UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

KLEBER GONÇALVES FELIX

CLASSIFICAÇÃO AUTOMÁTICA DE FALHAS EM ARQUITETURA ORIENTADA A SERVIÇOS

DISSERTAÇÃO

CURITIBA
2017

KLEBER GONÇALVES FELIX

CLASSIFICAÇÃO AUTOMÁTICA DE FALHAS EM UMA ARQUITETURA ORIENTADA A SERVIÇOS

Dissertação apresentada ao Programa de Pós-graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de "Mestre em Ciências" – Área de Concentração: Sistemas Distribuídos.

Orientador: Prof. Dr. Mauro Sergio Pereira Fonseca

Coorientadora: Prof[a]. Dr[a]. Ana Cristina B. Kochem Vendramin

CURITIBA

2017

Dados Internacionais de Catalogação na Publicação

F316c
2017

Felix, Kleber Gonçalves
Classificação automática de falhas em uma arquitetura orientada a serviços / Kleber Gonçalves Felix.-- 2017.
90 f.: il.; 30 cm.

Disponível também via World Wide Web.
Texto em português, com resumo em inglês.
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Computação Aplicada. Área de Concentração: Sistemas Distribuídos, Curitiba, 2017.
Bibliografia: f. 74-79.

1. Sistemas operacionais distribuídos (Computadores). 2. Localização de falhas (Engenharia). 3. Controle automático. 4. Aprendizado do computador. 5. Arquitetura orientada a serviços (Computador). 6. Enterprise service bus (Software). 7. Algoritmos. 8. Métodos de simulação. 9. Computação - Dissertações. I. Vendramin, Ana Cristina Barreiras Kochem, orient. II. Fonseca, Mauro Sergio Pereira, coorient. III. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada. IV. Título.

CDD: Ed. 22 -- 621.39

Bibliotecária: Luiza Aquemi Matsumoto CRB-9/794

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Pesquisa e Pós-Graduação

ATA DA DEFESA DE DISSERTAÇÃO DE MESTRADO Nº 60

DISSERTAÇÃO PARA OBTENÇÃO DO TÍTULO DE MESTRE EM COMPUTAÇÃO APLICADA
PROGRAMA DE PÓS-GRADUAÇÃO EM: COMPUTAÇÃO APLICADA
ÁREA DE CONCENTRAÇÃO: CIÊNCIA DA COMPUTAÇÃO
LINHA DE PESQUISA: REDES E SISTEMAS DISTRIBUÍDOS.

No dia 29 de agosto de 2017 às 10h reuniu-se na Sala C301 da Sede centro a banca examinadora composta pelos pesquisadores indicados a seguir, para examinar a dissertação de mestrado do(a) candidato Kleber Gonçalves Felix**,** intitulada: Classificação Automática de falhas em arquitetura orientada a serviços.

Orientador(a): Prof. Dr. Mauro Sergio Pereira Fonseca

Co-orientador: Prof. Ana Cristina B. Kochem Vendramin

Após a apresentação, o(a) candidato(a) foi arguido(a) pelos examinadores que, em seguida à manifestação dos presentes, consideraram o trabalho de pesquisa: ( ) Aprovado. ( ) Aprovado com restrições. Revisor indicado para verificação: _____. ( ) Reprovado.

Observações:

_____
_____
_____

Nada mais havendo a tratar, a sessão foi encerrada às __h__, dela sendo lavrado a presente ata, que segue assinada pela Banca Examinadora e pelo Candidato.

O candidato está ciente que a concessão do referido título está condicionada à: (a) satisfação dos requisitos solicitados pela Banca Examinadora; (b) entrega da dissertação em conformidade com as normas exigidas pela UTFPR; (c) atendimento ao requisito de publicação estabelecido nas normas do Programa; e (d) entrega da documentação necessária para elaboração do Diploma. A Banca Examinadora determina um **prazo máximo de** _____ **dias**, considerando os prazos máximos definidos no Regulamento Geral do Programa, para o cumprimento dos requisitos (desconsiderar caso reprovado), sob pena de, não o fazendo, ser desvinculado do Programa sem o Título de Mestre.

Prof. Dr. Mauro Sergio P. Fonseca - Presidente - UTFPR _____

Prof. Dr. Carlos Marcelo Pedroso - UFPR _____

Prof. Dr. Luiz Nacamura Jr – UTFPR

Prof. Dr. Cesar Augusto Tacla - UTFPR _____

Assinatura do Candidato: _____

(reservado à Coordenação)

DECLARAÇÃO PARA A OBTENÇÃO DO TÍTULO DE MESTRE

A Coordenação do Programa declara que foram cumpridos todos os requisitos exigidos pelo Programa de Pós-Graduação para a obtenção do Título de Mestre.

Curitiba, _____de _____de 20___.

Carimbo e Assinatura do(a) Coordenador(a) do Programa

DEDICATÓRIA

Dedico este trabalho a minha esposa Elizangela, minha amiga e companheira eterna. Aos meus pais, que sempre acreditaram em mim e aos meus meninos Felipe e Davi: fonte de força e inspiração.

AGRADECIMENTOS

Concluir um mestrado sempre foi, para mim, mais do que um projeto profissional ou acadêmico é uma realização pessoal. Nessa caminhada que levou a elaboração do presente trabalho, tenho muito o que agradecer à várias pessoas.

Em primeiro lugar agradeço ao meu Pai Celestial pela oportunidade concedida de realizar esse mestrado, por me dar a força e disciplina necessária para conciliar minhas obrigações familiares, profissionais e eclesiásticas com esse programa.

Agradeço aos meus pais Dirceu Gonçalves Felix e Neusa Feijó Felix pelo amor, incentivo e sacrifícios empenhados na minha educação. Sou muito grato a minha amada esposa, Elizangela Felix, por compartilhar deste sonho comigo e pelo apoio incondicional: sem ele nada disso seria possível.

Agradeço aos meus filhos Felipe Felix e Davi Felix que mesmo tão novos, compreenderam o período dedicado às atividades do mestrado e abdicaram de alguns de nossos tradicionais passeios e diversão juntos. Também por frequentemente oferecerem um carinho extra enquanto estava em frente ao computador.

Fui muito abençoado pelos professores que meu conheci nessa caminhada, agradeço muito ao Prof. Dr. Mauro Sergio Pereira Fonseca por me aceitar como seu aluno, e por sua experiência e competência, que foram essenciais na finalização desse trabalho.

Agradeço muito a professora Drª. Ana Cristina B. Kochem Vendramin, que esteve comigo desde a entrevista inicial até a conclusão deste trabalho, por todo apoio, dedicação e correções, ganhando assim minha admiração, respeito e amizade.

Agradeço a professora Drª Anelise Munaretto Fonseca, que esteve presente em todos os seminários de qualificação e a todos os professores que direta ou indiretamente contribuíram para a realização desse trabalho: Prof. Dr. Cesar Augusto Tacla, Prof. Dr. Jean Marcelo Simão, Prof. Dr. Robson Ribeiro Linhares, Prof. Dr. Marco Aurélio Wehrmeister, Prof. Dr. Luiz Nacamura Jr. e Prof. Dr. Rodrigo Minetto, profissionais excelentes e dedicados que mostram que o Brasil tem ensino superior de muita qualidade.

A todos o meu eterno agradecimento.

RESUMO

FELIX, Kleber Gonçalves. CLASSIFICAÇÃO AUTOMÁTICA DE FALHAS EM UMA ARQUITETURA ORIENTADA A SERVIÇOS. 2017. 86 f. Dissertação – Programa de Pós-Graduação em Computação Aplicada, Universidade Tecnológica Federal do Paraná. Curitiba, 2017.

Uma arquitetura distribuída é composta de diversos sistemas que trocam mensagens entre si. Falhas na integração destes sistemas podem ocorrer, exigindo uma investigação detalhada dos profissionais de suporte para encontrar a causa raiz do problema. O processo manual de identificação de falhas é difícil e demorado. Ganhos significativos podem ser obtidos através da automação do processo de classificação de falhas. Este trabalho tem por objetivo apresentar um método para auxílio no processo de diagnóstico de falhas, classificando automaticamente as falhas geradas em uma arquitetura orientada a serviços. Este método, denominado SOAFaultControl, se beneficia de arquiteturas distribuídas que adotam SOA e um *Enterprise Service Bus* (ESB). Utilizando-se de técnicas de aprendizado de máquina, foi possível estabelecer um modelo para classificação de falhas em categorias preestabelecidas. Para alcançar o objetivo deste trabalho foi necessário testar e avaliar os seguintes algoritmos de aprendizagem de máquina: *Support Vector Machine*, *Naive Bayes* e *AdaBoost.* Como resultado, o algoritmo *Support Vector Machine* obteve melhor desempenho nas métricas: acurácia, precisão, revocação e F1.

Palavras-chave: Sistemas Distribuídos, Classificação de falhas, Aprendizado de Máquina, SOA, ESB.

ABSTRACT

A distributed architecture is composed of many systems that exchange messages between each other. Faults in the integration of these systems may occur and they required a detailed investigation of support professionals to identifying the root cause of the problem. The manual process to identify causes of failure is difficult and time-consuming. Significant efficiency gains can be achieved by automating the faults classification process. This work presents a method to support the automated fault diagnostic process, automatically classifying faults generated in a Service Oriented Architecture (SOA). This method denominated SOAFaultControl, may be executed in a distributed architecture that adote SOA and an Enterprise Service Bus (ESB). Using machine learning techniques, was possible build a model to classify fault messages captured in a SOA environment, in pre-established classes. To achieve the objectives of this work it was necessary to test the following machine learning algorithms: Support Vector Machine, Naive Bayes, and AdaBoost. Results show that Support Vector Machine algorithm achieved better performance in the following metrics: precision, accuracy, recall, and F1.

Keywords: Distributed Systems, Fault Classification, Machine Learning, SOA, ESB.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AM	Aprendizado de Máquina
BoW	*Bag of Words*
BPEL	*Business Process Execution Language*
DAO	*Data Access Object*
ESB	*Enterprise Service Bus*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
JMS	*Java Message Service*
JSON	*JavaScript Object Notation*
MD5	*Message-Digest algorithm 5*
NLTK	*National Language Tookit*
OSB	*Oracle Service Bus*
QA	*Quality Assurance*
REST	*Representational State Transfer*
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SNMP	*Simple Network Management Protocol*
SVM	*Support Vector Machine*
TAE	Teoria de Aprendizado Estatístico
TI	Tecnologia da Informação
URI	*Uniform Resource Identifier*
XML	*Extensible Markup Language*
WSDL	*Web Service Description Language*

LISTA DE SÍMBOLOS

f	Função capaz de classificar os dados de treinamento
k	Número de subconjuntos utilizados em um treinamento
VP	Número de exemplos positivos que foram corretamente classificados
FP	Número de exemplos negativos classificados como positivos
FN	Número de exemplos positivos classificados como negativos
VN	Número de exemplos negativos corretamente classificados

SUMÁRIO

1 INTRODUÇÃO

Um sistema distribuído é aquele no qual os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas enviando mensagens entre si (COULOURIS, 2005).

Sistemas distribuídos, em sua grande maioria, são heterogêneos. Grandes sistemas usam diferentes plataformas, linguagens e paradigmas de programação e mesmo *middlewares* diferentes. Muitas abordagens tem sido propostas para resolver o problema da heterogeneidade na integração distribuída, a *Service-Oriented Architecture* (SOA) é uma delas. SOA estabelece um modelo arquitetônico que visa aprimorar a eficiência, agilidade e produtividade de uma empresa, posicionando os serviços como os principais meios para que a solução lógica seja representada (ERL, 2009).

Falhas podem ocorrer durante todas as etapas SOA. O diagnóstico de falhas manual em um sistema distribuído é demorado e complexo. A razão principal, segundo REIDEMEISTER, MUNAWAR e WARD (2010), é a dimensão e complexidade desses sistemas e também a vasta quantidade de dados de monitoramento que estes sistemas podem gerar.

De acordo com REIDEMEISTER, MUNAWAR e WARD (2010) vários estudos indicam que a maioria das falhas em sistemas distribuídos são causadas por defeitos recorrentes e ganhos significativos podem ser obtidos através da automação do processo de classificação de falhas.

Falhas em sistemas distribuídos são inevitáveis e podem trazer graves consequências para as organizações como: perda de clientes, atraso no atendimento, prejuízos financeiros, entre outros. Estas consequências podem ser evitadas se o tratamento do problema for efetuado de forma rápida e correta. Quando uma falha nova é encontrada em um sistema ela deve ser rapidamente classificada (rotulada) e encaminhada para a equipe responsável.

Este trabalho propõe um método de classificação automática das falhas recebidas na troca de mensagens em uma arquitetura orientada a serviços, utilizando técnicas de aprendizagem de máquina e classificação de texto. O método é denominado neste trabalho como **SoaFaultControl**.

1.1 MOTIVAÇÃO

A ausência de métodos que classificassem mensagens de erros automaticamente em uma arquitetura orientada a serviços motivou esta pesquisa. Um método de classificação automática de falhas pode trazer vários benefícios em uma arquitetura orientada a serviços, como:

- A classificação das falhas pode ajudar a identificar a origem de um problema (CHAN, BISHOP, *et al.*, 2009). É importante que a causa raiz de uma falha possa ser classificada e reparada o mais rápido possível. Se o sistema pode distinguir entre os diferentes tipos de falhas, pode-se então começar a aplicar um método correto de recuperação;
- Mecanismos podem ser desenvolvidos para monitorar uma arquitetura orientada a serviços e recuperar as falhas automaticamente. Um fator chave no processo de auto recuperação é saber qual falha procurar. Se a natureza de uma falha é conhecida, o sistema pode sugerir um mecanismo de recuperação adequado o mais rápido possível (CHAN, BISHOP, *et al.*, 2009);
- Falhas que não são recuperadas automaticamente devem ser encaminhadas para tratamento direto de uma equipe específica, como exemplo, equipes de suporte ao *middleware*, redes de computadores, banco de dados, sistemas operacionais, entre outras. Exceções de negócios devem ser encaminhadas para as áreas de suporte a um determinado seguimento como faturamento, vendas, operações de campo, entre outras. Uma classificação automática auxilia neste processo, ganhando agilidade na resolução da falha.

1.2 OBJETIVOS

O objetivo principal desta tese é desenvolver uma arquitetura que utilize técnicas de aprendizagem de máquina para automatizar a tarefa de classificação de falhas em uma arquitetura orientada a serviços

Além do objetivo principal, este trabalho tem como objetivos específicos:

1) Desenvolver um método de pré-processamento das amostras de mensagens de falha para otimizar a eficiência dos algoritmos de classificação;

2) Apresentar uma abordagem integrada de captura e classificação de falhas em uma arquitetura orientada a serviços;
3) Prover uma integração do método SOAFaultControl com os usuários do sistema;
4) Testar e avaliar algoritmos de aprendizagem de máquina aplicados na tarefa de classificação de mensagens de falha com o objetivo de selecionar a técnica mais eficiente para essa função, de acordo com os experimentos e métricas de avaliação apresentados neste trabalho.

1.3 ESTRUTURA DO TRABALHO

Os próximos capítulos deste documento são compostos da seguinte forma: O capítulo 2 apresenta os principais conceitos utilizados neste trabalho com o objetivo de familiarizar o leitor com os termos e técnicas apresentados. Também apresenta estudos sobre classificação de texto, aprendizagem de máquina, detecção de falhas em sistemas distribuídos e taxonomias de falhas em uma arquitetura orientada a serviços. O capítulo 3 descreve o método de pesquisa utilizado para o desenvolvimento do trabalho. O capítulo 4 apresenta a arquitetura do método **SoapFaultControl**. O capítulo 5 apresenta os experimentos realizados. O capítulo 6 conclui este trabalho, discute os resultados e apresenta trabalhos futuros relacionados ao tema.

2 FUNDAMENTOS TEÓRICOS

Neste capítulo são discutidos os conceitos básicos ao entendimento deste trabalho, abrangendo SOA, web services, ESB, conceitos de falha e aprendizagem de máquina.

2.1 SOA

Para COULOURIS (2005), SOA é um conjunto de princípios de projeto por meio do qual os sistemas distribuídos são desenvolvidos usando-se conjuntos de serviços pouco acoplados que podem ser descobertos dinamicamente e, então, comunicar-se uns com os outros, ou que são coordenados por meio de coreografia para fornecer serviços aprimorados.

Para JOSUTTIS (2008), SOA é um paradigma para a realização e manutenção de processos de negócios, não é uma arquitetura concreta. É algo que leva a uma arquitetura completa. Ou seja, SOA não é uma ferramenta ou estrutura que pode ser comprada. É uma abordagem, uma maneira de pensar, um sistema de valores que leva a certas decisões concretas na concepção de uma arquitetura de software concreta.

Segundo JOSUTTIS (2008), SOA baseia-se em três grandes conceitos técnicos:

- **Serviços:** módulo de negócios independente. A funcionalidade pode ser simples (armazenamento ou recuperação de dados do cliente), ou complexa (um processo de negócio para uma ordem de um cliente). Como os serviços concentram-se no valor comercial de uma interface, eles preenchem a lacuna de negócio e TI (*Tecnologia da Informação*);
- **Alta interoperabilidade:** ao se trabalhar com sistemas heterogêneos, o primeiro objetivo deve ser a capacidade de conectar estes sistemas facilmente. Isto é chamado de alta interoperabilidade. Alta interoperabilidade não é uma ideia nova. No entanto, para SOA, alta interoperabilidade é o começo, não o fim. É a base pela qual se começa a implementar a funcionalidade de negócios (serviços), que está espalhada por múltiplos sistemas distribuídos;
- **Baixo acoplamento:** conceito destinado a reduzir as dependências entre os diferentes sistemas. Quando dependências são minimizadas, o impacto nas modificações no sistema tem seu efeito minimizado, e o sistema ainda é executado, mesmo quando alguma parte está indisponível ou apresentando problemas. Minimizar dependências

contribui para a tolerância de falhas e flexibilidade, que é exatamente o que se precisa na implementação de sistemas distribuídos.

Em SOA composições de serviços são descritas, segundo SAUDATE (2014), como orquestrações ou coreografias. Uma coreografia é uma composição de serviços descentralizada. Na coreografia, cada serviço participante deve saber especificamente quais ações executar. Já na orquestração existe um controlador centralizado, dizendo a cada serviço a maneira de prosseguir com a troca de dados.

Entre as duas formas, SAUDATE (2014) cita a orquestração como a forma mais adotada pelo mercado. Ela pode ser implementada através do *Business Process Execution Language* (BPEL), que é uma linguagem especificamente criada com o propósito de coordenar a comunicação entre vários serviços.

2.2 WEB SERVICES

Um *webservice* é uma maneira padronizada de distribuir serviços através de uma rede de computadores. Os clientes não têm conhecimento do serviço antes que eles realmente comecem a utilizá-los. Portanto, *web services* são independentes da linguagem de programação e fracamente acoplados (MUMBAIKAR; PADIYA et al., 2013).

Dois padrões de *web services* são amplamente utilizados:

- ***Simple Object Access Protocol (SOAP):*** a comunicação entre as entidades é baseada na linguagem de marcação *Extensible Markup Language* (XML). As mensagens SOAP são compostas por um envelope que é puramente um *container* para os elementos *Header* e *Body*. O elemento *Body* contém o corpo da requisição: o nome da operação, parâmetros, etc. O elemento *Header* contém metadados pertinentes à requisição. Um dos elementos do SOAP é seu contrato, chamado *Web Service Description Language* (WSDL).
- ***Representational State Transfer (REST):*** os *web services RESTful* seguem o princípio REST para sistemas hipermídios distribuídos. A arquitetura REST utiliza o protocolo *Hypertext Transfer Protocol* (HTTP) através dos métodos *GET*, *PUT*, *POST* e *DELETE* para recuperar, criar, atualizar e excluir os recursos. Um recurso é

uma funcionalidade que é identificado por uma *Uniform Resource Identifier* (URI) (MUMBAIKAR; PADIYA et al., 2013).

O presente trabalho fará a classificação de falhas em mensagens provenientes de *web services* SOAP.

2.3 ESB

De acordo com JOSUTTIS (2008) ao se trabalhar com sistemas heterogêneos, o primeiro objetivo deve ser a capacidade de conectar estes sistemas facilmente. Isto é chamado de "alta interoperabilidade". Um *Enterprise Service Bus* (ESB) é a parte técnica da SOA que permite alta interoperabilidade.

De acordo com Papazoglou e Heuvel (2007), um ESB exibe duas características proeminentes:

- Promove o baixo acoplamento entre as partes de um sistema integrado;
- Facilita a integração lógica entre diferentes sistemas;

A principal função de um ESB, de acordo com JOSUTTIS (2008), é permitir a chamada de serviços entre sistemas heterogêneos e suas responsabilidades incluem:

- **Fornecimento de conectividade:** integrar diferentes plataformas de hardware e software, mesmo diante de diferentes *middlewares* e protocolos;
- **Transformação de dados:** um dos principais papéis do ESB é prover a interoperabilidade, pois realiza integração de diferentes plataformas e linguagens de programação. Parte fundamental deste papel é a transformações de dados. A mediação realizada pelo ESB, entre diferentes serviços, permite transformar os dados de um formato para outro;
- **Roteamento de serviços inteligente:** o ESB provê os mecanismos necessários para enviar uma chamada de serviço de um consumidor a um provedor e, em seguida, enviar uma resposta de volta a partir do provedor para o consumidor;
- **Lidar com segurança:** um ESB pode fornecer formas de lidar com os diferentes aspectos da segurança;

- **Lidar com confiabilidade:** o ESB trabalha com diferentes tipos de protocolos. Protocolos diferentes oferecem diferentes formas de confiabilidade. Nem todos os protocolos garantem que uma mensagem será entregue "uma vez e apenas uma vez". Por esta razão, um ESB deve definir como lidar com problemas de confiabilidade, falhas e erros técnicos;
- **Gerenciamento de serviços:** permite que os analistas e projetistas encontrem dentre os serviços existentes aqueles que sejam reutilizáveis para a construção de novos processos de negócio;
- **Monitoramento e *logging*:** o ESB se torna o depurador para processos distribuídos em uma empresa que adota SOA como arquitetura. Por esta razão, deve se estabelecer conceitos como IDs de correlação e ferramentas que suportam o monitoramento e depuração sobre sistemas distribuídos.

A Figura 1 ilustra uma arquitetura simplificada de um ESB, integrando aplicações escritas em diferentes plataformas e integradas ao barramento através de serviços de mensagem, como o *Java Message Service* (JMS), *Web services*, interfaces com aplicativos herdados, adaptadores e banco de dados.

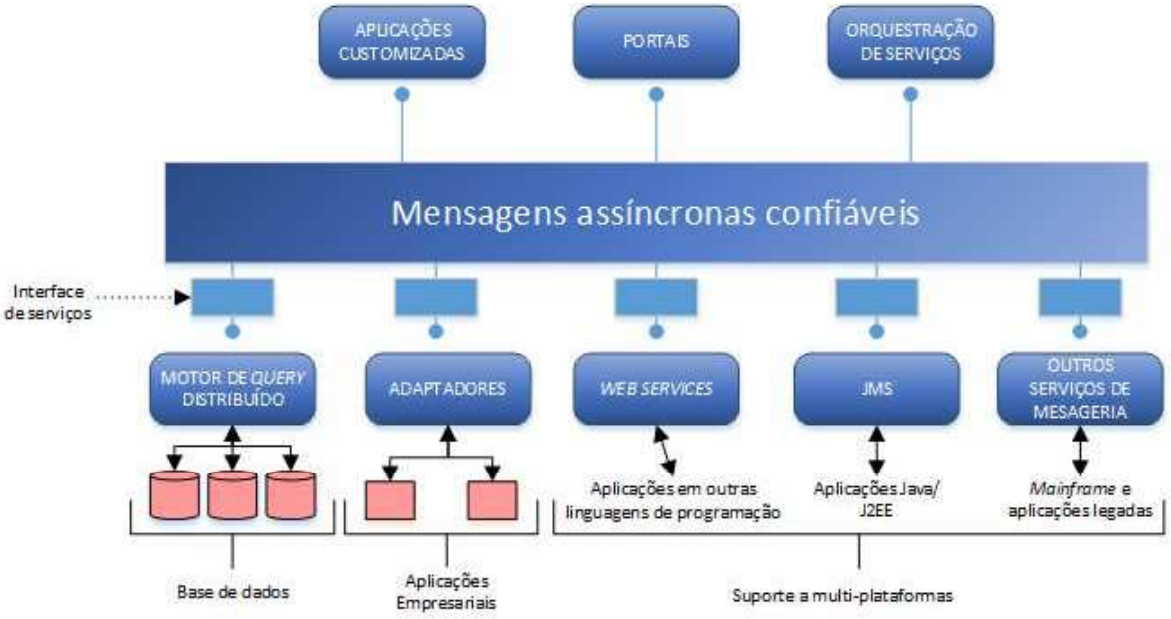

Figura 1 – Arquitetura simplificada de um ESB. Fonte: adaptado de Papazoglou e Heuvel (2007)

Um ESB, como retratado na Figura 1, agrega valor para a integração de diferentes componentes, posicionando-os atrás de uma fachada orientada a serviços fornecida pelo barramento e comunicando através de *web services*.

Um ESB centraliza os artefatos utilizados na comunicação, o processo de comunicação é feito através de dois tipos de serviços:

- ***Business Service:*** a comunicação feita com cada serviço é realizada através de um Serviço de Negócio. Um *business service* além das diversas configurações do serviço, carrega outras funcionalidades como segurança, codificação da comunicação, balanceamento de carga, monitoramento, etc;
- ***Proxy Services:*** os mecanismos de roteamento, transformação de dados, enriquecimento de mensagens e outras técnicas são incorporados pelos serviços de p*roxy*. Um serviço de *proxy* recebe a requisição e pode efetuar diversos tipos de filtragens, transformação e enriquecimento até finalmente entregá-la a um ou mais serviço de negócio.

STATION (2017) apresenta uma pesquisa com profissionais de Tecnologia da Informação, comparando os melhores fornecedores de ESB do mercado, com base em nas avaliações, classificações e comparações de produtos. Os produtos foram avaliados e classificados pelos profissionais registrados e validados por um processo de autenticação. A Tabela 1 apresenta os resultados desta pesquisa.

Posição	Produto	Fornecedor
1	MULE ESB	MuleSoft
2	*WebMethods*	Software AG
3	***Oracle Service Bus***	**Oracle**
4	WSO2 *Carbon*	WSO2
5	IBM *Integration Bus*	IBM
6	IBM *WebSphere Data Power*	IBM
7	IBM *WebSphere Message Broker*	IBM
8	TIBCO *ActiveMatrix Service Bus*	TIBCO
9	*Red Hat Fuse* ESB	Red Hat

Tabela 1 – Resultado de uma comparação entre os fornecedores de ESB.

Fonte: STATION (2017)

Neste trabalho o ESB utilizado na implementação do exemplo será o *Oracle Service Bus* (OSB), o método, porém, poderá ser adaptado a qualquer outro ESB do mercado.

2.4 CONCEITOS DE FALHA, ERRO E DEFEITO

Segundo WEBER (2003), um determinado sistema de computação é desenvolvido para atendimento da sua especificação. Um **defeito** (*failure*) é definido como um desvio da especificação. Defeitos não podem ser tolerados e devem ser evitados. Um sistema está em **estado errôneo** se o processamento posterior a partir desse estado levar a um defeito. Finalmente, define-se **falha** (ou falta) como a causa física ou algorítmica do erro.

A Figura 2 apresenta uma simplificação destes conce itos. Segundo WEBER (2003), falhas estão associadas ao universo físico, erros ao universo da informação e defeitos ao universo do usuário. É interessante observar que uma falha não necessariamente leva a um erro e um erro não necessariamente conduz a um defeito.

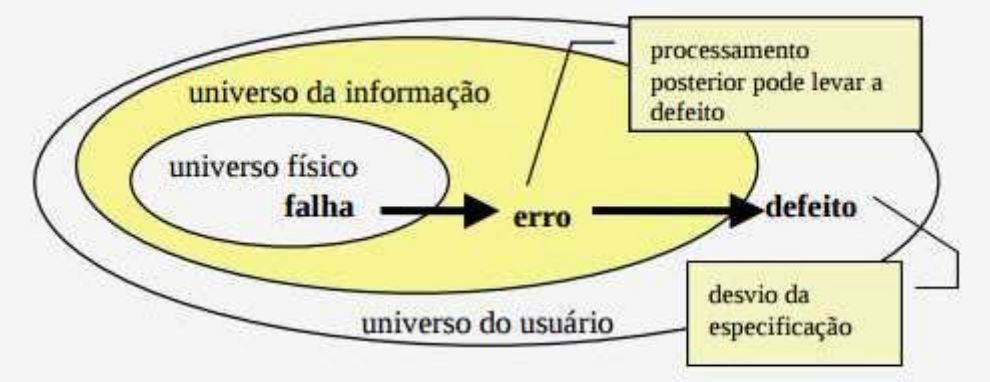

Figura 2 - Modelo de três universos: falha, erro e defeito. Fonte: WEBER (2003)

Um defeito por estar relacionado a uma exceção de uma regra de negócio. Uma regra de negócio é uma diretiva específica, acionável e estável que está sob controle de uma organização e que apoia uma política de negócio (LAGUNA, 2011). Cada regra de negócio pode gerar uma exceção. Por exemplo, se a regra de negócio para pedir livros estabelece que o valor de um pedido não pode ser menor do que R$ 50,00, então quando um pedido for finalizado ocorre uma exceção se ele não atingir este valor. Para lidar com essa exceção, será necessário adicionar um ou mais livros ao pedido, ou ele será cancelado (WAZLAWICK, 2016).

Segundo WAZLAWICK (2016), uma exceção no sentido usado em ciência da computação não é necessariamente um evento que ocorre raramente, mas um evento que se não for tratado evita que o processo continue. Uma exceção não impede um processo de iniciar, mas é normalmente algo que impede o processo de ser concluído.

2.5 APRENDIZADO DE MÁQUINA

De acordo com MITCHELL (1997), o Aprendizado de Máquina (AM) é um campo da inteligência computacional que estuda o desenvolvimento de técnicas capazes de extrair conceitos a partir de amostras de dados.

Segundo LORENA e DE CARVALHO (2007), as técnicas de AM empregam um princípio de inferência denominado indução, no qual obtém-se conclusões genéricas a partir de um conjunto particular de exemplos.

O aprendizado de máquina pode ser distinguido em três casos, conforme definido por RUSSELL e NORVIG (1995):

- **Aprendizado supervisionado:** envolve aprender uma função a partir de exemplos de suas entradas e saídas;
- **Aprendizado não-supervisionado:** envolve aprender padrões de entradas sem que haja o fornecimento de valores de saídas especificadas;
- **Aprendizado por reforço:** envolve aprender através de um retorno indicativo que um determinado comportamento não é desejável, o que implica um subproblema de aprender como o ambiente funciona.

LORENA e DE CARVALHO (2007) define o aprendizado supervisionado: *"Dado um conjunto de exemplos rotulados na forma:* (x_i, y_i) *em que* x_i *representa um exemplo e* y_i *denota o seu rótulo, deve-se produzir um* ***classificador****, também denominado modelo ou hipótese, capaz de predizer precisamente o rótulo de novos dados. Esse processo de indução de um classificador a partir de uma amostra de dados é denominado* ***treinamento****. O classificador obtido também pode ser visto como uma função* f *a qual recebe um dado* x *e fornece uma predição* y*".*

As técnicas de AM devem possuir a capacidade de lidar com dados imperfeitos, denominados ruídos. Deve-se também minimizar a influência de *outliers* no processo de

indução. Os *outliers* são exemplos muito distintos dos demais presentes no conjunto de dados. Esses dados podem ser ruídos ou casos muito particulares, raramente presentes no domínio.

2.5.1 MÁQUINA DE VETORES SUPORTE *(SUPPORT VECTOR MACHINE)*

Support Vector Machine (SVM) é um algoritmo supervisionado de aprendizagem de máquina, desenvolvido por VAPNIK (1995). A técnica toma como entrada um conjunto de dados e prediz, para cada entrada, de qual das duas classes a entrada faz parte através do reconhecimento de padrões. Esta técnica originalmente desenvolvida para classificação binária, busca encontrar o hiperplano de separação ideal o qual maximiza a margem da base de treinamento. (GONÇALVES, 2007).

Este método é fundamentado na teoria da aprendizagem estatística (GONÇALVES, 2007). A Teoria de Aprendizado Estatístico (TAE) visa estabelecer condições matemáticas que permitam a escolha de um classificador *f* com bom desempenho para os conjuntos de treinamento e teste. Ou seja, busca-se uma função *f* capaz de classificar os dados de treinamento da forma mais correta possível, sem dar atenção a qualquer ponto individual do mesmo (LORENA e DE CARVALHO, 2003). Resumindo, a teoria busca encontrar um bom classificador levando em consideração todo o conjunto de dados, porém se abstendo de casos particulares.

Segundo LORENA e DE CARVALHO (2003), as *SVMs* são originalmente utilizadas para a classificação dos dados em duas classes distintas, denominada **classificação binária**. Porém como muitas aplicações envolvem o agrupamento em mais de duas classes, é possível empregar diversas técnicas para estendê-las para uma **classificação multiclasses**.

Duas abordagens usuais são utilizadas para realização dessa tarefa “Um-Contra-Todos" (*OneVsRest*) e “Um-Contra-Um" (*OneVsOne*) (DOSCIATTI, 2015).

As estratégias *OneVsRest* e *OneVsOne* são as duas estratégias mais populares para uma classificação SVM multiclasses (MILGRAM, 2006).

A estratégia *OneVsRest* consiste em construir um SVM por classe, que é treinada para distinguir as amostras de uma classe das amostras de todas as classes restantes. Normalmente, a classificação de um padrão desconhecido é feita de acordo com o número máximo saídas entre todos os SVMs (MILGRAM, 2006).

A estratégia *OneVsOne* consiste em construir um SVM para cada par de classes. Assim, um problema com classes *C*, *C (C-1) / 2* SVMs são treinadas para distinguir as amostras de uma classe das amostras de outra classe. Geralmente, classificação de um padrão desconhecido é feito de acordo com a votação máxima, onde cada SVM vota em uma classe (MILGRAM, 2006).

2.5.2 CLASSIFICAÇÃO BAYESIANA (*NAIVE BAYES*)

O modelo de classificação *Naive Bayes* é uma técnica probabilística de classificação baseada no teorema de *Bayes*. Segundo o teorema é possível encontrar a probabilidade de um certo evento ocorrer, dada a probabilidade de um outro evento que já ocorreu (CAMILO e SILVA, 2009). Por exemplo, uma fruta pode ser considerada uma maçã se for vermelha, redonda e aproximadamente com de 3 polegadas de diâmetro, mesmo que estes recursos dependam um do outro ou da existência dos outros recursos, todas essas propriedades contribuem de forma independente para a probabilidade de que essa fruta seja uma maçã e é por isso que ela é conhecida como *Naive* (ingênuo) (RAY, 2017). Apesar desta premissa, classificadores *Naive Bayes* obtiveram resultados compatíveis com os métodos de árvore de decisão e redes neurais.

Devido a sua simplicidade e o alto poder preditivo, é um dos algoritmos mais utilizados (CAMILO e SILVA, 2009).

O modelo *Naive Bayes* é particularmente útil para conjuntos de dados muito grandes (RAY, 2017).

O teorema de *Bayes* fornece uma maneira de calcular a probabilidade posterior *P (c | x)* de *P (c)*, *P (x)* e *P (x | c)*. Conforme apresentado na Equação 1 (RAY, 2017):

$$P(c|x) = \frac{P(x|c)\,P(c)}{P(x)} \tag{1}$$

De acordo com a Equação 1:

- *P(c|x)* é a probabilidade posterior da classe (*c*, alvo) dado um preditor (x, atributos);
- *P(c)* é a probabilidade anterior da classe;
- *P(x|c)* é a probabilidade de preditores sejam dadas a classe;
- *P(x)* é a probabilidade anterior do preditor.

RAY (2017) cita os pontos positivos e negativos do modelo *Naive Bayes.* Como pontos positivos são destacados: um modelo fácil e rápido prever a classe de um conjunto de dados testado, o bom funcionamento para uma previsão multiclasses e os excelentes resultados comparados a outros modelos como a regressão logística, se mantida a independência de suposição. Como ponto negativo é destacado: "se a variável categórica tiver uma categoria no conjunto de testes que não foi observada no conjunto de treinamento, o modelo atribuirá uma probabilidade de 0 (zero) e será incapaz de fazer uma previsão".

2.5.3 CLASSIFICAÇÃO BASEADA EM CONJUNTOS (*ADABOOST*)

Segundo ROKACH (2010), a ideia principal por trás da metodologia baseada em conjuntos é pesar vários classificadores individuais e combiná-los para obter um classificador que supera todos. De fato, o ser humano tende a buscar várias opiniões antes de tomar qualquer decisão importante. Nós pesamos as opiniões individuais, e realizamos uma combinação para chegar à nossa decisão final.

Essa metodologia também é conhecida sob vários outros nomes: sistemas de classificação múltipla, comitê de classificadores ou mistura de especialistas. De acordo com POLIKAR (2006), os sistemas baseados em conjunto demonstraram produzir resultados favoráveis em comparação com os sistemas de um único perito para uma ampla gama de aplicações.

Boosting é um método para melhorar o desempenho de um classificador fraco. O método funciona executando repetidamente um conjunto de classificadores denominados fracos, modelos que são apenas um pouco melhores do que uma adivinhação aleatória como pequenas árvores de decisão, em vários dados de treinamento. Os classificadores produzidos são combinados em um único classificador forte a fim de obter uma maior precisão do que os classificadores fracos teriam tido (ROKACH, 2010). Seu funcionamento é ajustado de acordo com os erros cometidos pelo classificador anterior (CHAVES, 2012).

O *AdaBoost* ou *Adaptive Boosting* é um dos algoritmos baseados em *Boosting* mais famoso e utilizado. Esta técnica não é utilizada de forma isolada, sendo adaptada a outras técnicas como Árvores de Decisão e Redes Neurais.

O diferencial do *Boosting* é a busca por um classificador melhor e mais adequado para o problema, principalmente por corrigir e aumentar a eficiência dos classificadores gerados de forma isolada (CHAVES, 2012). A técnica *Boosting* gera conjuntos de treinos e

classificadores de forma sequencial, baseados nos resultados da iteração anterior, conforme Figura 3.

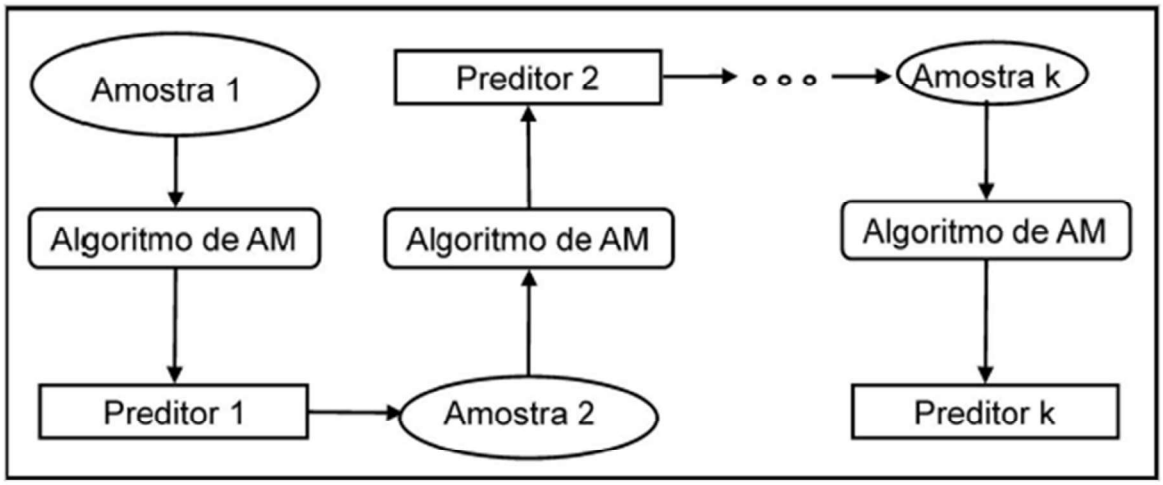

Figura 3 – *Boosting*: Geração sequencial de classificadores. Fonte: (CHAVES, 2012).

2.5.4 VALIDAÇÃO CRUZADA (*CROSS VALIDATION*)

O método de Validação Cruzada (*Cross Validation*) ou *K-Fold* é uma técnica de validação de algoritmos de aprendizagem de máquina utilizado para avaliar como o algoritmo de aprendizagem se comporta ao receber conjuntos de dados independentes.

Neste método os dados são divididos em uma parte para treinamento e outra parte para teste. Segundo HAN, PEI e KAMBER (2011), o método consiste em particionar aleatoriamente os dados em k subconjuntos mutuamente exclusivos, de tamanhos aproximadamente iguais. Com isso, o treinamento é realizado k vezes, a cada vez deixando um dos grupos para teste. Se $k = 5$, o modelo será treinado cinco vezes, na primeira vez o primeiro grupo será usado para teste e os outros quatro serão usados para treinamento. Na segunda vez, o segundo grupo será usado para teste e os outros quatro serão usados para treinamento, e assim por diante. O resultado final é uma matriz, onde é apontada a média de acertos e erros de cada classe, conforme apresentado na Figura 4.

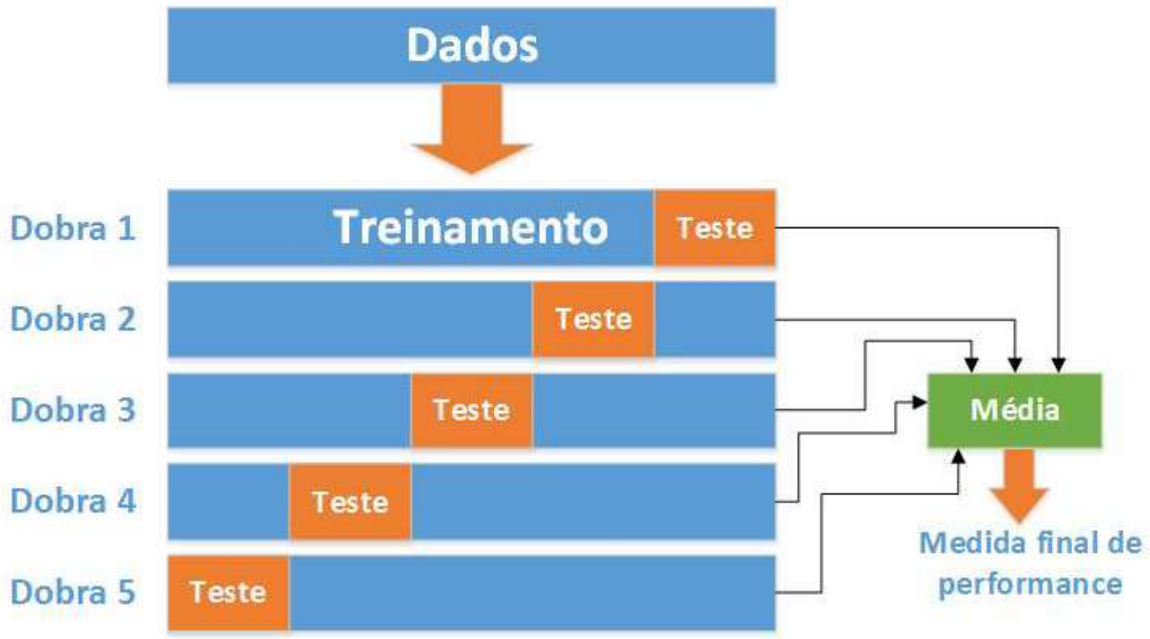

Figura 4 - Exemplo de *Cross Validation* para $k = 5$. Fonte: adaptado de KACMAJOR (2016)

2.6 ESTADO DA ARTE

Após a definição do problema uma revisão da literatura foi realizada sobre o tema abordado. Esta seção apresenta os estudos de maior relevância, relacionados a este trabalho, apresentados de forma estruturada e brevemente explicados.

A pesquisa do estado da arte focou em três assuntos: automação do processo de detecção de falhas em sistemas distribuídos, técnicas de aprendizagem de máquina para classificação de dados e taxonomias de falhas em arquitetura orientada a serviços.

2.6.1 DETECCÃO DE FALHAS EM SISTEMAS DISTRIBUÍDOS

REIDEMEISTER, MUNAWAR e WARD (2010), apresentam um método para a identificação de sintomas de falhas recorrentes em sistemas distribuídos. A abordagem utiliza técnicas de aprendizagem de máquina para prever probabilidades de falhas, com base nos padrões observados. Também utiliza árvores de decisão que incluem regras humanas para deduzir sintomas de falhas recorrentes. O método utiliza amostras de arquivos de *logs*, previamente marcados, ou seja, arquivos de sistemas que contenham uma única falha conhecida, para treinar o classificador. O classificador exige arquivos de *log* modelados em

um formato padronizado. O trabalho mostrou que é viável classificar arquivos de *log* simples para a identificação de falhas sem a necessidade de atributos pré-existentes específicos mantendo uma boa precisão de classificação.

O projeto *LogCluster* de VAARANDI e PIHELGAS (2015), propõe um algoritmo de agrupamento de dados, cujo objetivo é descobrir frequência de eventos atípicos em registros de *logs* de eventos dos sistemas distribuídos. Como resultado o algoritmo pode gerar modelos de entrada de *log* com base nos padrões identificados que podem auxiliar o diagnóstico de sistemas anômalos.

A ferramenta DISTALYZER proposta por NAGARAJ, KILLIAN e NEVILLE (2012), é focada na investigação de problemas de desempenho em sistemas distribuídos. O projeto implementa uma técnica para o diagnóstico: compara dois conjuntos de dados, um com um bom desempenho outro com desempenho ruim, assim através de técnicas de aprendizado de máquina, compara o comportamento dos registros extraídos e extrai um diagnóstico automático de desempenho, com base nas associações realizadas entre os componentes.

O projeto Draco de KAVULYA, DANIELS, *et al.* (2012) utiliza análise *bayesiana* para localizar problemas crônicos em sistemas distribuídos através da comparação de requisições bem-sucedidas e com falha, o estudo obteve como resultado final uma precisão de menos de 4% de falsos positivos.

PHAM, WANG, *et al.* (2017) introduz uma abordagem para automatizar o diagnóstico de falhas em sistemas distribuídos, combinando injeção de falhas e análise de dados. A injeção de falhas é feita preenchendo um banco de dados com as falhas geradas por um sistema distribuído de destino. Quando uma falha é relatada no ambiente de produção, o banco de dados é consultado para encontrar falhas "combinadas" geradas por injeções de falha. Baseando-se na suposição de que falhas semelhantes geram falhas semelhantes, a abordagem utiliza informações das falhas correspondentes como dicas para localizar a causa real das falhas relatadas. Como resultado utiliza o método para identificar as causas reais de falhas nos sistemas observados.

2.6.2 CLASSIFICAÇÃO DE DADOS

CABRAL (2017) apresenta em seu trabalho uma pesquisa sobre a utilização de redes neurais artificiais para tratar sinais provenientes da monitoração de desgastes das peças

de sistemas eletromecânicos. O objetivo é criar um sistema de detecção e classificação de falhas automatizado, não supervisionado, aplicado à manutenção preventiva e preditiva de processos eletromecânicos. Foi possível verificar nos resultados que modos de falha foram detectados e classificados pelo sistema com 99,7% de precisão.

NISA e QAMAR (2015) propõem um método de mineração de texto para classificar automaticamente os serviços para domínios específicos, identificando conceitos-chave dentro da documentação textual de serviço. Esta abordagem foi validada em um conjunto de 600 serviços web, categorizados em 8 campos que forneceram uma precisão até 90%.

XUAN, JIANG, *et al.* (2017) propõem uma abordagem de classificação semi-supervisionada para a triagem de erros em projetos de desenvolvimento de *software*. A abordagem combina um classificador *Bayesiano* e técnicas de aprendizagem de máquina para classificar e rotular um erro corretamente, criar um relatório de erros e enviar para o desenvolvedor correto ganhando produtividade. Os resultados experimentais demonstram que a abordagem semi-supervisionada melhorou a precisão de classificação e da triagem de erros em até 6% de precisão em relação a triagem manual.

KAZEMIAN e AHMED (2015) apresentam uma comparação de técnicas de aprendizado de máquina para detectar páginas da *Web* mal-intencionadas. O trabalho utiliza três técnicas de aprendizagem supervisionada: *K-Nearest Neighbor*, *SVM* e *Naive Bayes* e duas técnicas de aprendizagem de máquina não supervisionadas: *K-Means* e *Affinity Propagation,* as técnicas não supervisionadas apresentaram uma precisão de até 98%.

HU, GAO, *et al.* (2014) apresenta algoritmos de detecção de ataques em redes de computadores baseados no modelo *Adaboost*. A técnica se beneficia de classificadores e mineração de dados com o objetivo de construir um modelo eficaz na detecção de intrusão em rede. Os algoritmos apresentaram precisão acima de 90%.

ANDRADE (2015) apresenta uma pesquisa de um modelo para a triagem automática de denúncias na Controladoria Geral da União (CGU), o órgão do Poder Executivo responsável pelas atividades de auditoria pública do Poder Executivo. Após cadastradas pelo cidadão as denúncias devem ser triadas e encaminhadas para a coordenação temática da CGU com competência para realizar a apuração. Atualmente essa triagem é feita de forma manual e a denúncia encaminhada para uma dentre as 91 opções de destino pré-determinadas. Essa grande quantidade de categorias é um fator que dificulta a classificação automática de textos. As técnicas de classificação de texto obtiveram uma precisão de até 84%.

SUN, FUJITA, *et al.*(2017) apresentam um modelo de previsão dinâmica de dificuldades financeiras para melhorar a gestão de riscos financeiros corporativos. O modelo utiliza as técnicas *Adaboost* e *SVM*. O experimento empírico foi realizado com dados de amostra de rateios financeiros de 932 empresas chinesas. Os resultados experimentais mostram que a técnica Adaboost/SVM se mostrou adequada para a previsão financeira proposta no trabalho.

DOSCIATTI (2015) apresenta um método que permite identificar as emoções básicas em textos, além de identificar também textos neutros (sem emoção). O método utiliza exclusivamente uma abordagem de Aprendizagem de Máquina supervisionada para classificar os textos. O método opera em camadas, sendo que na primeira camada utiliza um classificador SVM multiclasse e o conceito de rejeição para rejeitar os textos mais complexos de serem classificados. O método obteve uma precisão de acerto de 65,5% ao identificar as seis emoções básicas, além de neutro, em textos.

2.6.3 TAXONOMIAS DE FALHAS EM UMA ARQUITETURA ORIENTADA A SERVIÇOS

Uma taxonomia de falhas na Arquitetura Orientada a Serviços é apresentada por BRUNING, WEISSLEDER e MALEK (2007). As classes desta taxonomia são fundamentadas nas etapas típicas de SOA:

- **Falhas de publicação:** falhas que podem ocorrer quando a descrição do serviço está incorreta ou não corresponder ao serviço;
- **Falhas de descoberta:** falha que pode ocorrer quando o serviço requisitado não existe ou não está listado no serviço de pesquisa;
- **Falhas de composição:** falha que pode ocorrer devido a várias razões, como exemplo, componentes incompatíveis que não podem ser conectados;
- **Falhas de ligação:** falhas ocorridas no processo de junção entre o consumidor do serviço e os componentes relacionados, geralmente quando um destes componentes nega o acesso;
- **Falhas de Execução:** ocorrem quando o serviço é executado, mas o resultado não corresponde ao esperado.

CHAN, BISHOP, *et al.* (2009) apresenta uma taxonomia dividindo as falhas em três grandes categorias: físicas, desenvolvimento e interação. Estas categorias são subdivididas em 13 subcategorias, conforme apresentado na Figura 5.

- **Falhas físicas:** falhas que ocorrem quando há indisponibilidade em um dos componentes que compõem o serviço. Incluem exceções de infraestrutura, rede ou no funcionamento correto do *middleware*. Exemplos de tais falhas seriam um servidor que está fora do ar ou uma conexão interrompida com o servidor;
- **Falhas de desenvolvimento**: são introduzidas em um ambiente produtivo pelos desenvolvedores, ferramentas de desenvolvimento ou através de processos de implantação.
- **Falhas de Interação**: ocorrem quando o serviço composto falha mais frequentemente ou mais severamente do que o aceitável. Essas falhas ocorrem no processo de junção entre o consumidor do serviço e os componentes relacionados

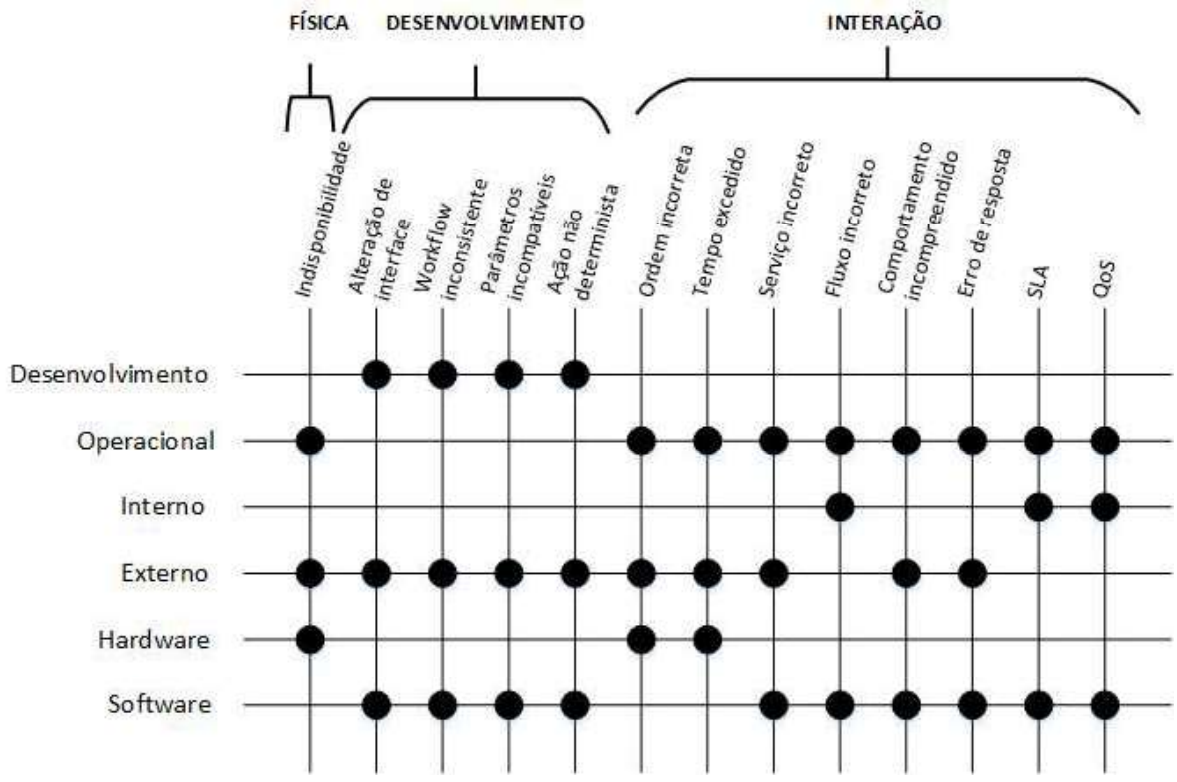

Figura 5 - Taxonomia de falhas em uma Arquitetura Orientada a Serviços. Fonte: adaptado de CHAN, BISHOP, *et al.* (2009).

A Figura 5 apresenta as três categorias principais da taxonomia proposta por CHAN, BISHOP, *et al.* (2009), com suas respectivas subcategorias e áreas onde os efeitos podem ser observados.

De acordo com BUELOW e KASI (2009), as exceções que ocorrem em um ambiente SOA podem ser classificadas em duas categorias:

- **Falhas de sistema:** são lançadas pela infraestrutura SOA em tempo de execução. Por exemplo, se um serviço invocado não for executado por causa de problemas de rede de computadores ou devido à indisponibilidade do provedor de serviços, a infraestrutura SOA lançará uma falha de conexão;
- **Falhas de negócio:** são lançadas quando ocorre uma condição de exceção durante o processamento de uma operação de um serviço, na lógica de negócios. Essa condição de exceção pode ser devida à violação de certas regras comerciais, a dados inválidos ou a outras condições não relacionadas ao sistema.

2.7 DISCUSSÃO DOS ESTUDOS CORRELATOS

Muitos trabalhos de pesquisa propuseram métodos para automatizar o processo de diagnóstico de falhas em sistemas distribuídos. Grande parte dos trabalhos encontrados na literatura especializada, se concentram na análise dos registros de *log* para identificação destas falhas, como os trabalhos de REIDEMEISTER, MUNAWAR e WARD (2010), NAGARAJ, KILLIAN e NEVILLE (2012) e VAARANDI e PIHELGAS (2015). Estes métodos se mostraram úteis e eficientes para medir a saúde do sistema monitorado como um todo, como falhas de *hardware* e rede, agrupando sintomas de erro e antecipando problemas. Se limitam, porém, no rastreamento de *logs*. Vários problemas são identificados na análise tradicional de registros de *log* como o formato de registro não consistente e registros descentralizados (CHHAJED, 2015). Este trabalho propõe uma captura dinâmica de falhas, através da integração com o ESB.

O projeto Draco de KAVULYA, DANIELS, *et al.* (2012) utiliza análise *bayesiana* para localizar problemas em sistemas distribuídos. Os excelentes resultados obtidos neste trabalho motivaram uma pesquisa sobre aplicação das técnicas de aprendizagem de máquina na classificação de falhas em SOA.

PHAM, WANG, *et al.* (2017) introduz uma abordagem para automatizar o diagnóstico de falhas, preenchendo um banco de dados com as falhas geradas por um sistema distribuído de destino. Esta técnica foi utilizada na arquitetura do presente trabalho.

O estudo sobre classificação de dados comprovou os benefícios de um método de classificação automática de falhas, como observado nos trabalhos de CABRAL (2017), XUAN, JIANG, *et al.* (2017), DOSCIATTI (2015) e NISA e QAMAR (2015). A abordagem de classificação de NISA e QAMAR (2015) pode ser usada para concentrar as consultas do usuário em um conjunto refinado de categorias de *web services,* porém não tem o foco na classificação das falhas geradas no ambiente.

Várias referências introduziram algoritmos da classificação de texto e efetuaram uma comparação entre as diversas técnicas de aprendizagem de máquinas. Algumas das técnicas mais testadas são utilizadas neste trabalho, como SVMs, *AdaBoost classifier* e *Naive Bayes classifier*. Também foi possível observar que estes trabalhos apresentaram uma média de precisão de valores verdadeiros positivos acima de 80%, auxiliando na formulação da hipótese de pesquisa, apresentada no próximo capítulo.

O estudo sobre taxonomias de falhas em SOA foi necessário para definir as categorias adotadas neste trabalho. Três estudos foram apresentados: BRUNING, WEISSLEDER e MALEK (2007), CHAN, BISHOP, *et al.* (2009) e BUELOW e KASI (2009). Os três trabalhos apresentam uma taxonomia consolidada, baseadas em problemas reais. A taxonomia de BRUNING, WEISSLEDER e MALEK (2007) é focada em falhas que podem ocorrer nas etapas de SOA, a de CHAN, BISHOP, *et al.* (2009) apresenta classes voltadas essencialmente na causa raiz de problemas durante a execução de um processo e a classificação de BUELOW e KASI (2009) é mais generalista, dividindo as falhas em dois grandes grupos principais. O presente trabalho optou por utilizar parcialmente as taxonomias apresentadas por CHAN, BISHOP, *et al.* (2009) e BUELOW e KASI (2009), por apresentarem classes de falhas que se enquadram nos objetivos do presente trabalho. As classes selecionadas serão apresentadas no próximo capítulo.

2.8 CONCLUSÃO

Este capítulo apresentou uma visão geral dos conceitos utilizados neste trabalho. Uma Arquitetura Orientada a Serviços foi definida resumidamente. O conceito central de um

ESB e sua importância em SOA foi apresentado. Também foram apresentados os principais conceitos e algoritmos de aprendizagem de máquina.

Buscando um maior embasamento teórico, foi realizada uma pesquisa de trabalhos correlatos relacionados ao tema de detecção de falhas em sistemas distribuídos e classificação de dados. A pesquisa de classificação de dados foi importante para conhecer as técnicas de aprendizagem de máquina utilizadas. Três taxonomias de falhas em SOA foram apresentadas finalizando o capítulo com a discussão destes trabalhos.

O próximo capítulo apresenta a metodologia utilizada para o desenvolvimento de todo o trabalho, descrevendo as principais atividades executadas desde o início até o fim desta dissertação.

3 MATERIAIS E MÉTODOS

Neste capítulo serão apresentas as etapas realizadas no desenvolvimento deste trabalho.

3.1 TECNOLOGIAS UTILIZADAS

As tecnologias utilizadas no desenvolvimento deste trabalho foram:

- **Linguagens de programação:** são utilizadas as linguagens *python* para desenvolvimento do módulo de classificação e Java para desenvolvimento dos demais módulos;
- ***Scikit-learn:*** biblioteca *open source* em *python* que implementa uma gama de modelos de aprendizado de máquina, validação cruzada e algoritmos de visualização utilizando uma interface unificada;
- **Oracle SOA Suite 12:** conjunto de soluções, fornecida pela Oracle Corporation, que quando reunidas facilitam a implementação de uma arquitetura orientada a serviços. Inclui várias aplicações, como BPEL, OSB, servidor de aplicação *Weblogic*, ferramentas de desenvolvimento, monitoração, entre outros. O SOA *Suite* será utilizado apenas para construir a aplicação de exemplo e demostrar a utilização do projeto;
- ***Java Agent Development Framework* (JADE):** é um *framework* de software implementado na linguagem Java. Simplifica a implementação agentes de software que suportam as fases de depuração e implantação.
- **JsonDB:** uma base de dados leve que armazena os dados em arquivos no formato Json, apresenta um baixo consumo de memória e permite que a base de dados seja embarcada em uma aplicação Java. Também fornece uma API para acesso as informações e suporta criptografia de dados.

3.2 ESTRUTURA DE FALHA NO ESB

É possível relatar uma falha gerada do OSB usando um *schema* de falha da especificação SOAP. A Figura 6 representa a estrutura de um arquivo XML representando uma falha SOAP.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <SOAP-ENV:Fault>
         <faultcode>SOAP-ENV:Server</faultcode>
         <faultstring>Server Error</faultstring>
         <detail>
            <e:myfaultdetails xmlns:e="Some-URI">
              <message>
                My application didn't work
              </message>
              <errorcode>
                1001
              </errorcode>
            </e:myfaultdetails>
         </detail>
      </SOAP-ENV:Fault>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 6 - Elemento SOAP *Fault* contendo informações do erro em uma mensagem SOAP. Fonte: W3C (2000).

Quando um serviço externo retorna uma falha SOAP, o núcleo do OSB configura uma variável interna nomeada ”*fault*”. Os elementos extraídos e usados para a construção de um elemento de falha do barramento de serviço são apresentados na Tabela 2.

O método de classificação de mensagens de falha do presente trabalho utiliza uma junção dos elementos *faultcode*, *faultstring* e *detail,* conforme representado na Figura 7, e esses campos formam o documento de falha. As amostras de falhas são escritas no idioma inglês e português e o trabalho de pré-processamento deve prever esse comportamento.

Figura 7 – Representação dos campos de um documento de falha

Elemento	Descrição
Faultcode	Código de identificação da falha. É fornecido pelo desenvolvedor do serviço ou fabricante da plataforma. Pode ser um código padronizado, um valor numérico indicado sucesso ou falha (como 0 ou 1) ou simplesmente um identificador da origem da falha (*Soap-env:Client* ou *Soap-env:Server*).
Faultstring	Uma explicação da falha.
Detail	Informações específicas do serviço que apresentou a falha. Este elemento pode conter elementos filhos com diversas informações úteis como o nome do serviço que originou falha.

Tabela 2 - Elementos de um documento de falha SOAP

3.3 COLETA DE DADOS

Para a execução das fases de treino, teste e validação dos algoritmos de classificação de texto foram utilizadas amostras de mensagens de falha retiradas dos registros de *log* de um ambiente de desenvolvimento do OSB. Os registros coletados são os *logs* de erro padrão do barramento. Para complementar e enriquecer os registros, mensagens de falha de serviços de fabricantes de *software* foram injetadas aos registros de *log* com o objetivo de manter o treino dos algoritmos o mais próximo possível de um ambiente de produção. Listas com amostras das mensagens e referências dos fabricantes de *software* selecionados se encontram no Apêndice C do presente trabalho, esta referência é útil para simulação dos resultados obtidos. Os dados dos registros de *log* foram extraídos automaticamente por um *script* desenvolvido na linguagem *python* especialmente para essa tarefa denominado *FaultMessagesExtractor.py.* Este *script* recebe como parâmetro de entrada um período de extração e contém métodos para a execução das seguintes tarefas:

- Extração dos registros de *log* de arquivos compactados;
- Identificação de mensagens de falha SOAP nos registros de *log*;

- Remoção de campos irrelevantes para a classificação como: campos de identificação de usuários e equipamentos, datas, número de ordens internas, etc;
- Eliminação de repetições (uma amostra de cada tipo de mensagem);
- Junção do arquivo com amostras de falhas exclusivas e pré-classificadas com todas as amostras de falha encontradas nos registros de *log* do período informado, gerando um segundo arquivo pré-classificado com todas as falhas, incluindo falhas repetidas.

As mensagens extraídas são pré-classificadas nas classes de falha estabelecidas neste trabalho e salvas no formato *Comma Separated Value* (CSV). O padrão CSV foi escolhido por ser simples de manter e fácil de manipular em uma linguagem de programação. Este arquivo, chamado de **conjunto de dados,** é utilizado na fase de treinamento, teste e validação dos algoritmos de aprendizagem de máquina.

Conforme apresentado no Capítulo 2 este trabalho adotou parcialmente as taxonomias de CHAN, BISHOP, *et al.* (2009) e BUELOW e KASI (2009). Selecionou os três grandes grupos de falhas do trabalho de CHAN, BISHOP, *et al.* (2009): **falhas físicas**, **falhas de desenvolvimento** e **falhas de interação.** As mensagens de falha das classes de **desenvolvimento** e **interação** possuem similaridades e geralmente são solucionadas por uma mesma equipe, com isso no presente trabalho essas classes foram unificadas em uma mesma categoria denominada: **falhas de desenvolvimento/interação.** A classe de **falhas de negócio** do trabalho de BUELOW e KASI (2009), também foi utilizada pelo presente trabalho. A taxonomia deste trabalho é apresentada na Figura 8.

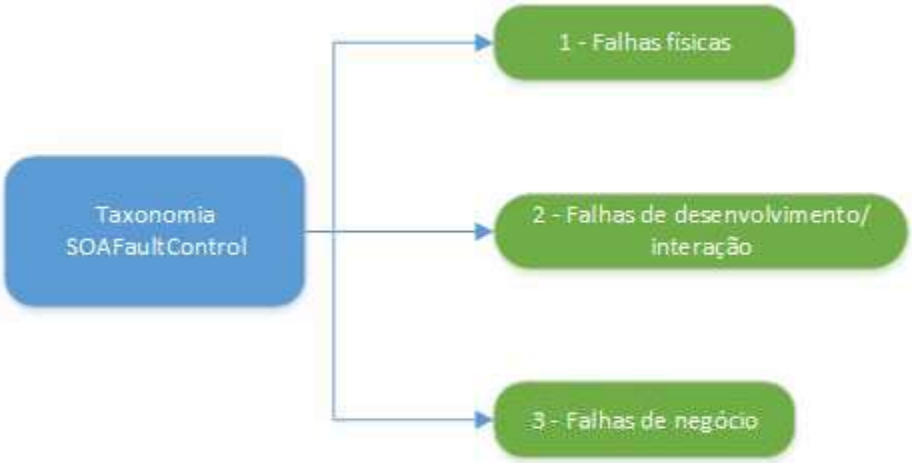

Figura 8 - Taxonomia do projeto SoaFaultControl.

3.4 VALIDAÇÃO DO MODELO PREDITIVO

A técnica de validação cruzada (*Cross Validation* ou *K-fold*) foi utilizada para validar como o modelo preditivo irá funcionar na prática.

Conforme descrito na contextualização, neste processo o conjunto de dados é dividido em *k* subconjuntos. Em cada etapa, um dos subconjuntos *k* é usado como teste e os outros *k-1* subconjuntos formam o conjunto de treinamento. As estatísticas de desempenho são calculadas em todos os ensaios *k*, fornecendo assim uma boa indicação de desempenho do classificador em dados não vistos. O valor de *k* utilizado foi de 10.

3.5 MÉTRICAS DE AVALIAÇÃO

As métricas de avaliação dos algoritmos são uma parte importante em um projeto de aprendizado de máquina. A escolha destas métricas influencia no modo como o desempenho de um algoritmo de aprendizagem de máquina será medido e comparado, pesando as diferentes características e influenciando a escolha final do algoritmo.

Algumas medidas de qualidade podem ser estimadas a partir dos seguintes resultados observados na Tabela 3.

Medida	Descrição
VP (verdadeiro positivo)	Número de exemplos positivos que foram corretamente classificados.
FP (falsos positivos)	Número de exemplos negativos classificados como positivos.
FN (falsos negativos)	Número de exemplos positivos classificados como negativos.
VN (verdadeiros negativos)	Número de exemplos negativos corretamente classificados.

Tabela 3 - Medidas de qualidade utilizadas na avaliação de modelos de classificação

Neste trabalho, os métodos de aprendizagem de máquina foram avaliados utilizando as seguintes métricas:

- **Precisão *(Precision):*** porcentagem de padrões classificados como pertencentes à classe positiva e que realmente pertencem à classe positiva (SILVA, ALMEIDA e YAMAKAMI, 2012). A precisão ou *precision* pode ser obtida pela fórmula representada pela Equação 2 (CAVALCANTI, 2016):

$$Precisão = \left(\frac{VP}{VP + FP}\right) \tag{2}$$

- **Acurácia *(Accuracy)***: taxa de acerto global, ou seja, proporção de predições corretas em relação ao tamanho do conjunto de dados (SILVA, ALMEIDA e YAMAKAMI, 2012). Seu cálculo é descrito pela Equação 3 (CAVALCANTI, 2016):

$$Acurácia = \left(\frac{VP + VN}{VP + FP + VN + FN}\right) \tag{3}$$

- **Revocação *(Recall)***: proporção de padrões da classe positiva identificada corretamente. Indica o quão bom o classificador é para identificar a classe positiva (SILVA, ALMEIDA e YAMAKAMI, 2012). O cálculo desta metida é descrito pela Equação 4 (CAVALCANTI, 2016):

$$Revocação = \left(\frac{VP}{VP + FN}\right) \quad (4)$$

- **F1** ***(F1 score)***: como forma de medir o equilíbrio entre precisão e revocação, a média harmônica entre estas duas medidas é calculada. É obtida a partir da Equação 5 (CAVALCANTI, 2016):

$$F1 = 2 . \left(\frac{Precisão\ .\ Revocação}{Precisão + Revocação}\right) \quad (5)$$

- **Tempo computacional:** termo utilizado para descrever duas métricas adicionais: tempo de construção e velocidade de classificação. O tempo de construção refere-se ao tempo em segundos necessário para treinar um classificador dado um determinado conjunto de dados. A velocidade de classificação descreve o tempo de classificação em segundos (WILLIAMS, ZANDER e ARMITAGE, 2006).

Os dados das métricas são apresentados em uma matriz de confusão, uma tabela que permite a visualização do desempenho de um algoritmo de aprendizado. Cada coluna da matriz representa as instâncias de uma classe prevista, enquanto as linhas representam os casos de uma classe real. O nome é originado do fato de que a matriz torna mais fácil verificar se o algoritmo está confundindo duas classes (TECNOLOGIA, 2015), um exemplo é apresentado na Figura 9.

		PREDITO	
		Classe A	**Classe B**
VERDADEIRO	**Classe A**	VP (VERDADEIRO POSITIVO)	FN (FALSO NEGATIVO)
	Classe B	FP (FALSO POSITIVO)	VN (VERADEIRO NEGATIVO)

Figura 9 - Exemplo de uma matriz de confusão onde as linhas representam os valores verdadeiros.e as colunas os valores previstos por testes. Fonte: TECNOLOGIA (2015).

3.6 CONSTRUÇÃO DA HIPÓTESE DE PESQUISA

A metodologia utilizada nos experimentos deste trabalho pode ser classificada como uma pesquisa experimental. Neste tipo de metodologia, segundo Koche (1997), o investigador analisa o problema, constrói suas hipóteses e trabalha manipulando os possíveis fatores e variáveis, que se referem ao fenômeno observado, para avaliar como se dão suas relações preditas pelas hipóteses.

Para que a ideia inicial de um sistema de classificação dinâmica evoluísse para uma proposta concreta de projeto, o primeiro passo foi realizar um estudo e experimentos de técnicas de aprendizado de máquina para classificação dos documentos de falhas.

O objetivo deste estudo é responder à pergunta de pesquisa: ***"Sobre uma classificação de mensagens de falha de uma Arquitetura Orientada a Serviços, dado um conjunto de dados pré-classificados, qual é a técnica de classificação que apresenta um melhor desempenho em termos de acurácia e desempenho computacional?"***

Diante da pergunta de pesquisa formulada, buscou-se verificar, então, a seguinte hipótese: ***"Fornecendo um conjunto de dados selecionados e pré-classificados manualmente para treino e efetuando um trabalho de pré-processamento de texto, é possível selecionar um algoritmo de classificação que supere 80% de valores VP."***

Para verificação da hipótese os seguintes algoritmos preditivos foram testados:

- ***Support Vector Machine:*** utilizando as abordagens *One-vs-the-rest* e *One-vs-one;*
- **Classificação baseada em conjuntos *(AdaBoost Classifier)*;**
- ***Naive Bayes Classifier.***

Estas técnicas de aprendizagem de máquina foram selecionadas seguindo critérios: ampla utilização em trabalhos de pesquisas observados no levantamento bibliográfico, resultados satisfatórios e adaptação para uma classificação multiclasses. No caso do modelo *AdaBoost* foi levado em consideração o fato desta técnica utilizar outros modelos de classificação, denominados fracos e combiná-los em um único classificador forte a fim de obter uma maior precisão.

Conforme apresentado na discussão dos estudos correlacionados, as pesquisas de classificação de texto em sistemas distribuídos apresentaram uma assertividade acima de 80% de valores VP, que ajudou a formular a hipótese do presente trabalho.

3.7 CONCLUSÃO

Este capítulo apresentou a metodologia utilizada no desenvolvimento deste trabalho, incluindo: tecnologias utilizadas, o processo de coleta e pré-processamento dos dados.

A técnica de validação cruzada foi adotada para avaliar como o modelo preditivo irá funcionar na prática.

O capítulo também apresentou as métricas adotadas para avaliação dos métodos de aprendizagem de máquina. A pergunta de pesquisa e a hipótese, que guiam este trabalho, também foram apresentadas.

As informações deste capítulo são fundamentais para entendimento da arquitetura do sistema e dos resultados obtidos, definidos nos próximos capítulos.

4 MÉTODO DE CLASSIFICAÇÃO AUTOMÁRICA SOAFAULTCONTROL

Através do estudo apresentado neste trabalho chegou-se a uma proposta de arquitetura descrita a seguir. A Figura 10 apresenta um diagrama de componentes com os elementos que compõe esta arquitetura.

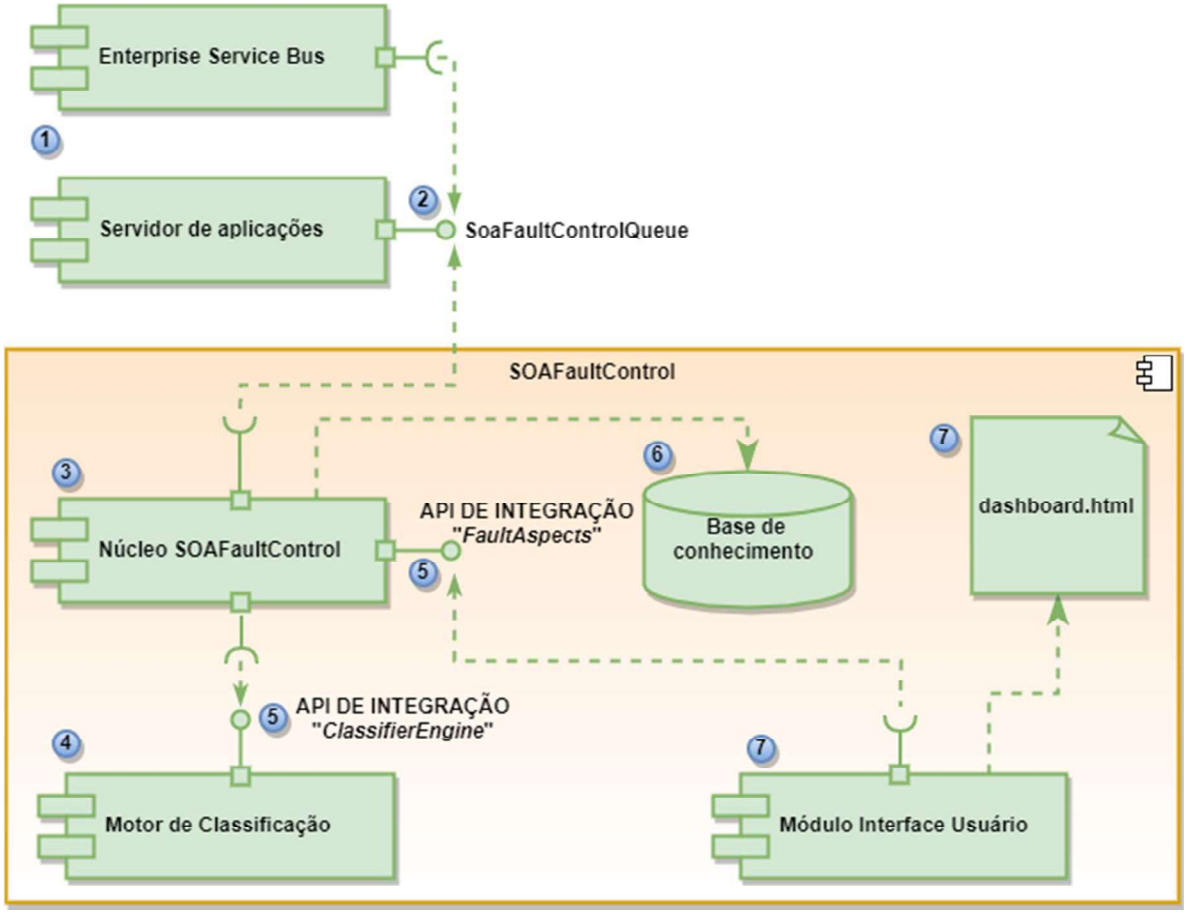

Figura 10 – Diagrama de componentes da arquitetura SOAFaultControl

A arquitetura *SoaFaultControl* é formada pelos seguintes componentes:

1) *Enterprise Service Bus* e Servidor de Aplicações;
2) Filas JMS;
3) Núcleo do sistema SOAFaultControl;
4) Motor de classificação;
5) APIs de integração;
6) Base de conhecimento;
7) Módulo de interface com usuário;

As próximas subseções detalham cada um destes componentes.

4.1 *ENTERPRISE SERVICE BUS* E SERVIDOR DE APLICAÇÕES

A arquitetura proposta é centrada em um ESB. Espera-se que o ambiente, onde a esta arquitetura será inserida, utilize um ESB para abstrair as implementações dos sistemas heterogêneos, conferindo um certo grau de desacoplamento em relação à implementação. Conforme apresentando anteriormente, uma das funções de um ESB é justamente centralizar os artefatos utilizados na comunicação dos sistemas distribuídos.

As principais implementações de ESB do mercado permitem a monitoração e coleta de informações em tempo de execução, como os eventos de falhas. Essa coleta é possível através de componentes de alerta. Os alertas são usados dentro do fluxo de mensagens de um serviço de *proxy.* A utilização destes alertas especificamente no fluxo de tratamento de exceções do serviço de *proxy* permite capturar os detalhes da falha e enviar a um destinatário. Esse destinatário pode ser um registro de *log, Simple Network Management Protocol* (SNMP), alertas de e-mail e filas JMS.

A Figura 11 apresenta um serviço de *proxy* do OSB com um componente de alerta configurado no tratamento de exceções do *proxy*.

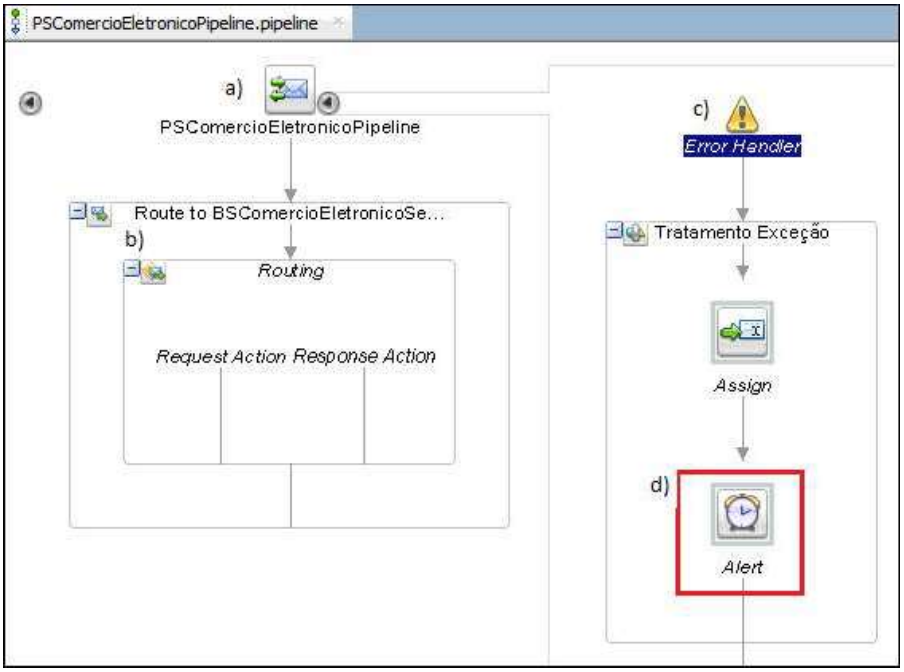

Figura 11 –Alerta no tratamento de exceção de um *ProxyService* do OSB versão 12 C.

Na Figura 11 é possível observar:

a) Início do fluxo de *Proxy Service*;
b) Componente *routing* do OSB, roteando uma requisição recebida pelo *Proxy Service* para o *Business Service*;
c) Fluxo de tratamento de exceção;
d) Componente de alerta que permite o direcionamento um evento de falha para um destinatário, como uma fila JMS.

Um ESB, na verdade, é uma aplicação que é executada sobre um servidor de aplicações. Portanto, um servidor de aplicações é parte da arquitetura deste método pois além de ser uma plataforma para o ESB é essencial para hospedar as filas JMS.

4.2 FILAS JMS

JMS é uma API da linguagem Java para *middleware* orientado a mensagens que permite a integração de duas aplicações através de mensagens postadas em uma fila.

Foram criadas duas filas: uma fila que recebe os eventos de falha provenientes do ESB e uma segunda fila para onde são direcionados os eventos que não puderam ser consumidos pelo núcleo do SOAFaultControl. É possível que uma mensagem contenha dados inválidos ou fora do padrão, é uma boa prática introduzir uma fila de mensagens inválidas para onde são direcionados os eventos que não puderam ser consumidos pelo destinatário. A Tabela 4 apresenta a configuração destas filas.

Configuração/Tipo	**Nome**
Factory de Conexão	cf.SoaFaultControl
Fila que recebe os eventos de falha do ESB	jms.SoaFaultControlQueue
Fila de mensagens inválidas	jms.SoaFaultControlErrorQueue

Tabela 4 – Configuração das filas do método SOAFaultControl

Os registros capturados no ESB devem ser modelados no formato *JavaScript Object Notation* (JSON) apresentado na Figura 12.

Figura 12 – Notação JSON esperada na fila JMS do método SOAFaultControl

A mensagem JMS representada na Figura 11 é composta pelos seguintes elemtos atributo-valor:

a) *faultcode:* código de identificação da falha. É fornecido pelo desenvolvedor do serviço ou fabricante da plataforma;
b) *faulmessage:* mensagem com a descrição da falha;
c) *detail: i*nformações específicas do serviço que apresentou a falha;
d) *serviceName:* nome do serviço que originou a falha;
e) *operation:* nome da operação invocada através do ESB que originou a falha;

4.3 NÚCLEO DO SISTEMA SOAFAULTCONTROL

O núcleo do sistema SOAFaultControl é o componente principal da arquitetura, responsável em coordenar o trabalho de todos os módulos do sistema.

A coordenação do trabalho é realizada por um agente de software, com sensor destinado a monitorar frequentemente a fila JMS de falhas. Caso perceba a inclusão de um novo evento, o agente aciona os atuadores responsáveis em: validar a existência da falha na base de conhecimento, classificar a mensagem de falha e salvar os eventos na base de conhecimento. A Figura 13 define as atividades do agente de controle da arquitetura SOAFaultControl.

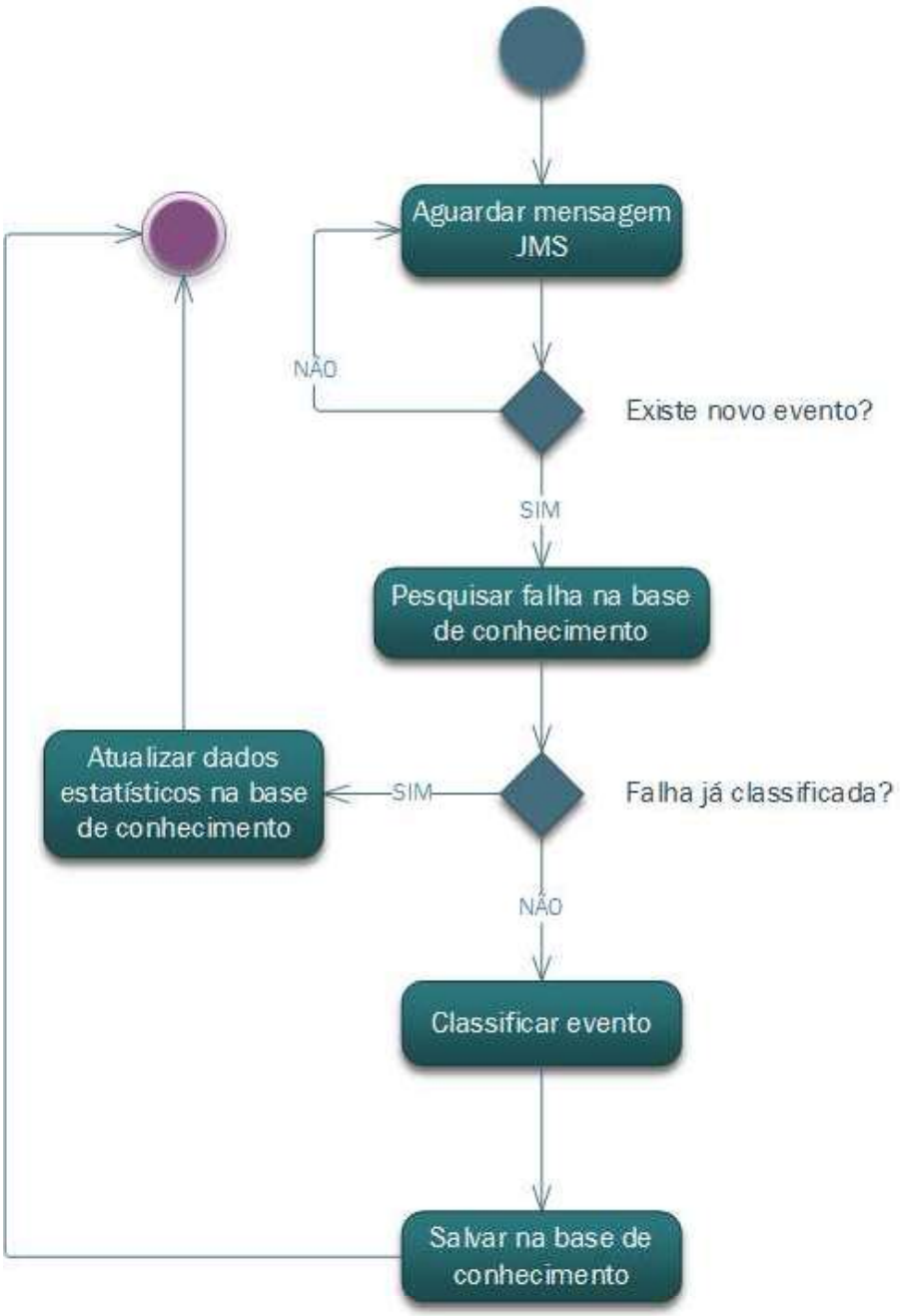

Figura 13 – Diagrama de atividades do núcleo do sistema SOAFaultControl.

4.4 MOTOR DE CLASSIFICAÇÃO

O motor de classificação utiliza técnicas de Aprendizado de Máquina supervisionado para formar um mecanismo de classificação das falhas recebidas do ESB. É nesta fase que as técnicas de classificação de texto são aplicadas. A classificação visa identificar as principais palavras-chave para predizer exemplos de falhas não vistos

anteriormente. O motor de classificação foi escrito na linguagem de programação *Python* e é composto de três fases distintas:

1. **Pré-processamento das amostras de falha:** executa uma limpeza no texto e prepara para a classificação;
2. **Treino do algoritmo de classificação:** o algoritmo de classificação selecionado para a arquitetura é treinado. A fase de treino é realizada somente na inicialização do sistema;
3. **Classificação da mensagem de falha:** o texto com a mensagem de falha é classificado utilizando um dos algoritmos apresentados nesse trabalho, será utilizado o algoritmo com melhor desempenho nas experiências realizadas no presente trabalho.

4.4.1 PRÉ-PROCESSAMENTO

Pré-processamento é a fase realizada para obter as características de classificação de texto. Tem o objetivo de melhorar a qualidade dos dados já disponíveis e organizá-los. Como parte do processo de pré-processamentos as seguintes etapas foram realizadas:

1. ***Case Folding:*** mantém um padrão para a comparação, mantendo todos os textos em caixa baixa. Para implementar o *case folding* foi utilizada a função *lower()* nativa da linguagem de programação *Python* para tratamento de *strings*;
2. ***Tokenização*:** processo que separa os elementos constituintes do texto, identificando as palavras que deverão compor a representação estatística do exemplo. O processo de *tokenização* inclui a remoção da pontuação e caracteres especiais, espaços em branco entre os termos, quebra de linhas e tabulações. Para efetuar um processo de *tokenização* esse trabalho utilizou a biblioteca da linguagem *Python National Language Tookit* (NLTK). A NLTK possui um pacote chamado *punkt* que é capaz de separar palavras de um texto, tanto pelos espaços em branco quanto pela pontuação;
3. **Remoção de palavras repetidas:** é necessário manter um conjunto de palavras distintas, portanto, a repetição de palavras deve ser tratada. Para efetuar esse tratamento as palavras são adicionadas em um conjunto nativo da linguagem *Python:*

Set(). Um *Set* é implementado em *Python* como uma coleção não ordenada de itens. Cada elemento é único e imutável, não permitindo duplicatas;

4. ***Validação do tamanho da palavra:*** foram mantidas somente palavras maior que três caracteres;
5. **Remoção das *stop words*:** remove as palavras de frequência alta que não constituem conhecimento no texto. As *stop words* tratam palavras que não apresentam um conteúdo semântico significante ao texto como palavras auxiliares ou conectivas (como exemplo: para, com, eles, elas). Para implementar essa função é necessário um conjunto de *stop words*. A NLTK possui o pacote *Corpus* que contém um conjunto de *stop words* para o idioma inglês e português.
6. ***Stemming:*** método de remoção do sufixo e prefixo dos termos que possam representar a raiz das palavras, removendo assim a variação verbal ou plural. Como exemplo as palavras "disponibilidade" e "disponível" são reduzidas para "disponi". As palavras "recuperar" e "recuperados" são reduzidas para "recuper". Para implementação da função *stemming* no motor de classificação foi utilizada a NLTK: o pacote o *RSLPStemmer* que é um extrator de raiz das palavras para a língua Portuguesa e o pacote *PorterStemmer* um extrator de raiz das palavras para a língua inglesa.

O resultado do pré-processamento é a geração de um conjunto de palavras distintas contidas nos textos, que são agrupadas em uma estrutura denominada *Bag-of-words* (BoW) ou dicionário de palavras. O modelo BoW aprende um vocabulário com os termos de todos os documentos analisados e converte em um modelo representado por uma matriz, onde cada linha representa um documento e cada coluna representa um termo. O valor do campo é o valor da frequência do termo no documento (S, RAJ e S.RAJARAAJESWARI, 2016).

A Figura 14 apresenta um diagrama de atividades representando o pré-processamento.

Figura 14 – Diagrama representando as atividades de pré-processamento.

4.4.2 TREINO DO ALGORITMO E CLASSIFICAÇÃO DAS MENSAGENS DE FALHA

Para execução das etapas de treino do algoritmo e classificação das mensagens de falha, é necessário a construção um modelo básico na linguagem de programação *Python* para cada algoritmo de classificação selecionado. Um modelo deve ser capaz de separar o conjunto de dados fornecido nas três categorias apresentadas neste trabalho. A construção do modelo é realizada através da biblioteca *scikit learn.*

A Tabela 5 apresenta os modelos de classificação da biblioteca *scikit learn* selecionados para os algoritmos apresentados neste trabalho.

Algoritmo	Modelo	Descrição
SVM OneVsRest e *OneVsOne*	*LinearSVC*	Modelo de classificação binário e multiclasses. Permite a classificação de um número superior a 10.000 amostras. A decisão de classificação é feita com base no valor de uma combinação linear das características.
Naive Bayes	*Multinomial*	Implementa um algoritmo *Naive Bayes* para dados distribuídos multinomialmente. É uma versão especializada de *Naive Bayes* projetada para classificação de documentos de texto.
AdaBoost	*DecisionTreeClassifier*	Utiliza o modelo padrão do *AdaBoost* definido na biblioteca *scikit learn,* um classificador de árvore de decisão

Tabela 5 - Modelos selecionados para treino dos algoritmos e classificação das mensagens de falha.

Os modelos de classificação *scikit learn* são treinados utilizado a função *fit(x,y)* e a classificação é feita através da função *predict*(x) , onde:

- *"x":* representa um conjunto com a entrada dados de um determinado domínio, ou seja, conjunto de amostras com as características do texto;
- *"y"* representa as marcações ou categorias do texto.

A Figura 15 apresenta um pseudocódigo exemplificando uma função responsável em criar um modelo *Multinomial Naive Bayes,* treinar o modelo e classificar uma amostra de texto.

```
1  FUNÇÃO TreinarEprever(arquivoCVS,colunaTexto,colunaMarcas,textoAprever)
2
3      x <- lerArquivoCSV(arquivoCVS,colunaTexto)
4      y <- lerArquivoCSV(arquivoCVS,colunaMarcas)
5      modelo <- MultinomialNB()
6      modelo.treinar(x, y)
7      previsto <- modelo.prever(textoAprever)
8
9      Retornar previsto
10
```

Figura 15 - Pseudocódigo de uma função responsável em treinar e classificar um texto.

O pseudocódigo da Figura 15 apresenta uma função denominada “TreinarEprever” que recebe como parâmetros:

- Nome do arquivo CVS;
- Coluna do arquivo CVS que contém as amostras de texto utilizado para o treinamento;
- Coluna do arquivo CVS que representa a classificação do texto;
- Texto que será classificado;

No pseudocódigo as linhas 3 e 4 criam as variáveis *x* e *y*. O modelo é criado na linha 5, treinado na linha 6 e realiza a previsão na linha 7.

4.5 APIS DE INTEGRAÇÃO

As APIs de integração permitem acesso os recursos e funcionalidades do sistema. Duas APIs são disponibilizadas:

1. ***ClassifierEngine*:** disponibiliza operações para integrar o motor de classificação, escrito na linguagem de programação *Python*, com os demais módulos do sistema;
2. ***FaultAspects*:** expõe funcionalidades do sistema para integração com a interface gráfica e clientes externos. Permite a consulta de uma falha específica ou um conjunto de falhas classificadas.

Os recursos disponibilizados pelas APIs são baseados em *Web services Restful* e as mensagens são padronizadas no formato JSON. A Figura 16 apresenta um exemplo do documento de entrada e saída da API *ClassifierEngine* que se integra com o processo de classificação de falha.

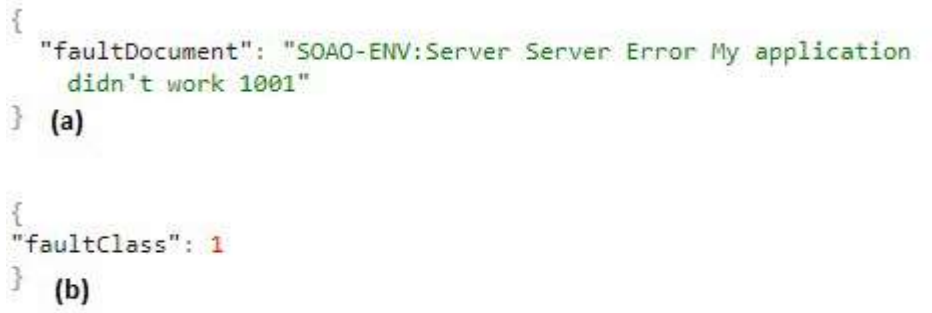

Figura 16 – Documento no padrão JSON da API *ClassifierEngine*: (a) documento de entrada e (b) documento de saída.

4.6 BASE DE CONHECIMENTO

A base de conhecimento é uma base de dados modelada a documentos que armazena as mensagens classificadas. A base de conhecimento é utilizada como fonte de consultas dos usuários do sistema através do módulo de interface com o usuário. A base contém registros das falhas capturadas pelo sistema, sua classificação e número de eventos recebidos.

A base de conhecimento é armazenada em arquivos de texto no formado *Json* através da API da base de dados JsonDB (ver Seção 3.1).

Para persistência e pesquisa dos dados é necessário gerar uma chave exclusiva para cada registro da base. Para gerar a chave é necessário: selecionar o documento de falha do registro (ver Seção 3.2), limpar os *outliers* (ver Seção 2.5) e gerar um *hash Message-Digest algorithm 5* (MD5) com os dados selecionados. O MD5 é um algoritmo de *hash* de 128 bits unidirecional.

O acesso a base de dados é realizado através de um *Data Access Object* (DAO), que é um padrão para persistência de dados que permite separar regras de negócio das regras de acesso ao banco de dados (Ver apêndice A).

4.7 INTERFACE COM USUÁRIO

O principal objetivo do módulo de interface com o usuário é permitir a interação entre os especialistas humanos e o sistema. O módulo é composto por um servidor HTTP embarcado e uma página HTML que permite aos usuários pesquisar as falhas já classificadas.

4.8 CONCLUSÃO

Este capítulo apresentou a arquitetura do sistema SOAFaultControl, definindo minuciosamente a função de cada módulo e a maneira como se relacionam. O diagrama de classes da arquitetura e a documentação dos artefatos de software pode ser consultada nos apêndices deste trabalho.

O próximo capítulo apresenta os experimentos realizados nesse trabalho, para a seleção do algoritmo de classificação de texto utilizado no motor de classificação. Também apresenta os experimentos relacionados à implementação de toda a arquitetura.

5 RESULTADOS

Neste capítulo são apresentados os resultados dos testes experimentais realizados neste trabalho. Estes experimentos são divididos em duas seções principais: a primeira seção descreve os experimentos relacionados à avaliação de técnicas de aprendizagem de máquina; A segunda seção está relacionada com experimentos de avaliação do funcionamento de toda arquitetura.

5.1 AVALIAÇÃO DE TÉCNICAS DE APRENDIZAGEM DE MÁQUINA

O objetivo dos experimentos apresentados nessa seção é responder à pergunta de pesquisa e testar a hipótese deste trabalho, ambos formulados na Seção 3.8, selecionando assim um algoritmo de classificação para a arquitetura SOAFaultControl.

5.1.1 PREPARAÇÃO DO CONJUNTO DE DADOS

Os testes contemplaram dois conjuntos de dados: um conjunto com todas as falhas retiradas dos registros de *log* e um conjunto de dados apenas com amostras exclusivas. O *script Python FaultMessagesExtractor.py,* citado no Capítulo 3, foi executado inicialmente para extrair um exemplo de cada mensagem de falha e retornou 162 amostras de falhas exclusivas. Estas mensagens de falha foram encontradas diversas vezes nos registros de *log,* porém o objetivo desta fase é separar um exemplo de cada mensagem de falha dos registros de *log*. Para separar as mensagens de falha exclusivas o *script* ignorou, através do uso de expressões regulares, informações como: data, hora, número de atividades, nome de máquinas, nome de domínios, etc.

Estas amostras foram pré-classificadas manualmente nas três categorias definidas para estre trabalho.

A Tabela 6 apresenta algumas amostras de mensagens de cada categoria.

Categoria	Mensagem
Falhas Físicas	Test "{1}" set up for pool "{0}" failed with exception: "{2}"
	Received an exception when closing a cached statement for the pool "{0}": {1}.
	Connection pool "{0}" deployment failed with the following error: {1}.
	The socket close failed. Reason: {0}.
Falhas de Desenvolvimento/Interação	Failed to assign value to context variable "{0}". Value must be an instance of
	Failed to marshall the value of context variable "{0}" to XML
	Message must be a soap:Envelope
	Failed to assign value to context variable "{0}". Variable is read-only
Falhas de Negócio	Unable to calculate work skill ID for given fields
	Can't start activity: start_appointment: The appointment cannot be started at the specified time
	Unable to determine time zone difference
	Can't link activities: add_appt_link: Duplicate appointment links are not allowed

Tabela 6 – Amostras de mensagens extraídas dos registros de *log*.

Tabela 7 apresenta o resultado desta pré-classificação.

Categoria	Quantidade de Amostras
Falhas Física	28
Falhas de Desenvolvimento/Interação	65
Falhas de Negócio	69
Total	**162**

Tabela 7 – Distribuição das amostras de falhas distintas após a pré-classificação.

A Figura 17 apresenta um gráfico do tipo “nuvens de palavras” apresentando as palavras com maior incidência nas amostras de falhas, antes de qualquer tratamento.

Figura 17 – Palavras com maior incidência nas amostras de falhas.

Após a pré-classificação, o *script FaultMessagesExtractor.py* foi executado com o objetivo de extrair todas as falhas dos registros de *log*, incluindo as mensagens de falhas repetidas, e compará-las com o registro pré-classificado. O objetivo é extrair todas as mensagens de falha, buscar a classificação no arquivo inicial e identificar a categoria da falha. O resultado é um conjunto de dados pré-classificado com todas as falhas extraídas dos registros. A distribuição do conjunto de dados em categorias é apresentada na Tabela 8.

Categoria	Quantidade de Amostras
Falhas Físicas	1.111
Falhas de Desenvolvimento/Interação	2.312
Falhas de Negócio	3.203
Total	**6.626**

Tabela 8 – Distribuição em categoria das amostras de falhas extraídas dos registros de *log*.

Através do processo de extração de dados, foi possível validar a afirmação de Reidemeister, Munawar e Ward (2010), citada na introdução deste trabalho, de que vários estudos indicam que uma grande parte das falhas em sistemas distribuídos são causadas por defeitos recorrentes. A extração trouxe 6.626 amostras que representam apenas 162 tipos de falhas diferentes.

5.1.2 PRIMEIRA AVALIAÇÃO: MÉTRICAS ACURÁCIA E TEMPO COMPUTACIONAL

Após a preparação do conjunto de dados com amostras de falhas extraídas dos registros de *log*, os algoritmos de classificação de texto foram executados e testados: *SVM OneVsRest* (Seção 2.5.1)*, SVM OneVsOne* (Seção 2.5.1)*, Naive Bayes* (Seção 2.5.2) e *AdaBoost* (Seção 2.5.3). O primeiro experimento mediu o desempenho dos algoritmos de classificação utilizando as métricas de avaliação **acurácia** e **tempo computacional** (ver Seção 3.5**)**. Utilizou-se o método *Cross Validation,* descrito na Seção 2.5.4, com um valor de *k = 10*, ou seja, o método dividiu o conjunto de dados em 10 partes e realizou 10 rodadas de teste. A cada interação uma parte foi utilizada para testar e as outras 9 para treinar o modelo. Ao final das *k* iterações calculou-se a métrica computacional sobre as falhas encontradas, obtendo assim uma medida mais confiável. A Tabela 9 apresenta os resultados do primeiro experimento.

Algoritmo	**Acurácia**	**Tempo Computacional (Segundos)**
SVM OneVsRest	0,9992	4,861
SVM OneVsOne	0,9993	3,610
Naive Bayes	0,9817	0,3138
AdaBoost	0,9532	28,004

Tabela 9 – Resultados das métricas acurácia e tempo computacional para um conjunto de dados extraído dos registros de *log*.

O objetivo principal deste experimento é medir como o sistema se comportará quando for solicitado a classificação de mensagens de falhas já conhecidas. Os resultados da Tabela 9 foram superiores aos trabalhos relacionados, apresentados na Seção 2.6. O alto

índice de acerto está relacionado diretamente à base de treinamento e testes. Como o conjunto de dados apresentou mensagens repetidas, muitas das mensagens da base de teste estão presentes também na base de treinamento.

A Figura 18 apresenta um gráfico comparando os valores das métricas obtidas neste experimento.

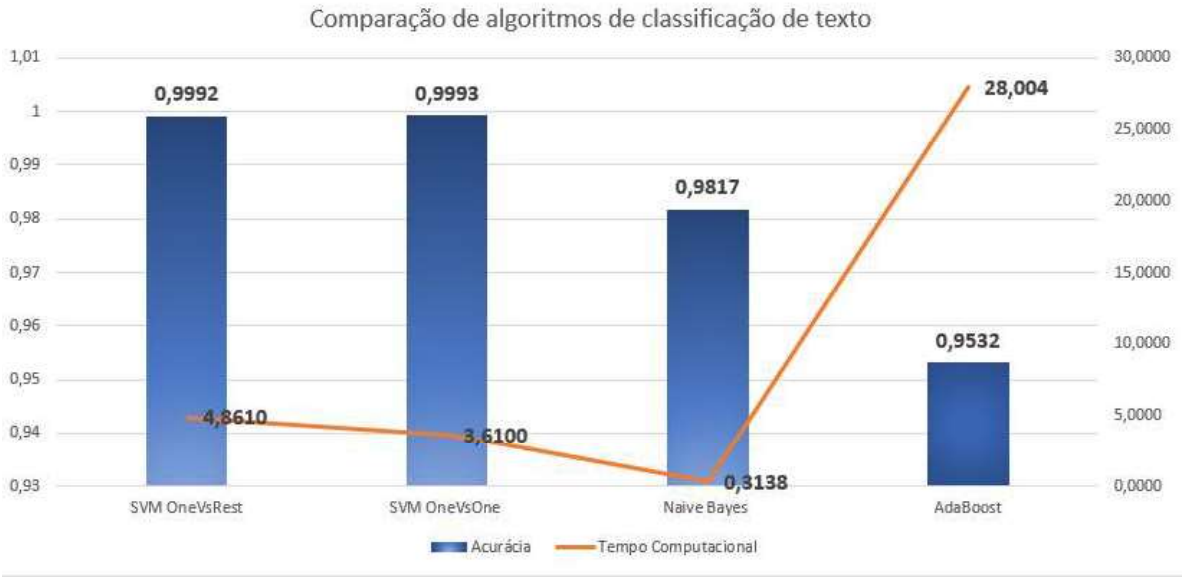

Figura 18 – Gráfico de comparação dos algoritmos de classificação de texto utilizando as métricas acurácia e tempo computacional.

É possível observar que os algoritmos *SVM* obtiveram o melhor desempenho na métrica acurácia, o algoritmo *Naive Bayes* obteve a melhor desempenho no tempo computacional e o algoritmo *AdaBoost* obteve os resultados piores em ambas métricas de avaliação.

5.1.3 SEGUNDA AVALIAÇÃO: MÉTRICAS PRECISÃO, REVOCAÇÃO E F1

A segunda avaliação comparou o desempenho dos algoritmos de classificação de texto através das métricas: precisão, revocação, F1 (Ver Seção 3.5). O objetivo desta avaliação é medir o desempenho dos algoritmos em cada categoria de falha. Foi utilizado o mesmo conjunto de dados da primeira avaliação. Nesse experimento foram separados 33% dos dados para teste e o restante para treino do algoritmo.

A Tabela 10 apresenta os resultados para o algoritmo *SVM* nas abordagens *OneVsRest e OneVsOne,* que obtiverem o mesmo resultado.

Categoria	**Métrica**		
	Precisão	**Revocação**	***F1***
Falhas Físicas	1,00	1,00	1,00
Falhas de Desenvolvimento/ Interação	1,00	1,00	1,00
Falhas de Negócio	1,00	1.00	1,00
Média	**1,00**	**1,00**	**1,00**

Tabela 10 – Relatório de classificação *SVM OneVsRest* e *OneVsOne* para um conjunto de dados extraídos dos registros de *log*. As duas técnicas obtiveram o mesmo resultado;

A matriz de confusão apresentada na Tabela 11 resume os resultados do experimento para algoritmos *SVM OneVsRest* e *OneVsOne.* Todas as suposições corretas estão localizadas na diagonal da tabela, por isso é fácil inspecionar visualmente a tabela por erros, pois são representados por valores fora da diagonal. Nesta matriz de confusão as 378 amostras de falhas físicas reais foram previstas corretamente, assim como as 1049 amostras de falha de interação reais, que também foram todas previstas corretamente. Das 760 amostras de falhas de desenvolvimento reais o algoritmo previu que 2 eram falhas físicas e uma era falha de interação.

		PREDITO		
		Falhas Físicas	**Falhas de Desenvolvimento/ Interação**	**Falhas de Negócio**
VERDADEIRO	**Falhas Físicas**	378	0	0
	Falhas de Desenvolvimento/ Interação	2	757	1
	Falhas de Negócio	0	0	1049

Tabela 11 – Matriz de confusão para os algoritmos *SVM OneVsRest* e *OneVsOne.*

A Tabela 12 apresenta o resultado da classificação para o algoritmo *Naive Bayes:*

Categoria	**Métrica**		
	Precisão	**Revocação**	***F1***
Falhas Físicas	0,96	0,99	0,98
Falhas de Desenvolvimento/ Interação	1,00	0,95	0,98
Falhas de Negócio	0,98	1,00	0,99
Média	**0,98**	**0,98**	**0,98**

Tabela 12 - Relatório de classificação *Naive Bayes* para um conjunto de dados extraídos dos registros de *log*.

É possível observar estes mesmos dados em uma representação gráfica de Mapa de Calor (*HeatMap*) onde os valores de cada categoria são representados por uma tonalidade de cores, facilitando a identificação das categorias mais assertivas de cada algoritmo e as categorias passíveis de melhoria. As tonalidades mais claras representam um desempenho ruim, enquanto as tonalidades mais escuras representam um desempenho satisfatório. O mapa de calor do algoritmo *Naive Bayes,* é apresentado na Figura 19. É possível observar

que a categoria de falhas de negócio obteve o melhor desempenho enquanto as categorias de falhas físicas e desenvolvimento/interação são passíveis de melhoria.

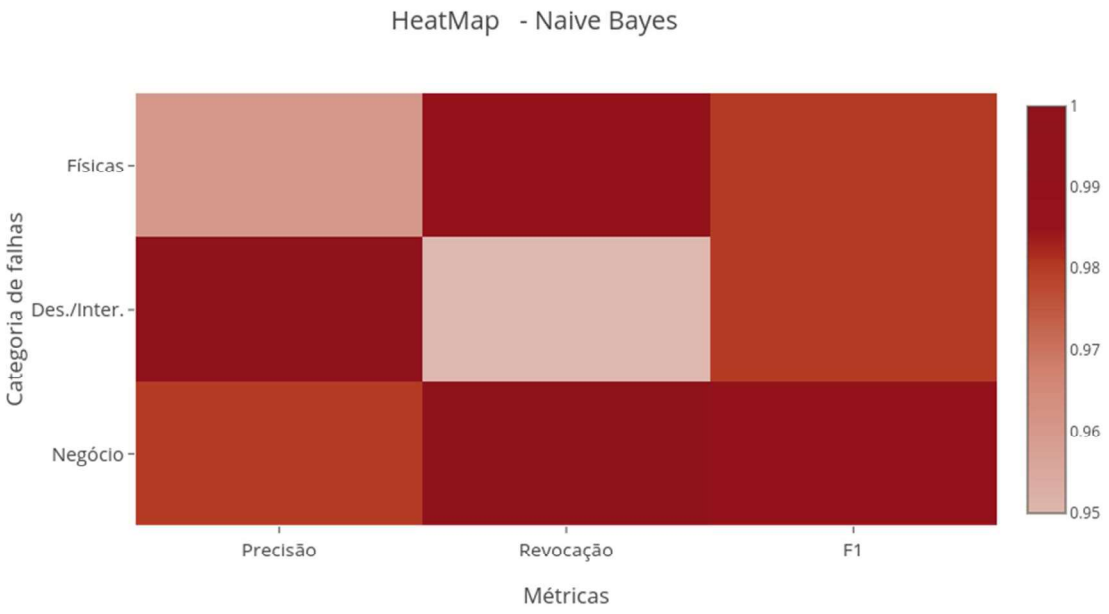

Figura 19 - Mapa de calor do relatório de classificação *Naive Bayes*

A matriz de confusão apresentada na Tabela 13 resume os resultados do experimento para o algoritmo *Naive Bayes.* Nesta matriz de confusão as 378 amostras de falhas físicas reais foram previstas corretamente, Das 1049 amostras de falha de interação reais o algoritmo previu que 2 eram falhas físicas. Das 760 amostras de falhas de desenvolvimento reais o algoritmo previu que 14 eram falhas físicas e 23 eram falhas de interação.

		PREDITO		
		Falhas Físicas	**Falhas de Desenvolvimento/ Interação**	**Falhas de Negócio**
VERDADEIRO	**Falhas Físicas**	378	0	0
	Falhas de Desenvolvimento/ Interação	14	723	23
	Falhas de Negócio	0	2	1047

Tabela 13 - Matriz de confusão para o algoritmo *Naive Bayes.*

Por fim, a Tabela 13 apresenta o relatório de classificação para o algoritmo *AdaBoost*. Como foi possível comprovar no primeiro experimento, esta técnica obteve o desempenho mais baixo em comparação com as demais técnicas apresentadas neste trabalho.

Categoria	**Métrica**		
	Precisão	**Revocação**	***F1***
Falhas Físicas	1,00	0,99	1,00
Falhas de Desenvolvimento/Interação	0,91	0,96	0,93
Falhas de Negócio	0,97	0,94	0,95
Média	**0,96**	**0,96**	**0,96**

Tabela 14 - Relatório de classificação *AdaBoost* para um conjunto de dados extraídos dos registros de *log*.

O mapa de calor do algoritmo *AdaBoost,* é apresentado na Figura 20. É possível observar que a categoria Desenvolvimento/Interação teve um desempenho ruim, em contrapartida a categoria física obteve um bom desempenho. No resultado geral é possível confirmar que este algoritmo obteve o pior desempenho em relação aos dois últimos apresentados para uma classificação de mensagens de falha em SOA.

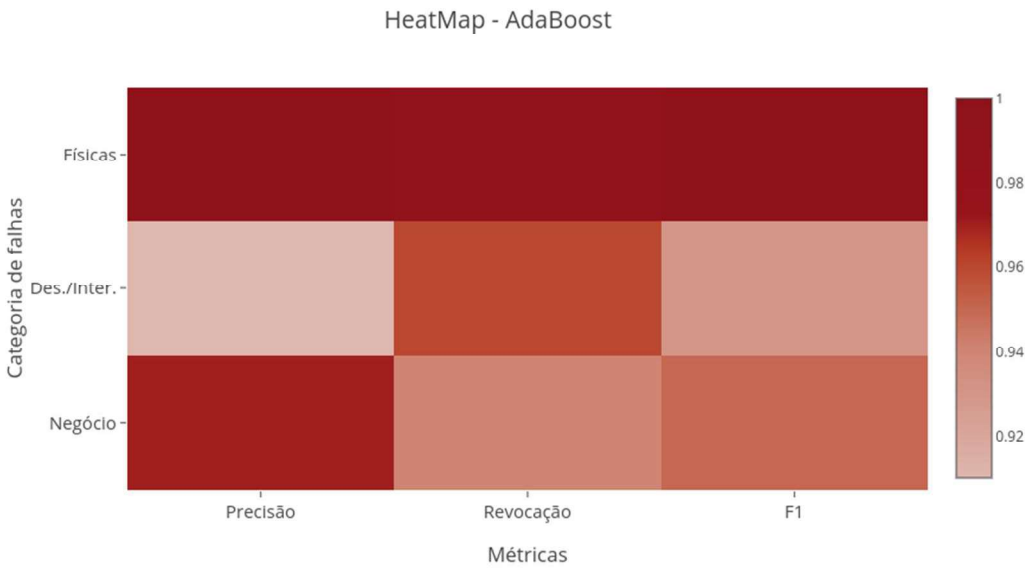

Figura 20 - *HeatMap* do relatório de classificação *AdaBoost*

A matriz de confusão apresentada na Tabela 15 resume os resultados do experimento para o algoritmo *AdaBoost.*

		PREDITO		
		Falhas Físicas	**Falhas de Desenvolvimento/ Interação**	**Falhas de Negócio**
VERDADEIRO	**Falhas Físicas**	378	0	0
	Falhas de Desenvolvimento/ Interação	2	735	23
	Falhas de Negócio	0	54	995

Tabela 15 - Matriz de confusão para o algoritmo *AdaBoost.*

Na matriz de confusão da Tabela 15 as 378 amostras de falhas físicas reais foram previstas corretamente. Das 1049 amostras de falha de interação reais o algoritmo previu que 54 eram falhas físicas. Das 760 amostras de falhas de desenvolvimento o algoritmo previu que 2 eram falhas físicas e 23 eram falhas de interação.

O gráfico de barras apresentado na Figura 21 apresenta uma comparação dos algoritmos de classificação de texto utilizando as métricas *F1,* revocação e precisão. Assim como na primeira experiência, pode-se observar que os algoritmos *SVM (OneVsRest* e *OneVsOne)* obtiveram o melhor desempenho.

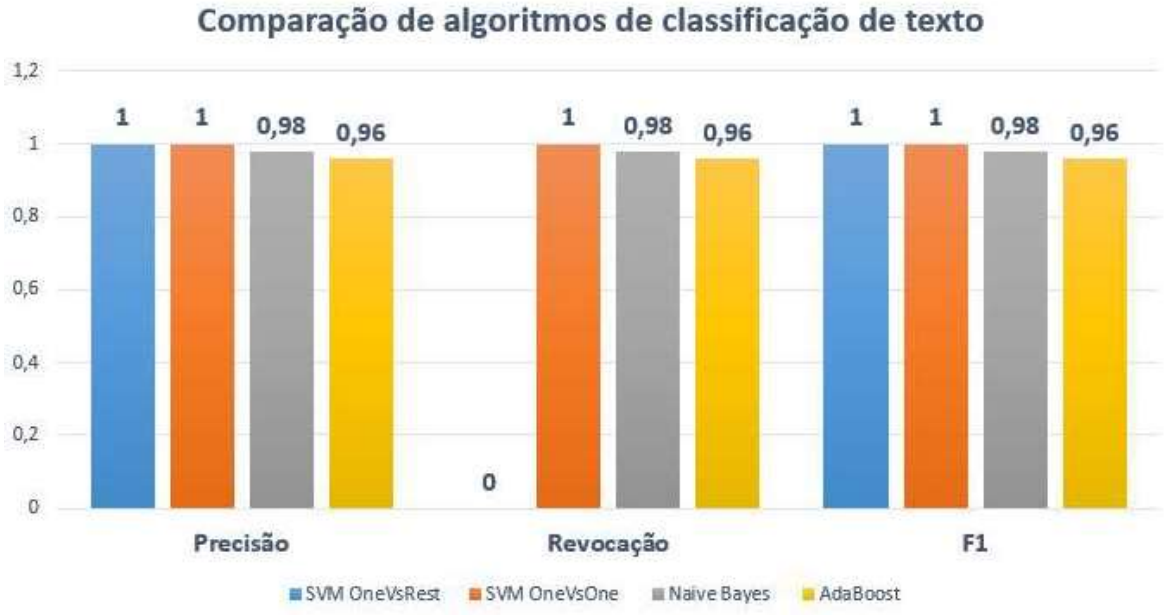

Figura 21 - Comparação de algoritmos utilizando as métricas: *F1,* revocação e precisão.

5.1.4 AVALIAÇÃO DE UM CONJUNTO DE DADOS COM AMOSTRAS EXCLUSIVAS

As primeiras avaliações (Seções 5.1.2 e 5.1.3) utilizaram um conjunto de dados extraído dos registros de *log*, com todas as falhas encontradas, essas avaliações são úteis para simular o desempenho do classificador no ambiente de produção, pois a extração das falhas demostrou que grande parte das falhas são recorrentes. O experimento realizado nessa seção visar simular o comportamento dos classificadores diante de novas falhas no ambiente. Esse experimento utilizou o primeiro conjunto de dados extraído, com as 162 falhas exclusivas. O método de validação *Cross Validation* foi utilizado e ajustado o valor de k=10, ou seja, 146 mensagens são utilizadas a cada rodada para treino do algoritmo e 16 amostras são utilizadas para testar o algoritmo. O objetivo desse esse tipo de experimento é realizar uma simulação, buscando prever o um cenário onde novas amostras de falhas inseridas no ambiente. Na fase

de extração de dados foi possível observar que muitos das falhas no ambiente são erros repetidos, porém o sistema deve estar apto a classificar novas mensagens de falhas inseridas no ambiente, como exemplo: novas plataformas acopladas ao barramento de serviços. As métricas utilizadas foram acurácia e tempo computacional. A Tabela 16 apresenta o resultado dessa execução.

Algoritmo	**Acurácia**	**Tempo Computacional (Segundos)**
SVM OneVsRest	0,9027	0,5201
SVM OneVsOne	0,8888	0,0952
Naive Bayes	0,8009	0,1359
AdaBoost	0,7685	1,3211

Tabela 16 - Resultados das métricas acurácia e tempo computacional para um conjunto de dados com amostras exclusivas extraídas dos registros de *log*.

É possível observar que, utilizando um conjunto de dados com amostras exclusivas, os algoritmos *SVM* continuaram com o melhor desempenho para a métrica acurácia. O experimento mostra que no pior cenário, os algoritmos *SVM* tem uma assertividade acima de 88%.

Com base nestes resultados, a Tabela 16 pode ser interpretada da seguinte forma, utilizando como exemplo o algoritmo *SVM OneVsRest:* para uma mensagem nova inserida no ambiente o sistema terá uma acurácia de 88% em comparação com a os dados apresentados na Tabela 9 que apresenta uma taxa de acerto de 99% para mensagens já conhecidas.

A Figura 22 apresenta a comparação dos resultados apresentados na Tabela 15.

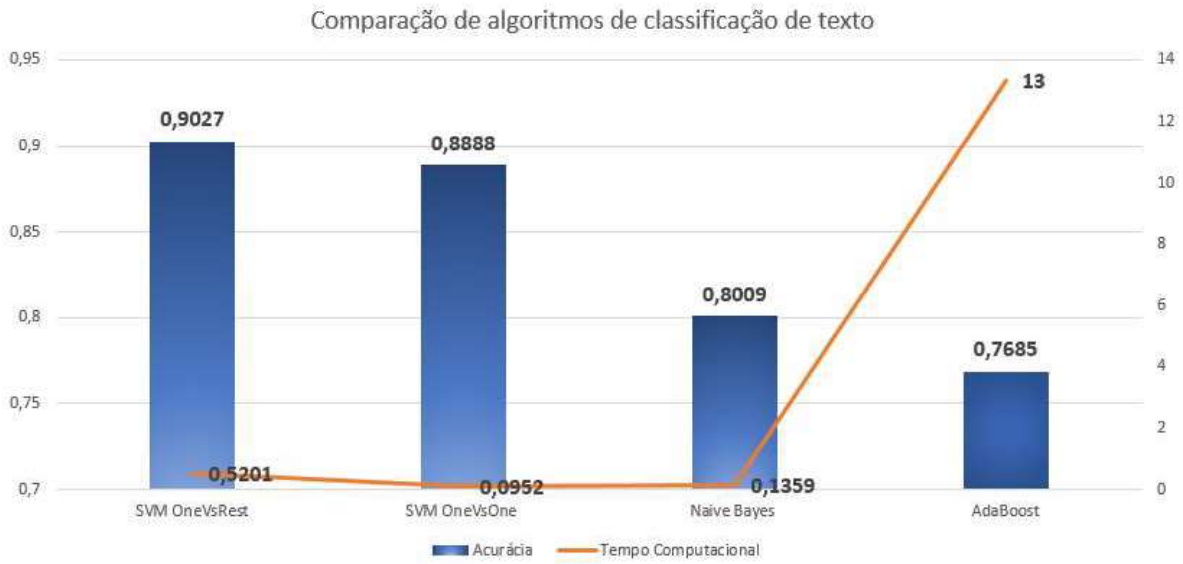

Figura 22 - Gráfico das métricas acurácia e tempo computacional para um conjunto de dados com amostras exclusivas extraídas dos registros de *log*.

5.1.5 CONCLUSÃO DOS EXPERIMENTOS DE CLASSIFICAÇÃO DE TEXTO

Foi possível observar a superioridade dos algoritmos SVM (*OneVsOne* e *OneVsRest*) na tarefa de classificação de mensagens de falhas em uma arquitetura orientada a serviços em todas as métricas apresentadas, utilizando um conjunto de dados com amostras extraídas dos registros de log, com exceção da métrica de tempo computacional, onde o algoritmo *Naive Bayes* obteve o melhor desempenho. Através dos experimentos foi possível validar que os algoritmos de classificação de texto SVM podem obter uma assertividade superior a 99% para falhas conhecidas (Tabela 8) e superior a 99% para novas falhas (Tabela 15). Essa avaliação utilizou a técnica *Cross Validation* (Seção 2.5.4), com um valor de k=10, ou seja, o conjunto de dados foi dividido em dez partes, nove para treino e uma para teste e após dez interações, onde cada parte da divisão foi utilizada como treino a média determinou a taxa de acerto.

Os resultados apontaram o algoritmo SVM *OneVsRest* como o ideal para a arquitetura SoaFaultControl, pois obteve um desempenho superior aos demais algoritmos classificando um conjunto de dados com amostras exclusivas e um tempo muito próximo ao SVM *OneVsOne* classificando um conjunto de dados com todas as amostras de falha retiradas

dos registros de *log* do período selecionado. Com relação ao tempo computacional, o algoritmo SVM *OneVsRest* teve um desempenho um pouco inferior ao SVM *OneVsOne* e *Naive Bayes*, porém essa diferença não é expressiva ao ponto de comprometer a performance do sistema.

6 CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS

Este capítulo apresenta as conclusões do presente trabalho e as sugestões para trabalhos futuros.

6.1 CONCLUSÃO

A SOA é uma das abordagens propostas para resolver o problema de heterogeneidade na integração de sistemas distribuídos, posicionando o serviço como um dos pilares desta arquitetura. Um ESB, implementação de um padrão de projeto SOA, é responsável por ocultar a heterogeneidade da implementação dos serviços distribuídos, expondo uma interface e transformando as requisições dos clientes para essa interface em algo compatível com a implementação real do serviço.

Este trabalho apresentou um problema comum na área de suporte a sistemas SOA: a classificação dos eventos de falha. Sem uma metodologia eficiente e automatizada este processo tende a ser lento e sujeito a erros, além de demandar mão de obra altamente especializada. Este trabalho propôs um método para automatizar o processo de classificação dos eventos de falha capturados por um ESB, denominado **SOAFaultControl**.

A classificação de falhas é essencial no processo de identificação da causa raiz de um problema. Se o sistema pode distinguir entre os diferentes tipos de falhas, pode-se então começar a aplicar um método correto de recuperação (CHAN, BISHOP, *et al.*, 2009).

Este trabalho teve como objetivo principal o desenvolvimento de uma arquitetura que utiliza técnicas de aprendizagem de máquina para automatizar a tarefa de classificação de falhas em uma arquitetura orientada a serviços. Foi possível atingir os objetivos propostos através da arquitetura apresentada no Capítulo 4 e os resultados do Capítulo 5.

6.2 RESPOSTA DA PERGUNTA DE PESQUISA E VALIDAÇÃO DA HIPÓSTESE

Após a realização dos experimentos apresentados neste trabalho, foi possível responder à pergunta de pesquisa definida na Seção 3.8: ***"Sobre uma classificação de mensagens de falha de uma Arquitetura Orientada a Serviços, dado um conjunto de dados pré-classificados, qual algoritmo de classificação que apresenta um melhor desempenho em termos de acurácia e tempo computacional?"***

Resposta: o algoritmo SVM apresentou o melhor desempenho nas métricas de acurácia em relação aos demais algoritmos apresentados. A técnica *OneVsRest* apresentou um desempenho melhor que a técnica *OneVsOne*. Os testes contemplaram dois conjuntos de dados: um conjunto com todas as falhas retiradas dos registros de *log* e um conjunto de dados apenas com amostras exclusivas. Os experimentos utilizaram a técnica de validação *Cross Validation* com um valor de $k = 10$. Além disso, para confirmar a resposta, outras métricas foram apresentadas: precisão, revocação *e f1,* onde o algoritmo *Support Vector Machine* com a técnica *OneVsRest* apresentou resultados semelhantes.

Também foi possível testar a hipótese definida na seção 3.8: ***"Fornecendo um conjunto de dados selecionados e pré-classificados manualmente para treino e efetuando um trabalho de eficiente de pré-processamento de texto, é possível selecionar um algoritmo de classificação que supere 80% de valores VP"***. O algoritmo SVM, com a técnica *OneVsRest* foi selecionada para a classificação de mensagens de falhas do método **SOAFaultControl**.

6.3 TRABALHOS FUTUROS

Como sugestão para continuidade deste trabalho, pode-se incluir:

- Sugerir correções para as falhas classificadas, com base no conhecimento adquirido de correções anteriores;
- Implementar correções automáticas de falhas, baseando-se nas classificações obtidas;
- Pesquisar a implementação de outros métodos de captura de falhas em sistemas distribuídos que podem ser adaptados à arquitetura do projeto;
- Melhorar a base de conhecimento, adicionando uma base de dados *NoSQL* mais robusta. A base de dados JsonDB é uma excelente opção para um projeto inicial, pois é simples e rápida, porém possui uma limitação de performance quando o número de registros cresce significativamente. Outras opções podem ser consideradas com o crescimento do sistema.
- Melhorar os dados estatísticos. Atualmente o projeto exibe apenas o número de ocorrências de uma determinada falha. Melhorando a modelagem de dados, outras informações podem ser adicionadas como: primeira e última ocorrência da falha, média por período, etc.

7 REFERÊNCIAS

ANDRADE, P. H. M. A. D. Aplicação de técnicas de mineração de textos para classificação de documentos: um estudo da automatização da triagem de denúncias na CGU. **Dissertação (Mestrado Profissional em Computação Aplicada)—Universidade de Brasília**, Brasília, p. xi, 54 f., il, 2015.

BRUNING, S.; WEISSLEDER, S.; MALEK, M. A fault taxonomy for service-oriented architecture. **High Assurance Systems Engineering Symposium, 2007. HASE'07. 10th IEEE}**, p. 367--368, 2007.

BUELOW, H.; KASI, J. **Getting started with oracle SOAsSuite 11g R1, a sands-on tutorial:** fast track your SOA adoption, build a service-oriented composite application in just hours! Packt Publishing Ltd, 2009.

CABRAL, M. A. L. Classificação automatizada de falhas tribológicas de sistemas alternativos com o uso de redes neurais artificiais não supervisionadas, Brasil, 2017.

CAMILO, C. O.; SILVA, J. C. D. Mineração de dados: conceitos, tarefas, métodos e ferramentas. **Universidade Federal de Goiás (UFC)**, Goiás, 2009. 1--29.

CAVALCANTI, L. C. A. M. Detecção de elementos antrópicos em imagens aéreas da floresta amazônica. **Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas**, Manaus, 2016.

CHAN, K. et al. A fault taxonomy for web service composition. **Springer service-oriented computing-ICSOC 2007 workshops**, p. 363--375, 2009.

CHAVES, B. B. Estudo do algoritmo adaboost de aprendizagem de máquina aplicado a sensores e sistemas embarcados. **Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo - Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos**, São Paulo, 2012.

CHHAJED, S. **Learning ELK Stack**. Packt Publishing, 2015.

COULOURIS, G. **Distributed systems:** concepts and design. Addison-Wesley, 2005.

DOS SANTOS, L. S. F. C. Estudo online da dinâmica espaço-temporal de crimes através de dados da rede social twitter. **UFMG**, 2015.

DOSCIATTI, M. M. Um método para a identificação de emoções básicas em textos em português do Brasil usando máquinas de vetores de suporte em solução multiclasse. **Tese (doutorado) – Pontifícia Universidade Católica do Paraná**, Curitiba, 2015.

ERL, T. **SOA:** Princípios de design de serviços. Peterson Prentice Hall, 2009.

GONÇALVES, A. R. Máquina de Vetores Suporte, 2007. Disponivel em: <http://www-users.cs.umn.edu/~andre/arquivos/pdfs/svm.pdf>. Acesso em: 09 abr. 2017.

HAN, J.; PEI, J.; KAMBER, M. **Data mining:** concepts and techniques. Elsevier, 2011.

HU, W. et al. Online adaboost-based parameterized methods for dynamic distributed network intrusion detection. **IEEE Transactions on Cybernetics**, v. 44, n. 1, p. 66--82, 2014.

JOSUTTIS, N. M. **SOA na prática, a arte da modelagem de sistemas distribuídos**. Alta Books, 2008.

KACMAJOR, T. SVM model selection - how to adjust all these knobs, 2016. Disponivel em: <http://tomaszkacmajor.pl/index.php/2016/05/01/svm-model-selection2>. Acesso em: 19 maio 2017.

KAVULYA, S. P. et al. **Draco:** statistical diagnosis of chronic problems in large distributed systems. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012). 2012. p. 1--12.

KAZEMIAN, H.; AHMED, S. Comparisons of machine learning techniques for detecting malicious webpages. **Elsevier Expert Systems with Applications**, v. 42, n. 3, p. 1166--1177, 2015.

LAGUNA, F. Um guia para o corpo de conhecimento de análise de negócios(TM) (Guia BABOK). **International Institute of Business Analysis**, 2011. ISSN ISBN 9780981129242.

LORENA, A. C.; DE CARVALHO, A. Introduçao as máquinas de vetores suporte. **Relatório técnico do instituto de ciências matemáticas e de computação (USP)**, São Carlos, v. 192, 2003.

LORENA, A. C.; DE CARVALHO, A. Uma introdução às support vector machines. **Revista de Informática Teórica e Aplicada**, v. 14, n. 2, p. 43--67, 2007.

MILGRAM, J. A. C. M. A. S. R. “One against one” or “one against all”: Which one is better for handwriting recognition with SVMs? **Tenth international workshop on frontiers in handwriting recognition**, n. Suvisoft, 2006.

MITCHELL, T. M. Machine learning. 1997. **Burr Ridge, IL: McGraw Hill**, v. 45, n. 37, p. 870--877, 1997.

MUMBAIKAR, S.; PADIYA, P.; OTHERS. Web services based on soap and rest principles. **International Journal of Scientific and Research Publications**, v. 3, 2013. ISSN 5.

NAGARAJ, K.; KILLIAN, C.; NEVILLE, J. **Structured comparative analysis of systems logs to diagnose performance problems**. Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). 2012. p. 353--366.

NISA, R.; QAMAR, U. A text mining based approach for web service classification. **Springer Information Systems and e-Business Management**, v. 13, n. 4, p. 751--768, 2015.

OGURI, P. Aprendizado de Máquina para o Problemade Sentiment Classification. **Dissertação (Mestrado) - Pontifícia Universidade Católica do Rio de Janeiro**, Rio de Janeiro, 2006.

ORACLE_OSB. Fusion Middleware Administrator's Guide for Oracle Service Bus. **Oracle Help Center**, 2017. Disponivel em: <https://docs.oracle.com/cd/E23943_01/admin.1111/e15867/app_error_codes.htm#OSBAG1390>. Acesso em: 17 out. 2017.

ORACLE_WEBLOGIC. Oracle Help Center. **WebLogic Server Error Messages Reference**, 2017. Disponivel em:

<https://docs.oracle.com/cd/E24329_01/doc.1211/e26117/chapter_bea_messages.htm#sthref7>. Acesso em: 17 out. 2017.

ORACLE_WORKFORCE. Oracle. **Activity Management SDK for Oracle Field Service**, 2015. Disponivel em: <https://docs.oracle.com/cloud/august2015/servicecs_gs/FAEAM/FAEAM.pdf>. Acesso em: 17 out. 2017.

PAZOGLOU, M. P.; HEUVEL, V. D.; WILLEM-JAN. Service oriented architectures: approaches, technologies and research issues. **Springer The VLDB journal**, v. 16, p. 389--415, 2007. ISSN 3.

PHAM, C. et al. Failure diagnosis for distributed systems using targeted fault injection. **IEEE Transactions on Parallel and Distributed Systems**, v. 28, n. 2, p. 503--516, 2017.

POLIKAR, R. Ensemble based systems in decision making. **IEEE Circuits and systems magazine**, v. 6, n. 3, p. 21--45, 2006.

RAY, S. 6 easy steps to learn naive bayes algorithm. **Analytic Svidhya**, 2017. Disponivel em: <https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/>. Acesso em: 14 out. 2017.

REIDEMEISTER, T.; MUNAWAR, M. A.; WARD, P. A. Identifying symptoms of recurrent faults in log files of distributed information systems. **IEEE Network Operations and Management Symposium-NOMS**, p. 187--194, 2010.

ROKACH, L. Ensemble-based classifiers. **Artificial Intelligence Review. Springer**, v. 33, n. 1, p. 1--39, 2010.

RUSSELL, S. J..; NORVIG, P. **Artificial Intelligence:** A Modern Approach The Intelligent Agent Book. Prentice Hall, 1995.

S, D.; RAJ, P.; S.RAJARAAJESWARI. 1st International Conference on “Innovations in Computing & Networking” (ICICN-2016). **International Journal of Advanced Networking & Applications (IJANA)**, Bangalore, 2016. ISSN ISSN: 0975-0282.

SAUDATE, A. **SOA aplicado:** Integrando com web services e além. Código, Editora Casa do, 2014.

SILVA, R. M.; ALMEIDA, T. A.; YAMAKAMI, A. Análise de desempenho de redes neurais artificiais para classificação automática de web spam. **Revista Brasileira de Computação Aplicada**, Passo Fundo, v. 4, n. 2, p. 42-57, out. 2012.

STATION, I. C. Best ESB Solutions, 2017. Disponivel em: <https://www.itcentralstation.com/categories/esb>. Acesso em: 06 abr. 2017.

SUN, J. et al. Dynamic financial distress prediction with concept drift based on time weighting combined with Adaboost support vector machine ensemble. **Elsevier Knowledge-Based Systems**, v. 120, p. 4--14, 2017.

TAN, P.; STEINBACH, M.; KUMAR, V. **Introdução ao data mining:** mineração de dados. Ciência Moderna, 2009.

TECNOLOGIA, R. B. D. W. Matriz de Confusão, 2015. Disponivel em: <http://www.revistabw.com.br/revistabw/matriz-de-confusao>. Acesso em: 04 28 2017.

VAARANDI, R.; PIHELGAS, M. **Logcluster-a data clustering and pattern mining algorithm for event logs**. IEEE Network and Service Management (CNSM), 2015 11th International Conference on. [s.n.]. 2015. p. 1--7.

VAPNIK, V. **The nature of statisticsal learning**. NY: Springer, 1995.

W3C. Simple Object Access Protocol (SOAP) 1.1. **W3C Note**, 2000. Disponivel em: <https://www.w3.org/TR/2000/NOTE-SOAP-20000508/>. Acesso em: 11 jul. 2007.

WAZLAWICK, R. Análise e design orientados a objetos para sistemas de informação: modelagem com UML, OCL e IFML. **Elsevier Brasi**, 2016. ISSN ISBN 9788535279856.

WEBER, T. S. Tolerância a falhas: conceitos e exemplos. **Apostila do progama de pós-graduação--instituto de informática-UFRGS**, Porto Alegre, 2003.

WILLIAMS, N.; ZANDER, S.; ARMITAGE, G. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. **ACM SIGCOMM Computer Communication Review**, v. 36, p. 5--16, 2006.

XUAN, J. et al. Automatic bug triage using semi-supervised text classification. **Proceedings of International Conference on Software Engineering & Knowledge Engineering**, Redwood City, p. 209–214, 2010.

APÊNDICE A – MODELAGEM SISTÊMICA DO NÚCLEO SOAFAULTCONTROL

A Figura 23 apresenta o diagrama de classes da no núcleo da arquitetura SOAFaultControl.

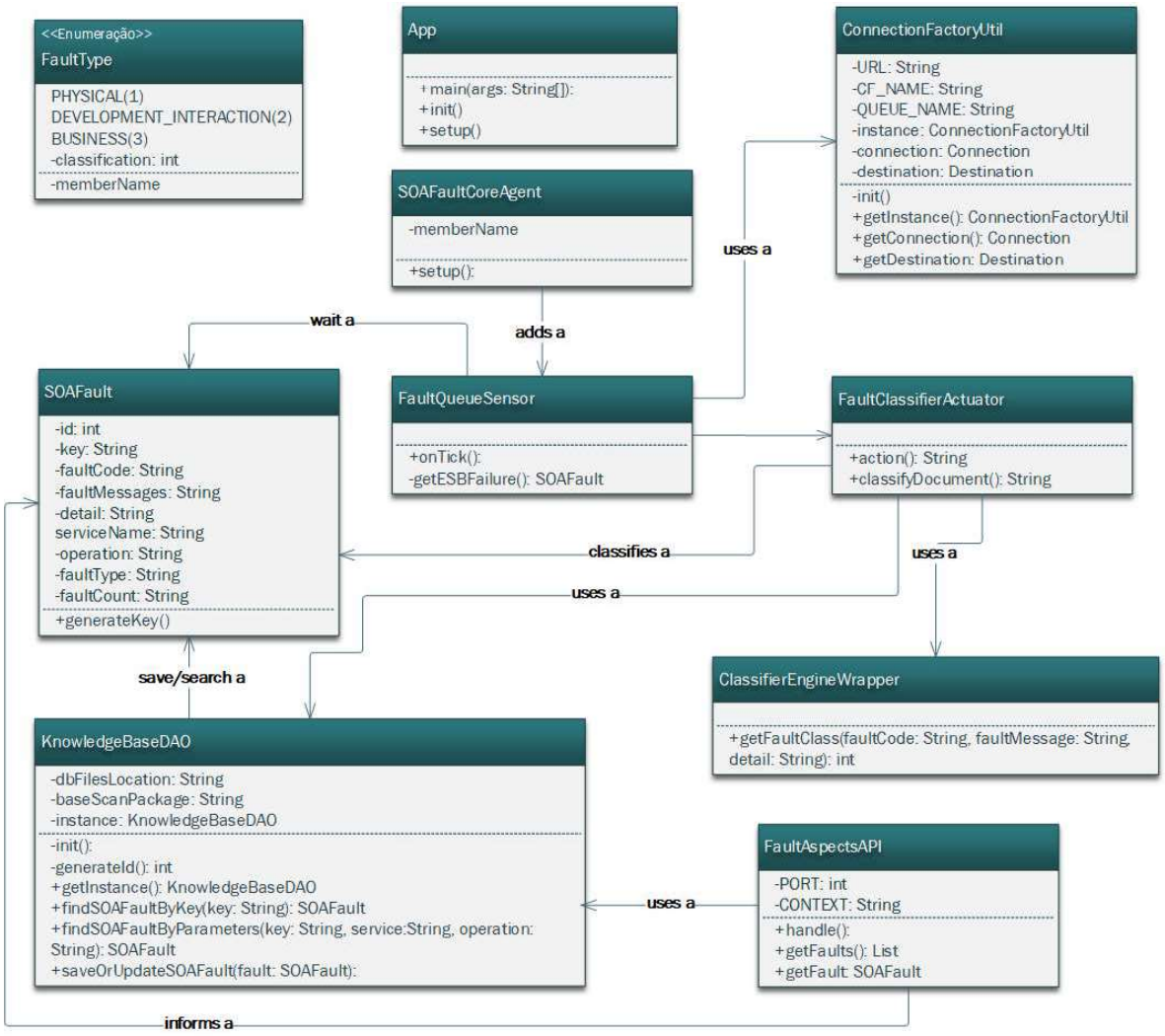

Figura 23 – Diagrama de classes da arquitetura SOAFaultControl

A Tabela 17 apresenta a descrição das classes apresentadas na Figura 23.

Classe	Descrição
App	Classe responsável em iniciar o sistema SOAFaultControl. Realiza as configurações iniciais e cria os agentes que controlam o ambiente.
SOAFaultCoreAgent	Representa um agente que implementa o núcleo do sistema SOAFaultControl. É o componente principal da arquitetura, responsável em coordenar o trabalho de todos os módulos do sistema.
FaultQueueSensor	Representa um sensor executa periodicamente um comportamento que valida a fila de mensagens de falhas capturadas pelo ESB. Se detecta a existência de um novo evento de falha delega o evento para um atuador.
FaultClassifierActuator	Representa um atuador responsável em tratar um evento de falha percebido pelos sensores. Como parte de seu comportamento verifica se a falha já foi classificada através de uma consulta a base de conhecimento, caso já exista uma classificação atualiza os dados estatísticos, se a falha é nova interage com o motor de classificação para prever a categoria da falha.
SOAFault	Representa um evento de falha na arquitetura SOAFaultControl.
ClassifierEngineWrapper	Implementa um ponto de interação com a API REST do motor de classificação.
KnowledgeBaseDAO	Implementa funções de acesso à base de conhecimento.
FaultAspectsAPI	Disponibiliza uma API REST com as funções de acesso ao sistema SOAFaultControl. Também implementa um servidor HTTP embarcado que disponibiliza uma página HTML para os usuários do sistema.
ConnectionFactoryUtil	Classe com um conjunto de função auxiliares.
FaultType	*Enumeration* com as constantes que representam uma classe de falha.

Tabela 17 – Descrição das classes da arquitetura SOAFaultControl.

A Figura 24 apresenta um diagrama representado a sequência de eventos do núcleo do sistema SOAFaultControl.

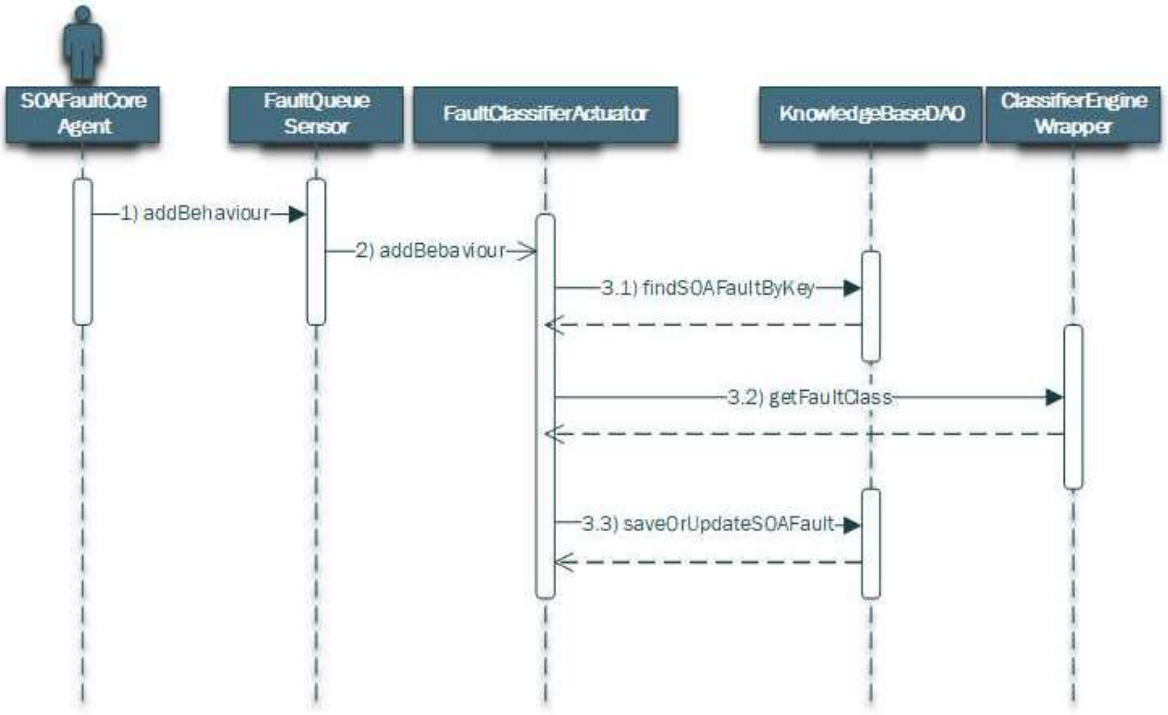

Figura 24 – Diagrama de sequencia do processo de classificação de mensagens de falhas – Visão do núcleo do sistema.

O diagrama de sequência apresentado na Figura 24, mostra o processo de classificação de falhas do ponto de vista da sequência de operações executadas pelo agente ***SOAFaultCoreAgent***, que implementa o núcleo do sistema SOAFaultControl. Abaixo uma descrição desse fluxo:

1. Na inicialização do agente ***SOAFaultCoreAgent*** é adicionado um comportamento, ou seja, uma tarefa sistematizada. Este comportamento desempenha um papel de sensor, monitorando frequentemente a fila JMS com as falhas do ESB, aguardando um novo evento de falha;
2. Ao encontrar um evento, o sensor ***FaultQueueSensor*** adiciona um comportamento que trabalha no papel um atuador, buscando a classificação adequada para a falha;
3. A sequência de ações do atuador ***FaultClassifierActuator***:
   3.1. Pesquisa na base de conhecimento se a falha já foi classificada;

3.2. Caso a falha seja nova, executa a API REST do motor de classificação, através de um objeto da classe ***ClassifierEngineWrapper***;

3.3. Cria um novo registro na base de conhecimento para novas falhas ou atualiza o registro com a falha classificada.

APÊNDICE B – MODELAGEM SISTÊMICA DO MOTOR DE CLASSIFICAÇÃO

A Figura 25 apresenta o diagrama de classes do motor de classificação da arquitetura SOAFaultControl.

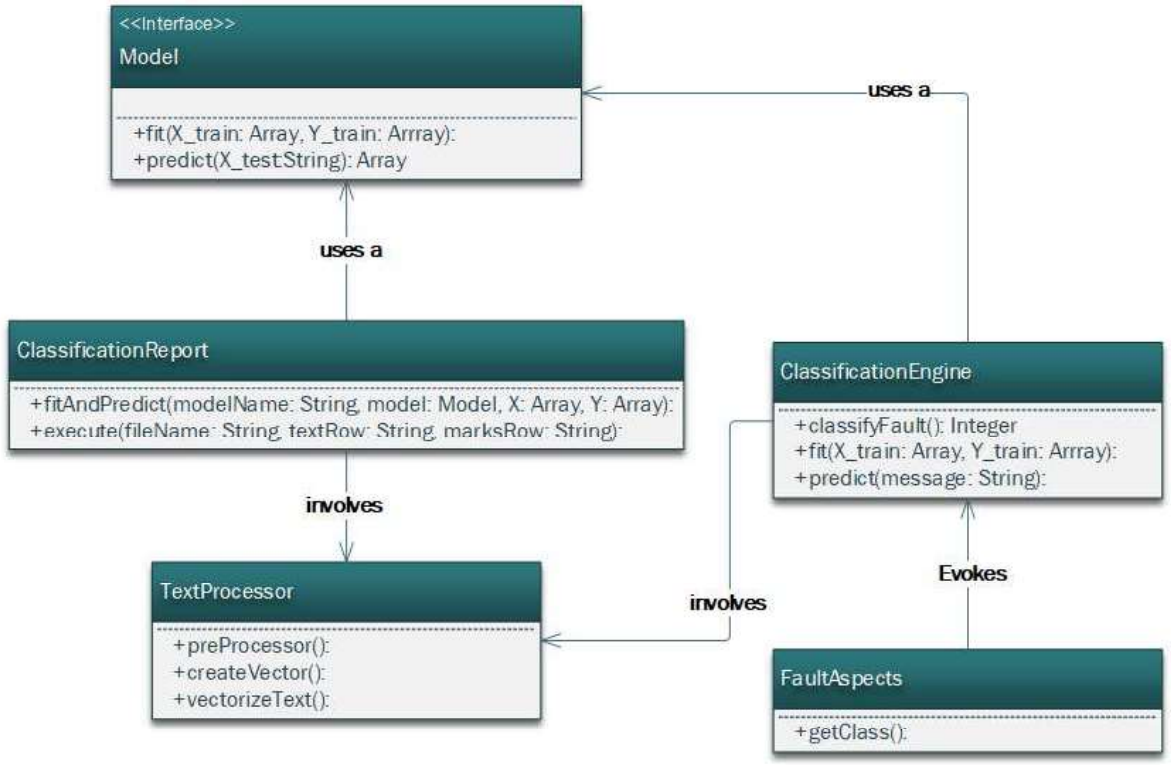

Figura 25 – Diagrama de classes do motor de classificação

A Tabela 18 apresenta a descrição das classes apresentadas na Figura 25.

Classe	Descrição
Model	Interface que representa um dos modelos de aprendizado de máquina disponíveis na biblioteca *scikit-learn* (ver Seção 3.1).
ClassificationReport	Classe utilizada para gerar o relatório de classificação. Informando um conjunto de dados, esse relatório compara vários modelos de classificação calculando as métricas: acurácia, precisão, revocação e F1. Também gera uma matriz de confusão.
ClassificationEngine	Classe responsável em treinar um modelo de classificação e prever uma categoria de falha.
FaultAspects	Classe que implementa uma API REST responsável em expor a operação que classifica uma mensagem de falha.
TextProcessor	Classe responsável pelo Pré-processamento de texto. Essa fase realizada imediatamente após a coleta de dados para obter as características de classificação de texto. Tem o objetivo de melhorar a qualidade dos dados disponíveis e organizá-los. As seguintes etapas são executadas: *Case Folding*, *Tokenização*, Remoção de palavras repetidas, Validação do tamanho da palavra, Remoção das *stop words* e *Stemming* (ver Seção 4.4.1)

Tabela 18 – Descrição das classes do motor de classificação.

A Figura 26 apresenta um diagrama representado a sequência de eventos do processo de classificação de falhas do motor de classificação.

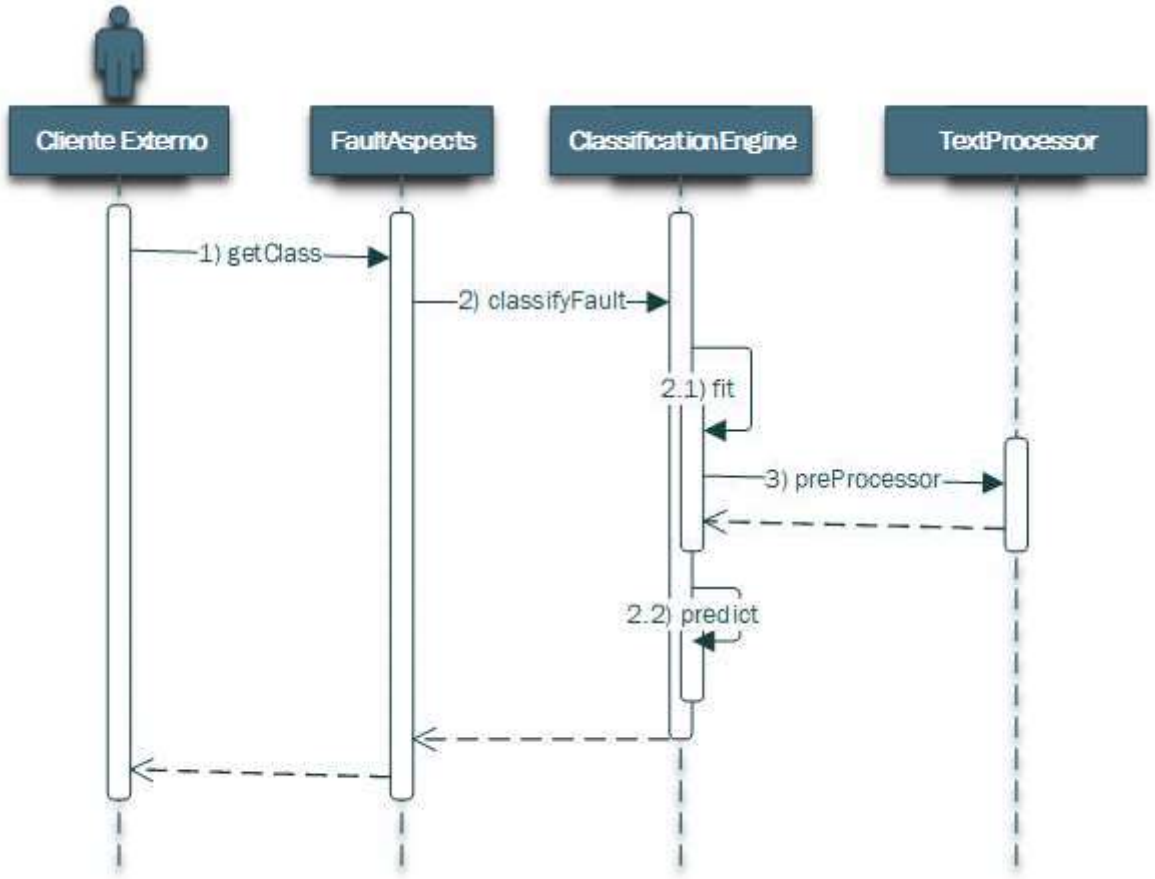

Figura 26 - Diagrama de sequencia do processo de classificação de mensagens de falhas – visão do motor de classificação.

O diagrama de sequência apresentado na Figura 26, mostra a classificação de falhas do ponto de vista da sequência de operações executadas pelo motor de classificação, quando acionado pela API REST ***FaultAspects*** por um cliente externo, como o núcleo do sistema. Abaixo uma descrição desse fluxo:

1. Cliente externo executa a operação *getClass* da API ***FaultAspects***;
2. API aciona o motor de classificação;
   2.1. Se é a primeira classificação executada pelo motor após a inicialização do sistema, o modelo de classificação é treinado;
   2.2. A mensagem de falha é classificada;
3. O texto é Pré-processado.

APÊNDICE C – AMOSTRAS DE MENSAGENS DE ERRO

Conforme citado no Capítulo 3, mensagens de erro de fabricantes de *software* foram adicionadas aos registros de *log* do ESB, essa injeção de mensagens teve por objetivo enriquecer estes registros e manter o mais próximo de um registro de produção com possíveis mensagens que podem ser geradas neste ambiente.

A Tabela 19 contém uma lista de mensagens classificadas como "Desenvolvimento/Interação". Estes erros podem ocorrer no OSB quando um serviço de *proxy* está sendo executado. Uma listagem completa pode ser consultada em (ORACLE_OSB, 2017).

Código	Mensagem de erro
BEA-382030	General parse failure from binding layer (e.g. message to XML service is not XML)
BEA-382032	Message must be a soap:Envelope XML Details: "A Non-SOAP or Invalid Envelope Was Received"
BEA-382033	A soap:Envelope must contain a soap:Body
BEA-382040	Failed to assign value to context variable "{0}". Value must be an instance of {1}
BEA-382041	Failed to assign value to context variable "{0}". Variable is read-only.
BEA-382045	Failed to initialize the value of context variable "{0}": {1}
BEA-382103	General binding error while processing outbound response
BEA-382104	Failed to prepare request metadata for service {0}
BEA-382150	Failed to dispatch request to service {0}
BEA-382151	Cannot dispatch to unknown service: {0}
BEA-386400	General outbound web service security error
BEA-386401	Failed to convert outbound message to SOAP
BEA-386402	Cannot determine the outbound operation
BEA-386420	A web service security error occurred while producing security header
BEA-386460	Web Service Security policy validation error
BEA-394500	An error was encountered while importing a resource
BEA-394502	An error was encountered while initializing the UDDI service
BEA-394505	Failed to connect to the UDDI registry
BEA-394506	An error was encountered while querying the UDDI registry for business services
BEA-394507	The registry name contains characters that are not allowed
BEA-394508	The publish URL was missing the UDDI registry configuration
BEA-394509	The Service Account configured is not valid
BEA-394510	A resource could not imported while importing a service
BEA-394512	A generic error was encountered while importing a resource

Tabela 19 – Amostras de mensagens de erro classificadas como “Desenvolvimento/Interação”.

A Tabela 20 contém uma lista de mensagens de erro classificadas como “Física”. Uma listagem completa pode ser consultada em (ORACLE_WEBLOGIC, 2017).

Código	Mensagem de erro
BEA-001112	Test "{1}" set up for pool "{0}" failed with exception: "{2}".
BEA-001131	Received an exception when closing a cached statement for the pool "{0}": {1}.
BEA-001150	Connection pool "{0}" deployment failed with the following error: {1}.
BEA-002606	The server is unable to create a server socket for listening on channel "{3}". The address {0} might be incorrect or another process is using port {1,number,0}: {2}
BEA-300030	The socket close failed. Reason: {0}.
BEA-300048	Unable to start the server {0} : {1}
BEA-300049	An I/O error occurred while writing the server URL file: "{0}".
BEA-310003	Free memory in the server is {0} bytes. There is danger of receiving an OutOfMemoryError.
BEA-310006	Critical subsystem {0} has failed. Setting server state to FAILED. Reason: {1}
BEA-320004	The following unexpected exception was thrown: {0}]
BEA-002005	OutOfMemoryError in adapter: {0}.
BEA-001555	Invalid member data source {1} specified for multi data source {0}. GridLink data source may not be used in multi data source configurations.
BEA-001151	Data source "{0}" deployment failed with the following error: {1}.
BEA-001131	Received an exception when closing a cached statement for the pool "{0}": {1}.
BEA-000172	TCP/IP socket failure occurred while fetching state dump over HTTP from {0}. Error message received: {2}. The request was generated by {3}. {1}
BEA-000170	The server {0} did not receive the multicast packets that it sent.
BEA-000141	TCP/IP socket failure occurred while fetching state dump over HTTP from {0}.

Tabela 20 - Amostras de mensagens de erro classificadas como “Física”.

A Tabela 21 contém uma lista de mensagens de erro classificadas como “Negócio”. Essas mensagens de erro são geradas por serviços externos ao ESB que podem ser chamados pelo barramento, simulam uma mensagem de falha de um serviço de negócio. Uma listagem completa pode ser consultada em (ORACLE_WORKFORCE, 2015).

Código	Mensagem de erro
0	Permission denied: operation='get_activity', user='admin' user has no permission for the action
9	Parameter 'position_in_route' is equal to 'activity_id' wrong activity position within a route
11	Can't start activity: start_appointment: The appointment starting order is invalid. activity cannot be started because it is ordered and is not the first pending activity in the route
15	Can't start activity: 9382903: start_appointment: The appointment cannot be started at the specified time.
16	Can't link activities:9382910: add_appt_link: Action on past date is not allowed. action cannot be performed on the date (e.g cancel activity in the past or start in the future)
17	Can't create activity: 9382905: insert_appointment: The 'appt.team_id' mandatory field is not assigned. mandatory field in request is missing in the request
18	Search failed: 1679041228: search_appointments: 'gfh' is not a valid select_count value wrong value of a parameter in the request
19	Activity not found: id=7996012 requested object is not found
20	Can't update activity: 9382904: update_appointment: Data has been changed record to be updated changed by another user
23	Can't create activity: 9382903: insert_appointment: Inconsistent data: sla_window_start > sla_window_end (2014-01-26 01:00:00 > 2014-01-22 01:00:00)
27	Cannot set 'min_interval' for this kind of link activity link does not support modification of minimal/maximal interval

Tabela 21 - Amostras de mensagens de erro classificadas como “Negócio”.