy and evolutionary algorithms for mining relationship-based access control policies. v. 80, p. 317–333, 2019. ISSN 0167-4048. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167404818304565>.

CASACA, J. A.; CORREIA, M. F. Porque é necessária a segurança da informação? da estratégia às políticas de segurança. v. 0, n. 3, p. 89–116, 2013. ISSN 1647-1342. Number: 3. Disponível em: <http://revistas.lis.ulusiada.pt/index.php/lpis/article/view/157>.

CASTRO, C. L. d.; BRAGA, A. P. Supervised learning with imbalanced data sets: an overview. v. 22, n. 5, p. 441–466, 2011. ISSN 0103-1759. Publisher: Sociedade Brasileira de Automática. Disponível em: <http://www.scielo.br/scielo.php?script=sci_abstract&pid=S0103-17592011000500002&lng=en&nrm=iso&tlng=pt>.

CHAKRABORTY, S.; SANDHU, R.; KRISHNAN, R. On the feasibility of attribute-based access control policy mining. In: *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*. Los Angeles, CA, USA: [s.n.], 2019. p. 245–252.

CHEN, M. Q. Flight Conflict Detection and Resolution Based on Neural Network. In: *2011 International Conference on Computational and Information Sciences*. Chengdu, Sichuan. China: [s.n.], 2011. p. 860–862. ISSN: null.

CHRISTODOULOU, M. A.; KONTOGEORGOU, C. Collision avoidance in commercial aircraft Free Flight via neural networks and non-linear programming. *International Journal of Neural Systems*, v. 18, n. 5, p. 371–387, out. 2008. ISSN 0129-0657.

CHWIF, L.; MEDINA, A.; SIMULATE, T. *Modelagem e simulação de eventos discretos, 4a edição: Teoria e aplicações*. Elsevier Brasil, 2014. ISBN 978-85-352-7933-7. Disponível em: <https://books.google.com.br/books?id=wZmoBQAAQBAJ>.

CLARK, P. *KBS/Ontology Projects Worldwide*. 2020. Disponível em: <https://www.cs.utexas.edu/users/mfkb/related.html>.

CORTES, C.; VAPNIK, V. Support-vector networks. v. 20, n. 3, p. 273–297, 1995. ISSN 1573-0565. Disponível em: <https://doi.org/10.1007/BF00994018>.

CRABBé, A. et al. Neural Network MLP Classifier. jan. 2020. Publisher: https://mlp-image-classifier.readthedocs.io/en/latest/. Disponível em: <https://lirias.kuleuven.be/3345825>.

CUPPENS, F.; CUPPENS-BOULAHIA, N.; GHORBEL, M. B. High level conflict management strategies in advanced access control models. v. 186, p. 3–26, 2007. ISSN 15710661. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S1571066107004562>.

DAVY, S.; JENNINGS, B.; STRASSNER, J. Application domain independent policy conflict analysis using information models. In: *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*. Salvador, Bahia, Brazil: [s.n.], 2008. p. 17–24. ISSN: 2374-9709.

DEBAR, H.; BECKER, M.; SIBONI, D. A neural network component for an intrusion detection system. In: *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy*. Oakland, CA: [s.n.], 1992. p. 240–250. ISSN: null.

DENG, X. et al. An improved method to construct basic probability assignment based on the confusion matrix for classification problem. v. 340-341, p. 250–261, 2016. ISSN 0020-0255. Disponível em: <http://www.sciencedirect.com/science/article/pii/S002002551600044X>.

DOHERTY, N. F.; FULFORD, H. Do information security policies reduce the incidence of security breaches: An exploratory analysis. v. 18, n. 4, p. 21–39, 2005. ISSN 1040-1628. Disponível em: <https://doi.org/10.4018/irmj.2005100102>.

DRISCOLL, W. C. Robustness of the ANOVA and tukey-kramer statistical tests. v. 31, n. 1, p. 265–268, 1996. ISSN 0360-8352. Disponível em: <https://www.sciencedirect.com/science/article/pii/0360835296001271>.

DUNLOP, N.; INDULSKA, J.; RAYMOND, K. Dynamic conflict detection in policy-based management systems. In: *Proceedings. Sixth International Enterprise Distributed Object Computing*. Lausanne, Switzerland: [s.n.], 2002. p. 15–26. ISSN: null.

ELRAKAIBY, Y.; CUPPENS, F.; CUPPENS-BOULAHIA, N. Formal enforcement and management of obligation policies. v. 71, n. 1, p. 127–147, 2012. ISSN 0169-023X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0169023X11001248>.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From data mining to knowledge discovery in databases. v. 17, n. 3, p. 37–37, 1996. ISSN 2371-9621. Disponível em: <https://www.aaai.org/ojs/index.php/aimagazine/article/view/1230>.

FERNEDA, E. Redes neurais e sua aplicação em sistemas de recuperação de informação. v. 35, n. 1, 2006. ISSN 1518-8353. Disponível em: <http://revista.ibict.br/ciinf/article/view/1149>.

FERRAIOLO, D.; ATLURI, V.; GAVRILA, S. The policy machine: A novel architecture and framework for access control policy specification and enforcement. Erode, India, v. 57, n. 4, p. 412–424, 2011. ISSN 1383-7621. Disponível em: <https://doi.org/10.1016/j.sysarc.2010.04.005>.

FERRAIOLO, D. F. et al. Proposed NIST standard for role-based access control. v. 4, n. 3, p. 224–274, 2001. ISSN 1094-9224. Disponível em: <https://doi.org/10.1145/501978.501980>.

FERRARI, D. G.; SILVA, L. N. d. C. *Introdução a mineração de dados*. Editora Saraiva, 2017. ISBN 9788547200992. Disponível em: <https://books.google.com.br/books?id=SSlrDwAAQBAJ>.

FONG, P. W. Relationship-based access control: protection model and policy language. In: *Proceedings of the first ACM conference on Data and application security and privacy*. Association for Computing Machinery, 2011. (CODASPY '11), p. 191–202. ISBN 978-1-4503-0466-5. Disponível em: <https://doi.org/10.1145/1943513.1943539>.

FONTES, E. *Políticas e Normas para a Segurança da Informação*. Rio de Janeiro, RJ, Brasil: Brasport, 2012. 289 p. ISBN 978-85-7452-515-0.

FUGINI, M.; BELLETTINI, C. *Information Security Policies and Actions in Modern Integrated Systems*. Idea Group Pub., 2004. ISBN 978-1-59140-186-5. Disponível em: <https://books.google.com.br/books?id=uTgd_lruELcC>.

GÉRON, A. *Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow*. Rio de Janeiro, Brasil: Alta Books, 2019.

GOLDSCHMIDT, R.; PASSOS, E. *Data mining: um guia Prático*. Rio de Janeiro, RJ, Brasil: Editora Campus, 2005. Elsevier. ISBN 978-85-352-1877-0.

GRUS, J.; NASCIMENTO, W. *Data Science Do Zero*. Rio de Janeiro, Brasil: ALTA BOOKS, 2016. ISBN 9788576089988. Disponível em: <https://books.google.com.br/books?id=EWJCvgAACAAJ>.

GUERRERO-HIGUERAS, A. M.; DeCastro-GARCIA, N.; MATELLAN, V. Detection of cyber-attacks to indoor real time localization systems for autonomous robots. v. 99, p. 75–83, 2018. ISSN 0921-8890. Disponível em: <http://www.sciencedirect.com/science/article/pii/S092188901730283X>.

HACHANA, S.; CUPPENS-BOULAHIA, N.; CUPPENS, F. Mining a high level access control policy in a network with multiple firewalls. v. 20, p. 61–73, 2015. ISSN 2214-2126. Disponível em: <http://www.sciencedirect.com/science/article/pii/S221421261400146X>.

HAGAN, M. T.; DEMUTH, H. B.; BEALE, M. H. *Neural Network Design*. 1st. ed. Boston, MA. USA: PWS Publishing Co., 1996. 1012 p. ISBN 978-0-534-94332-5.

HAYKIN, S. *Redes Neurais - 2ed*. Porto Alegre, RS, Brasil: Bookman, 2001. ISBN 978-85-7307-718-6.

HE, Q.; ANTóN, A. I. Requirements-based access control analysis and policy specification (ReCAPS). v. 51, n. 6, p. 993–1009, 2009. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584908001699>.

HORRIDGE, M.; BECHHOFER, S. The OWL API: A Java API for OWL ontologies. *Semantic Web*, v. 2, n. 1, p. 11–21, jan. 2011. ISSN 1570-0844.

HOSSIN, M.; M.N, S. A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, v. 5, p. 01–11, mar. 2015.

HWANG, J. et al. ACPT: A tool for modeling and verifying access control policies. In: *2010 IEEE International Symposium on Policies for Distributed Systems and Networks*. George Mason University, Fairfax, Virginia, USA: [s.n.], 2010. p. 40–43.

JIN, X.; CHEU, R. L.; SRINIVASAN, D. Development and adaptation of constructive probabilistic neural network in freeway incident detection. *Transportation Research Part C: Emerging Technologies*, v. 10, n. 2, p. 121–147, abr. 2002. ISSN 0968-090X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0968090X01000079>.

KADIMISETTY, A. *TensorFlow — A hands-on approach*. Towards Data Science: Disponível em: https://towardsdatascience.com/tensorflow-a-hands-on-approach-8614372f021f, 2018. (Acesso em 07 de julho de 2020). Library Catalog: towardsdatascience.com.

KALAM, A. et al. Organization based access control. In: *Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks*. Lake Como, Italy: [s.n.], 2003. p. 120–131.

KALASKAR, A.; BARKADE, V. FP-growth policy mining for access control policies. In: *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. Pune, India: [s.n.], 2018. p. 1–4.

KAMODA, H.; YAMAOKA, M.; MATSUDA, S. Policy Conflict Analysis Using Free Variable Tableaux for Access Control in Web Services Environments. p. 8, 2005.

KASTURI, S. N. *Underfitting and Overfitting in machine learning and how to deal with it !!!* [S.l.]: Disponível em: https://towardsdatascience.com/underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6fe4a8a49dbf, 2019. (Acesso em: 04 de Abril de 2020). Library Catalog: towardsdatascience.com.

KESAVARAJ, G.; SUKUMARAN, S. A study on classification techniques in data mining. In: *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. Tiruchengode, India: [s.n.], 2013. p. 1–7.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. 2017. Disponível em: <http://arxiv.org/abs/1412.6980>.

KLOSTERMAN, S. *Projetos de Ciência de Dados com Python: Abordagem de estudo de caso para a criação de projetos de ciência de dados bem-sucedidos usando Python, pandas e scikit-learn*. NOVATEC, 2020. ISBN 9786586057102. Disponível em: <https://books.google.com.br/books?id=GLTfDwAAQBAJ>.

KLUYVER, T. et al. Jupyter notebooks – a publishing format for reproducible computational workflows. In: LOIZIDES, F.; SCHMIDT, B. (Ed.). *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Göttingen, Germany, 2016. p. 87 – 90.

KNAPP, K. J. et al. Information security policy: An organizational-level process model. v. 28, n. 7, p. 493–508, 2009. ISSN 0167-4048. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167404809000765>.

KOCH, M.; MANCINI, L. V.; PARISI-PRESICCE, F. Conflict Detection and Resolution in Access Control Policy Specifications. In: NIELSEN, M.; ENGBERG, U. (Ed.). *Foundations of Software Science and Computation Structures*. Berlin, Heidelberg: Springer, 2002. (Lecture Notes in Computer Science), p. 223–238. ISBN 978-3-540-45931-6.

KROPIWIEC, D. D. Policy viewer: Ferramenta para visualizac¸~ao de pol´ıticas de seguranc¸a em grafos. p. 100, 2005.

KUANG, T. P. et al. MODALITY CONFLICT DETECTION MODEL WITH AUTHORIZATION PROPAGATION IN POLICY EVALUATION. . *Vol.*, n. 17, p. 14, 2020.

KUMAR, A.; JAIN, M. *Ensemble Learning for AI Developers: Learn Bagging, Stacking, and Boosting Methods with Use Cases*. Apress, 2020. ISBN 978-1-4842-5939-9. Disponível em: <https://books.google.com.br/books?id=1FVuzQEACAAJ>.

KUMAR, R. Classification Algorithms for Data Mining: A Survey. *International Journal of Innovations in Engineering and Technology*, v. 1, n. 2, p. 8, 2012.

LI, N.; TRIPUNITARA, M. V. Security analysis in role-based access control. ACM Trans. Inf. Syst. Secur. ACM, New York. v.9. p.391-420, p. 28, 2006.

LIMA, I.; PINHEIRO, C.; SANTOS, F. *Inteligência Artificial*. Elsevier Brasil, 2016. ISBN 978-85-352-7809-5. Disponível em: <https://books.google.com.br/books?id=qjJeBgAAQBAJ>.

LIMA, R. A. F.; PEREIRA, A. C. M. Fraud detection in web transactions. ACM Press, p. 273, 2012. Disponível em: <http://dl.acm.org/citation.cfm?doid=2382636.2382695>.

LOPES, I. M. Adopção de políticas de segurança de sistemas de informação na administração pública local em portugal. 2012. Accepted: 2012-09-03T09:19:35Z Publisher: Universidade do Minho. Disponível em: <https://bibliotecadigital.ipb.pt/handle/10198/7422>.

LUGER, G. F. *Inteligência Artificial*. Edição: 4. São Paulo. SP. Brasil: ARTMED Editora S.A., 2015.

LUPU, E.; SLOMAN, M. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, v. 25, n. 6, p. 852–869, nov. 1999. ISSN 2326-3881.

MACHADO, A. *Estudo da EMC prevê que volume de dados virtuais armazenados será seis vezes maior em 2020*. 2014. O Globo. Disponível em: <https://glo.bo/2Eo8AGK>.

MARCIANO, J. L. P. Segurança da informação - uma abordagem social. p. 212, 2006.

MARTIN, E.; Tao Xie. Inferring access-control policy properties via machine learning. In: *Seventh IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06)*. London, Ontário, Canada: [s.n.], 2006. p. 4 pp.–238.

MCCULLOCH, W. S.; PITTS, W. A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY. v. 52, n. 1, p. 17, 1943.

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

MINEWISKAN, O. D. *Modelos de mineração (Analysis Services-Mineração de dados)*. Acesso em: 06 de março de 2020., 2020. (Microsoft Academy). Library Catalog: docs.microsoft.com. Disponível em: <https://docs.microsoft.com/pt-br/analysis-services/data-mining/mining-models-analysis-services-data-mining>.

MINSKY, M.; PAPERT, S. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969. Disponível em: <https://books.google.com.br/books?id=KhI-uwEACAAJ>.

MOFFETT, J. D.; SLOMAN, M. S. Policy conflict analysis in distributed system management. v. 4, n. 1, p. 1–22, 1994. ISSN 1054-1721. Disponível em: <https://doi.org/10.1080/10919399409540214>.

MOHAN, A. et al. Detection of conflicts and inconsistencies in taxonomy-based authorization policies. v. 2011, p. 590–594, 2012. ISSN 2156-1125.

MOHRI, M.; ROSTAMIZADEH, A.; TALWALKAR, A. *Foundations of Machine Learning, Second Edition*. 2ª. ed. The MIT Press, 2018. (Adaptive Computation and Machine Learning). ISBN 978-0-262-03940-6. Disponível em: <https://mitpress.mit.edu/books/foundations-machine-learning-second-edition>.

MONTEIRO, J. R. Adoção de políticas de segurança de sistemas de informação nas universidades moçambicanas. p. 66, 2017.

MUKKAMALA, S.; JANOSKI, G.; SUNG, A. Intrusion detection using neural networks and support vector machines. In: *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*. Honolulu, HI, USA: [s.n.], 2002. v. 2, p. 1702–1707 vol.2. ISSN: 1098-7576.

OBAIDAT, M.; MACCHAIROLO, D. A multilayer neural network system for computer access security. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 24, n. 5, p. 806–813, maio 1994. ISSN 2168-2909.

OLIPHANT, T. E. *A guide to NumPy*. North Charleston, SC, United States: Trelgol Publishing USA, 2006. v. 1.

PASZKE, A. et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Curran Associates, Inc., 2019. 8024–8035 p. Disponível em: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.

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

QI, H.; DI, X.; LI, J. Formal definition and analysis of access control model based on role and attribute. v. 43, p. 53–60, 2018. ISSN 2214-2126. Disponível em: <http://www.sciencedirect.com/science/article/pii/S221421261730368X>.

RAZALI, N. M.; YAP, B. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. v. 2, 2011.

REED, R.; MARKSII, R. J. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press, 1999. Library Catalog: mitpress.mit.edu Publisher: The MIT Press. Disponível em: <https://mitpress.mit.edu/books/neural-smithing>.

ROCHA, T. et al. *Tutorial sobre Fuzzy-c-Means e Fuzzy Learning Vector Quantization: abordagens híbridas para tarefas de agrupamento e classificação*. Institute of Informatics at Federal University of Rio Grande do Sul - UFRGS, Brazil: Revista de Informática Teórica e Aplicada, 2012.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. v. 65, n. 6, p. 386–408, 1958. ISSN 1939-1471(Electronic),0033-295X(Print).

ROSSUM, G. V.; DRAKE, F. L. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN 1441412697.

RUSSELL, S.; NORVIG, P. *Inteligêngia Artificial - Tradução da3ª Edição*. Rio de Janeiro, Brasil: Elsevier, 2013. ISBN 9788535237016.

RUUSKA, S. et al. Evaluation of the confusion matrix method in the validation of an automated system for measuring feeding behaviour of cattle. v. 148, p. 56–62, 2018. ISSN 0376-6357. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0376635717301146>.

SAMARATI, P.; VIMERCATI, S. C. de. Access Control: Policies, Models, and Mechanisms. In: FOCARDI, R.; GORRIERI, R. (Ed.). *Foundations of Security Analysis and Design*. Berlin, Heidelberg: Springer, 2001. (Lecture Notes in Computer Science), p. 137–196. ISBN 978-3-540-45608-7.

SANDHU, R.; SAMARATI, P. Access control: principle and practice. *IEEE Communications Magazine*, v. 32, n. 9, p. 40–48, set. 1994. ISSN 1558-1896. Conference Name: IEEE Communications Magazine.

SANDHU, R.; SAMARATI, P. Authentication, access control, and audit. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 28, n. 1, p. 241–243, mar 1996. ISSN 0360-0300.

SANTOS, A. M. R. C. d. Segurança nos sistemas de informação hospitalares : políticas, práticas e avaliação. 2007. Accepted: 2007-12-18T21:06:45Z. Disponível em: <http://repositorium.sdum.uminho.pt/>.

SANTOS, F. J. d. *Um Modelo Preditivo de Classificação de Operações de Crédito*. phdthesis — Instituto Federal de Educação do Ceará - IFCE, 2013.

SANTOS, T. G. S. *Google Colab: o que é e como usar? | Alura Cursos Online*. Artigos Alura: Acesso em 07 de junho de 2020, 2020. (Disponível em: https://www.alura.com.br/artigos/google-colab-o-que-e-e-como-usar). Library Catalog: www.alura.com.br.

SARKAR, D.; BALI, R.; SHARMA, T. *Practical Machine Learning with Python: A Problem-Solver's Guide to Building Real-World Intelligent Systems*. New York. NY, USA: Apress, 2017. ISBN 978-1-4842-3207-1.

SARKIS, L. C. *Uma abordagem para detecção de conflitos indiretos entre políticas de controle de acesso*. Tese de Doutorado — Universidade Federal Fluminense, 2017.

SARKIS, L. C.; SILVA, V. T. da; BRAGA, C. Detecting indirect conflicts between access control policies. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2016. (SAC '16), p. 1570–1572. ISBN 978-1-4503-3739-7. Disponível em: <http://doi.acm.org/10.1145/2851613.2851979>.

SHAZMEEN, S. F.; MUSTAFA, M. A.; BAIG, A. Performance Evaluation of Different Data Mining Classification Algorithm and Predictive Analysis. 2013.

SHEN, J.; SHAFIQ, M. O. Learning Mobile Application Usage - A Deep Learning Approach. In: *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. Boca Raton, FL, USA: [s.n.], 2019. p. 287–292.

SHOHAM, Y.; TENNENHOLTZ, M. On social laws for artificial agent societies: off-line design. v. 73, n. 1, p. 231–252, 1995. ISSN 00043702. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/000437029400007N>.

SHU, C.; YANG, E. Y.; ARENAS, A. E. Detecting conflicts in ABAC policies with rule-reduction and binary-search techniques. In: *2009 IEEE International Symposium on Policies for Distributed Systems and Networks*. [S.l.: s.n.], 2009. p. 182–185.

SILVA, B. M. d. *Uma Proposta de Controle de Acesso como Serviço para Federação de Identidades*. Monografia (Bacharelado em Engenharia de Software) — Bacharelado em Engenharia de Software, 2018.

SILVA, I. N. d. *Redes Neurais Artificiais Para Engenharia e Ciências Aplicadas. Fundamentos Teóricos e Aspectos Práticos*. Edição: 2ª. São Paulo, SP - Brasil: Artliber Editora, 2016. ISBN 978-85-88098-87-9.

SILVA, L. A. d.; PERES, S. M.; BOSCARIOLI, C. *Introdução à Mineração de Dados: Com Aplicações em R*. Elsevier Brasil, 2017. ISBN 9788535284478. Disponível em: <https://books.google.com.br/books?id=5LA4DwAAQBAJ>.

SILVA, L. C. e. *Aprendizado de máquina com treinamento continuado aplicado à previsão de demanda de curto prazo: o caso do restaurante universitário da Universidade Federal de Uberlandia*. Mestrado em Engenharia Elétrica — Universidade Federal de Uberlandia, 2019.

SILVESTRE, E. A. *Verificação de conflitos entre múltiplas normas em sistemas multiagentes*. Tese de Doutorado — UFF - Universidade Federal Fluminense, 2017.

SIMON, P. *Too Big to Ignore: The Business Case for Big Data*. Hoboken, New Jersey, USA: Wiley, 2013. 1256 p. (Wiley and SAS Business Series). ISBN 978-1-118-64186-6.

SLOMAN, M.; LUPU, E. Security and management policy specification. v. 16, n. 2, p. 10–19, 2002. ISSN 08908044. Disponível em: <http://ieeexplore.ieee.org/document/993218/>.

SOMAN, K.; LOGANATHAN, R.; AJAY, V. *Machine Learning with SVM and Other Kernel Methods*. India: Prentice-Hall of India Pvt.Ltd, 2009. ISBN 978-81-203-3435-9.

SUN, L. et al. *Privacy Preserving Access Control Policy and Algorithms for Conflicting Problems*. IEEE Computer Society, 2011. ISSN: 2324-898X Pages: 250-257. ISBN 978-1-4577-2135-9. Disponível em: <https://www.computer.org/csdl/proceedings-article/trustcom/2011/06120826/12OmNscOUhm>.

SWAMYNATHAN, M. *Mastering Machine Learning with Python in Six Steps: A Practical Implementation Guide to Predictive Data Analytics Using Python*. 2ª. ed. Apress, 2019. ISBN 978-1-4842-4947-5. Disponível em: <https://books.google.com.br/books?id=dEqzDwAAQBAJ>.

THENMOZHI, D.; KALAIVANI, A.; ARAVINDAN, C. Multi-lingual author profiling on SMS messages using machine learning approach with statistical feature selection. p. 9, 2018. Disponível em: <https://pdfs.semanticscholar.org/5d3e/7741c33bc7613fd4d5a7711d7f33830dc28f.pdf>.

TRAVASSOS, G. H.; GUROV, D. Introdução à engenharia de software experimental. Programa de Engenharia de Sistemas de Computação - COPPE/RJ, Rio de Janeiro/Brasil, p. 52, 2002.

TUKEY, J. *Exploratory Data Analysis*. Addison Wesley Publishing Company, 1970. (Exploratory Data Analysis). ISBN 978-0-608-08225-7. Disponível em: <https://books.google.com.br/books?id=AzOYwgEACAAJ>.

UEDA, E. T. *Análise de políticas de controle de acesso baseado em papéis com rede de Petri colorida*. Doutorado em Sistemas Digitais — USP - Universidade de São Paulo, 2012. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3141/tde-08032013-120904/>.

VASCONCELOS, L. M. R. d.; CARVALHO, C. L. d. Aplicação de regras de associação para mineração de dados na web. v. 1, n. 1, 2018. Number: 1. Disponível em: <https://telematicafractal.com.br/revista/index.php/telfract/article/view/8>.

VIJAYALAKSHMI, K.; JAYALAKSHMI, V. A priority-based approach for detection of anomalies in ABAC policies using clustering technique. In: *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*. [S.l.: s.n.], 2020. p. 897–903.

VIMERCATI, S. De Capitani di; SAMARATI, P.; JAJODIA, S. Policies, models, and languages for access control. In: BHALLA, S. (Ed.). *Databases in Networked Information Systems*. Aizu-Wakamatsu, Japan: Springer, 2005. (Lecture Notes in Computer Science), p. 225–237. ISBN 978-3-540-31970-2.

WANG, Y. et al. Conflicts analysis and resolution for access control policies. In: *2010 IEEE International Conference on Information Theory and Information Security*. Beijing, China: [s.n.], 2010. p. 264–267. ISSN: null.

WAZLAWICK, R. S. *Metodologia de Pesquisa em Ciência da Computação*. 1a edição. ed. Rio de Janeiro, RJ, Brasil: Elsevier, 2009. ISBN 978-85-35235-22-7.

WIEDERHOLD, G.; McCarthy, J. Arthur samuel: Pioneer in machine learning. v. 36, n. 3, p. 329–331, 1992. ISSN 0018-8646.

WITTEN, I. et al. *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier Science, 2016. (The Morgan Kaufmann Series in Data Management Systems). ISBN 978-0-12-804357-8. Disponível em: <https://books.google.com.br/books?id=1SylCgAAQBAJ>.

WOHLIN, C. et al. *Experimentation in Software Engineering: An Introduction*. Springer US, 2000. (International Series in Software Engineering). ISBN 978-1-4613-7091-8. Disponível em: <https://www.springer.com/gp/book/9781461370918>.

WOLPERT, D. H. The lack of a priori distinctions between learning algorithms. *Neural Comput.*, MIT Press, Cambridge, MA, USA, v. 8, n. 7, p. 1341–1390, oct 1996. ISSN 0899-7667. Disponível em: <http://dx.doi.org/10.1162/neco.1996.8.7.1341>.

WOLPERT, D. H.; MACREADY, W. G. Working Papers. *No Free Lunch Theorems for Search*. Santa Fe Institute, 1995. Disponível em: <https://EconPapers.repec.org/RePEc:wop:safiwp:95-02-010>.

WU, X. et al. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, Springer-Verlag New York, Inc., New York, NY, USA, v. 14, n. 1, p. 1–37, dec 2007. ISSN 0219-1377. Disponível em: <http://dx.doi.org/10.1007/s10115-007-0114-2>.

XU, Z.; STOLLER, S. D. Mining attribute-based access control policies from RBAC policies. In: *2013 10th International Conference and Expo on Emerging Technologies for a Smarter World (CEWIT)*. Melville, NY, EUA: [s.n.], 2013. p. 1–6.

XU, Z.; STOLLER, S. D. Mining attribute-based access control policies. Melville, NY, EUA, 2014. Disponível em: <http://arxiv.org/abs/1306.2401>.

YAHIAOUI, M.; ZINEDINE, A.; HARTI, M. Deconflicting policies in attribute-based access control systems. In: *2018 IEEE 5th International Congress on Information Science and Technology (CiSt)*. Marrakech, Morocco: [s.n.], 2018. p. 130–136. ISSN: 2327-1884.

YANDELL, B. *Practical Data Analysis for Designed Experiments*. London. GB: Taylor & Francis, 1997. (Chapman & Hall/CRC Texts in Statistical Science). ISBN 978-0-412-06341-1.

YEUNG, N.; BOTVINICK, M. M.; COHEN, J. D. The neural basis of error detection: conflict monitoring and the error-related negativity. *Psychological Review*, v. 111, n. 4, p. 931–959, out. 2004. ISSN 0033-295X.

ZHENG, G.; XIAO, Y. A research on conflicts detection in ABAC policy. In: *2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)*. Dalian, China: [s.n.], 2019. p. 408–412.