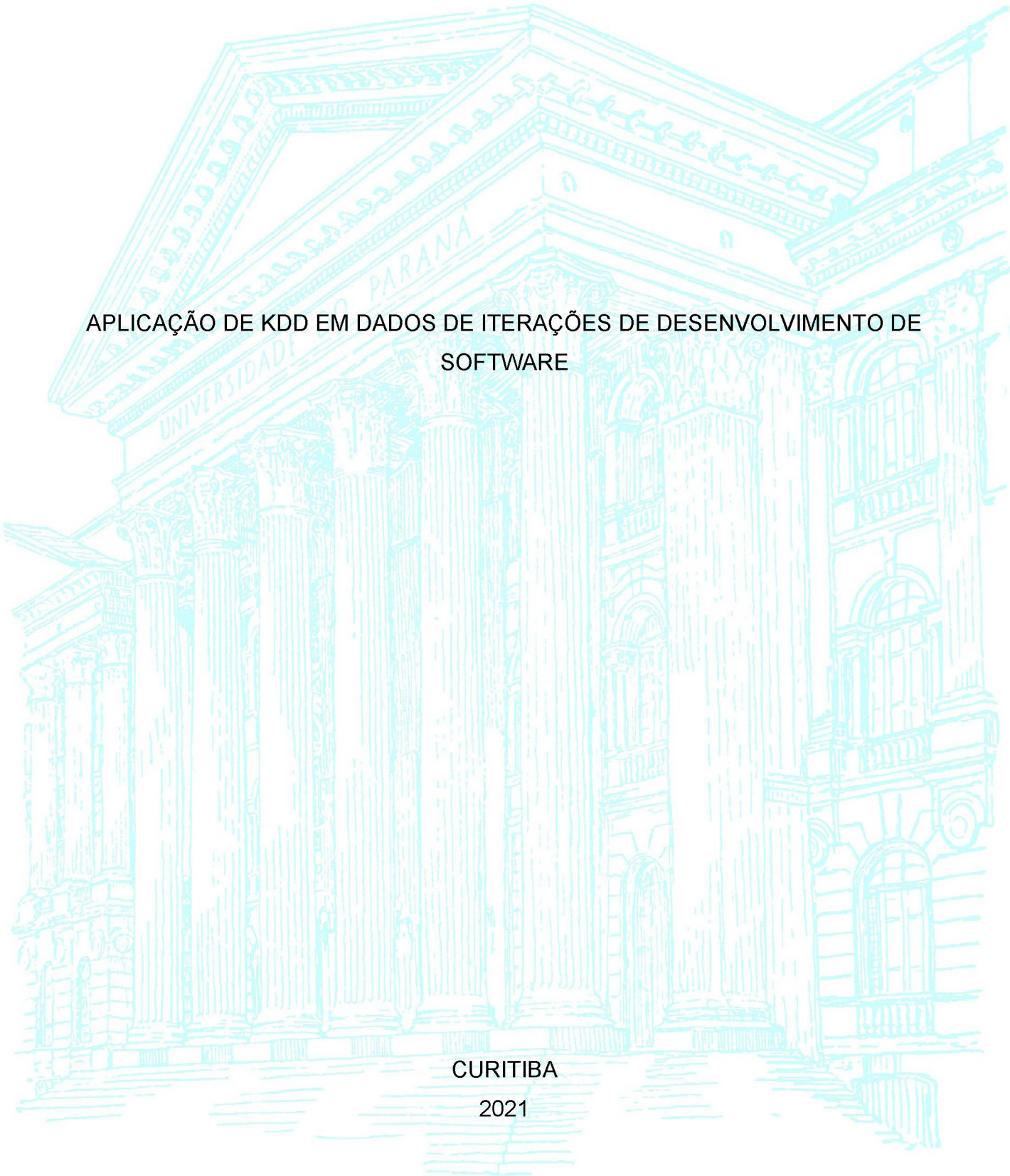
UNIVERSIDADE FEDERAL DO PARANÁ

VAGNER CARLOS MARCOLINO LIMA



APLICAÇÃO DE KDD EM DADOS DE ITERAÇÕES DE DESENVOLVIMENTO DE SOFTWARE

CURITIBA

2021

VAGNER CARLOS MARCOLINO LIMA

APLICAÇÃO DE KDD EM DADOS DE ITERAÇÕES DE DESENVOLVIMENTO DE SOFTWARE

Trabalho de Conclusão de Curso apresentada ao curso de Pós-Graduação em Inteligência Artificial Aplicada, Setor de Educação Profissional e Tecnológica, Universidade Federal do Paraná, como requisito parcial à obtenção do título de Especialista em Inteligência Artificial Aplicada.

Orientador: Prof. Dr. Jaime Wojciechowski

CURITIBA

2021

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INTELIGÊNCIA ARTIFICIAL APLICADA da Universidade Federal do Paraná foram convocados para realizar a arguição da Monografia de Especialização de **VAGNER CARLOS MARCOLINO LIMA** intitulada: **Aplicacao de KDD em dados de Iteracoes de Desenvolvimento de Software**, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua __APROVAÇÃO__ no rito de defesa.
A outorga do título de especialista está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 19 de Abril de 2021.

JAIME WOJCIECHOWSKI
Presidente da Banca Examinadora (UNIVERSIDADE FEDERAL DO PARANÁ)

RAZER ANTHOM NIZER ROJAS MONTAÑO
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Aplicação de KDD em dados de Iterações de Desenvolvimento de Software

Vagner Carlos Marcolino Lima
SEPT - Setor de Educação Profissional e Tecnológica
UFPR - Universidade Federal do Paraná
Curitiba, Brasil
vagnercml@hotmail.com

Jaime Wojciechowski
SEPT - Setor de Educação Profissional e Tecnológica
UFPR - Universidade Federal do Paraná
Curitiba, Brasil
jaimewo@ufpr.br

***Resumo* – O software está presente em várias categorias de sistemas atualmente. Ele pode ser desenvolvido de diferentes formas, o Desenvolvimento Iterativo é uma delas. Esse modelo de processo de software é a base para processos ou métodos amplamente praticados na indústria de software. Ele é capaz de lidar mais facilmente com softwares modernos, os quais estão cada vez complexos e exigindo entregas mais frequentes que agreguem valor ao serviço ou produto do cliente. Nessa abordagem o software é desenvolvido por meio de várias iterações. Esse processo pode gerar um grande volume de dados tornando-se inviável uma análise manual. É comum, portanto, que grandes empresas apliquem métodos ou processos específicos para obter conhecimento a partir desses dados para na sequência difundi-lo dentre os seus integrantes buscando melhorar seus processos produtivos. Nesse contexto, o objetivo deste trabalho foi identificar modelos e padrões que sejam válidos e potencialmente interpretáveis de tal forma que possam auxiliar as equipes de software na tomada de decisões inerente à realização de iterações de desenvolvimento de software. Esse objetivo foi norteado pelas seguintes questões, dadas as informações históricas dessas iterações: "é possível prever o resultado de uma iteração?" e "esses dados podem revelar algum padrão relacionado às práticas das equipes de software?". Para alcançar esse objetivo, foi aplicado um método de estudo baseado no *KDD-process* envolvendo dados de projetos reais e as técnicas de mineração de dados Árvore de Decisão e Floresta Aleatória. Foram identificados modelos capazes de prever o sucesso (ou falha) de uma iteração de software com até 93% de acurácia e padrões que evidenciam, por exemplo, a importância da prática de constantemente analisar e refinar os itens do *backlog* do produto.**

Palavras-chave* – *Processo de software, Desenvolvimento de Software, Iteração, Sprint, KDD, Mineração de dados, Modelos preditivos, Padrões

I. INTRODUÇÃO

O software está presente em várias categorias de sistemas, tais como, sites web, aplicativos de celulares, sistemas embutidos em carros, ferramentas de análise de dados e outros. O software é tanto o produto – de um processo de elaboração conduzido por profissionais – quanto o meio utilizado para seu desenvolvimento e entrega – por exemplo, linguagens de programação e ferramentas. O desenvolvimento de software pode ser realizado de diferentes formas, mas, em geral, é instanciado e realizado com base em modelos de processo de software definidos tanto pela comunidade acadêmica quanto pela indústria de software. Pode-se citar como exemplo desses modelos de processo de software o Desenvolvimento Iterativo. Nesse modelo o produto de software é resultado da realização de uma ou mais iterações de desenvolvimento de software [1], [2].

Uma iteração ou sprint de desenvolvimento de software é um ciclo de trabalho que envolve a realização de ações e/ou atividades de engenharia de software e de apoio de tal maneira que ao final dele uma parte do sistema está em condições de ser entregue ao cliente. Essa parte ou incremento de software é resultado, por exemplo, de ações e/ou atividades de definição de requisitos, projeto de sistema e software, implementação e teste de unidade, integração e teste de sistema e operação. No desenvolvimento iterativo, portanto, o software ou solução se torna mais e mais completo ao final de cada ciclo [2]–[4]. O desenvolvimento iterativo e incremental é base de vários processos e métodos de desenvolvimento de software. Ele é amplamente praticado na indústria dado que os projetos de software modernos estão cada vez complexos e exigindo entregas mais frequentes que agreguem valor ao serviço ou produto do cliente [2], [5].

Há um aumento na quantidade e na variedade de dados nas empresas nos últimos anos, isso em decorrência de diferentes fatores, tais como: a crescente popularização e uso da Internet, o surgimento de (novas) tecnologias de conectividade – por exemplo, Internet das Coisas (IoT), Redes Sociais e Automação residencial e outras –, a diversidade de fontes de dados internas e externas às empresas e hardware cada vez mais potentes e com maior capacidade de armazenamento [6], [7]. No contexto do desenvolvimento de software não é diferente. O processo de software gera um grande volume de dados tornando-se inviável uma análise manual. É comum, portanto, que grandes empresas apliquem métodos ou processos de análise de dados para obter conhecimento a partir deles e, na sequência, difundi-lo dentre os integrantes de suas equipes de software para que eles possam melhorar seus processos produtivos [8].

Existem diversos métodos ou processos de análise de dados utilizados no mercado e na academia. Eles, no geral, possuem etapas equivalentes e a mesma finalidade, que é auxiliar na definição do ciclo de vida dos projetos de análise de dados. Neste contexto, ressalta-se o processo de Descoberta de Conhecimento em Base de Dados (ou, em inglês, *Knowledge Discovery in Databases Process – KDD-process*) [9]. O KDD é um processo não trivial para identificar a partir de bases de dados padrões válidos, novos, potencialmente úteis e compreensíveis [10]. Esses padrões podem ajudar em processos de tomada de decisão, tais como, definição de catálogo de produtos (ou serviços) e campanhas de marketing [11], [12] e, conforme [8], [13], [14], na definição de práticas de desenvolvimento e de gestão de novos produtos e de projetos de software.

Nesse contexto, o objetivo deste trabalho é identificar modelos e padrões que sejam válidos e potencialmente interpretáveis de tal forma que possam auxiliar as equipes de software na tomada de decisões inerente à realização de iterações de desenvolvimento de software. Esse objetivo é norteado pelas seguintes questões, dadas as informações históricas dessas iterações de software: "é possível prever o resultado de uma iteração de software?" e "esses dados podem revelar algum padrão relacionado às práticas das equipes de software?". Pretende-se alcançar tal objetivo por meio da aplicação de um método de estudo baseado no processo KDD. Esse método utiliza dados reais de desenvolvimento e as seguintes técnicas de mineração de dados: Árvore de Decisão e Floresta Aleatória.

O documento está estruturado em cinco capítulos, a saber. O atual capítulo apresenta o tema e objetivo deste trabalho e cita o método de estudo. O Capítulo II traz um resumo de alguns trabalhos relacionados com o objetivo e o tema deste estudo, descreve processos e métodos de análise e transformação de dados, apresenta tecnologias associadas com base de dados relacionais, descreve duas técnicas utilizadas para mineração de dados, apresenta a linguagem de programação e a ferramenta utilizada neste estudo, descreve processo de software em particular o desenvolvimento iterativo e incremental. O Capítulo III detalha o método de estudo adotado neste trabalho e os dados utilizados para sua execução. O Capítulo IV apresenta os resultados obtidos com a aplicação do método descrito no capítulo anterior. Por fim, no Capítulo V são realizadas as considerações finais.

II. FUNDAMENTAÇÃO TEÓRICA

Os seguintes assuntos serão abordados neste capítulo: trabalhos relacionados com este estudo, processo de análise de dados (*KDD-process*) [9]–[11], transformação de dados textuais utilizando o método *Bag of Words* [15], [16], seleção de atributos a partir do método *Correlation-based Feature Subset Selection* [17], [18], técnicas e algoritmos de mineração de dados (Árvore de Decisão /J48 e Floresta Aleatória/*RandomForest*) [10]–[13], [15], [19]–[21], métricas utilizadas na avaliação dos modelos preditivos [5], [17], [22], [23], tecnologias que podem facilitar a mineração de dados (Python/Spyder e Weka) [12], [16], [17], [22], [24]–[35], processo de software (desenvolvimento iterativo e incremental de software) [1]–[4], [13], [36]–[44] e, por fim, tecnologias que viabilizam o trabalho com base de dados relacionais (MySQL, SQL e MySQL Workbench) [19]–[22].

A. Trabalhos relacionados

A Descoberta de Conhecimento em Base de Dados e Mineração de Dados surgiram há alguns anos ou décadas atrás e estão em constante evolução desde então. Elas são aplicadas em diferentes áreas, tais como, científica, comercial, lazer, industrial, entre outras com finalidades ou objetivos distintos [13], [49]. Nos parágrafos a seguir a descrição resumida de alguns trabalhos relacionados com objetivo e/ou tema deste estudo.

O trabalho realizado por [50] apresenta modelos preditivos capazes de auxiliar gerentes de projeto e/ou tomadores de decisão na identificação de riscos relevantes em um projeto de software. Os modelos conseguem prever a probabilidade de um dado problema representar ou não um risco para o projeto. Os autores utilizaram dados históricos de cinco projetos de código aberto para compor os conjuntos de dados, o modelo de regressão logística esparso para selecionar os fatores de risco e cinco técnicas para construção dos modelos preditivos. Dentre essas técnicas que foram aplicadas ao conjunto de dados para construção de modelos estão a Árvore de Decisão e a Floresta Aleatória, em que o desempenho da segunda é melhor que o dos outros quatro na tarefa de predição dos riscos relevantes aos projetos.

Beltrão [8] em seu estudo apresenta um exemplo de uso de análise de dados de desenvolvimento de software para apoiar gerentes ou engenheiros de software no planejamento de sprint (ou iteração) e na atividade de priorização de atividades. Para isso, o autor aplicou técnicas, tais como, Teoria de Grafos (centralidade) e Algoritmo de Ranqueamento de Páginas com intuito de relacionar *issues,* planejamento de sprint e entregas do Portal do Software Público Brasileiro. A coleta e a transformação das *issues* (dados) em grafos e a aplicação do ranqueamento são realizadas por meio da linguagem de programação Python.

O trabalho desenvolvido por [14] apresenta um modelo de extração de conhecimento em repositório de código fonte baseado no processo de Descoberta de Conhecimento em Base de Dados (ou, em inglês, *Knowledge Discovery in Database* – KDD). O modelo usa Regras de Associação – mais especificamente o algoritmo *Apriori* – para minerar e extrair padrões a partir de métricas do código fonte de projetos de software. O autor valida e exemplifica a aplicação desse modelo desenvolvendo um sistema web que faz uso de dados de dez projetos presentes na *Apache Software Foundation*[1], e na aplicação o modelo em questão é capaz de extrair regras, tais como, "se o *commit*[2] é do tipo *bug*, então a complexidade ciclomática[3] do código aumenta".

Nascimento [51] desenvolve um sistema web de gestão de projetos de software que incorpora tarefas de Mineração de Dados e de Descoberta de Conhecimento em suas funcionalidades, as básicas são baseadas em ferramentas já existentes no mercado. No sistema web há uma funcionalidade que recomenda ao gerente de projeto ações tratativas relacionadas à priorização e ao prazo do projeto. A ferramenta usa os dados históricos dos próprios projetos e a técnica de Árvore de Decisão para predizer a ação a ser realizada pelo gerente do projeto. A solução é validada através de dados de dois projetos fictícios simulando um projeto que aciona a funcionalidade de recomendação e outro que não aciona tal recurso, e o resultado é que o desempenho do primeiro normaliza com mais antecedência que o segundo.

O trabalho realizado por [5] apresenta uma abordagem capaz de prever a capacidade de entrega da equipe em uma

1 Organização sem fins lucrativos criada para suportar os projetos de código aberto, tal como, o do servidor web Apache HTTP, mais detalhes em https://www.apache.org/foundation/.

2 É um comando do sistema de controle de versão que envia as modificações realizadas na área de trabalho do desenvolvedor ao repositório de código fonte do programa/projeto. Para mais detalhes, ver https://git-scm.com/docs/git-commit.

3 Complexidade ciclomática é, conforme [1], "...uma métrica de software que fornece uma medida quantitativa da complexidade lógica de um programa".

iteração de desenvolvimento de software. A previsão da capacidade é através da velocidade da equipe baseada nas informações da iteração (por exemplo, duração, tamanho da equipe) e nos itens de trabalho associados a ela. A velocidade é soma dos pontos dos itens de trabalho da iteração, porém o valor previsto pela abordagem é a diferença (em pontos) da velocidade comprometida junto ao cliente e da velocidade alcançada. Os autores utilizaram as seguintes técnicas e estratégias para transformar essas informações em um conjunto de características que melhor representa uma iteração de desenvolvimento de software: dados estatísticos, *Bag of Words* e medidas de dependência e complexidade obtidas a partir de Grafos Acíclicos Dirigidos. Os dados históricos de cinco grandes projetos de código aberto são utilizados na composição de conjuntos de dados distintos, nos quais são aplicados três classificadores para construção dos modelos preditivos da abordagem. Inclusive, os autores adaptaram a saída dos modelos para predizer o resultado da iteração em si – "meta atingida", "abaixo da meta" e "acima da meta", as três classes em função da diferença de velocidade, por exemplo, a primeira classe significa resumidamente diferença igual a 0. Os modelos demonstram um ótimo desempenho preditivo perante os dados dos cinco projetos avaliados, principalmente o resultante da aplicação da técnica Floresta Aleatória.

Bianchi e Amaral [13] realizam uma Revisão Bibliográfica Sistemática (RBS) para identificar e analisar estudos que usam mineração de dados por meio de Regras de Associação para auxiliar o gerenciamento de projetos que são baseados em modelos híbridos de gestão. Resumidamente, um modelo híbrido é um processo sistemático de gestão que combina princípios, práticas, técnicas e ferramentas de diferentes abordagens de gestão considerando o contexto de negócio e o tipo específico de projeto, buscando maximizar o desempenho do projeto. Os autores constatam o uso de técnicas de mineração de dados que podem aperfeiçoar ou auxiliar atividades de gestão de projetos, por exemplo, alocação de recursos ou formação de equipes, estimativas, definição de políticas de gestão e até definição de padrões (critérios/fatores) que possam orientar o sucesso – ou o fracasso – dos projetos. O aperfeiçoamento ou auxílio é realizado por meio de Regras de Associação, Árvore de Decisão e Redes Neurais.

B. Análise de Dados: KDD-process, Bag of Words e CFS

Há um aumento na quantidade e na variedade de dados nas empresas nos últimos anos, isso em decorrência de diferentes fatores, tais como: a crescente popularização e uso da Internet, o surgimento de (novas) tecnologias de conectividade – por exemplo, Internet das Coisas (IoT), Redes Sociais e Automação residencial e outras –, a diversidade de fontes de dados internas e externas às empresas e hardware cada vez mais potentes e com maior capacidade armazenamento. Esse aumento combinado com a necessidade de uso eficiente e eficaz dos dados em processos de tomada de decisão elevou o interesse em métodos para extração de informações, de padrões e/ou de conhecimento com intuito em geral de dar uma resposta rápida e flexível a mudanças e a oportunidades do mercado [6], [7], [12], [52].

Existem diversos métodos ou processos utilizados no mercado e na academia, tais como, CRISP-DM, SEMMA, DAL, SAS-ALC e KDD-Process. Eles, no geral, possuem etapas equivalentes e a mesma finalidade, que é auxiliar na definição do ciclo de vida dos projetos de análise de dados. Ressalta-se o KDD-Process por ser um dos mais antigos e por estar entre os mais populares na área de análise de dados [9].

O KDD é um processo não trivial para identificar a partir de bases de dados padrões válidos, novos, potencialmente úteis e compreensíveis [10]. Esse processo é não trivial porque é iterativo e interativo: iterativo porque demanda a execução sequencial de várias etapas em que o resultado de uma etapa depende da outra – e essa sequência (ou parte dela) pode se repetir inúmeras vezes, os *loops* – e é interativo porque envolve a participação (ou intervenção) humana durante o processo em si [9], [10].

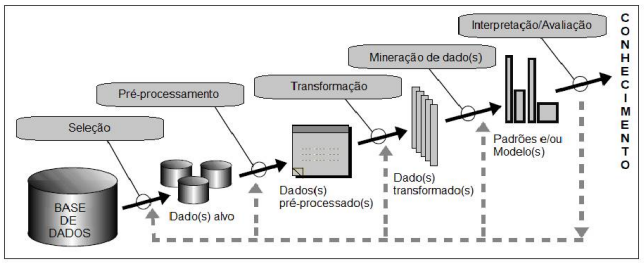


Figura 1. Processo KDD. Adaptado de [10]

Na Figura 1, o fluxo básico das etapas do processo de KDD é representado pelas setas contínuas e a possibilidade de repetição (ou *loop*) de etapas são representadas pelas setas tracejadas. De acordo com [10], as etapas do Processo KDD são Seleção, Pré-processamento, Transformação, Mineração de dados (ou *data mining*) e Interpretação/Avaliação – a seguir a descrição de cada uma delas.

A etapa de Seleção é a primeira etapa do processo. Nela deve ser definido o objetivo ou meta do processo de KDD na perspectiva do cliente, desenvolvido o entendimento do contexto e, por fim, selecionado o conjunto de dados alvo. A seleção envolve escolher o conjunto de dados e as respectivas variáveis (ou atributos) relacionadas ao problema, ou focar em uma amostra de dados ou subconjunto de variáveis no qual o processo de análise de dados será realizado [10].

A segunda etapa é de Pré-processamento. Nessa etapa os dados necessários para a mineração de dados são coletados a partir de diferentes fontes de dados, tais como, planilhas eletrônicas, Bases de Dados e sites na Internet [11]. Os dados passam, então, por um processo de avaliação e manutenção da qualidade – que pode envolver identificação, limpeza, correção e remoção de dados inconsistentes e incompletos ou valores extremos (ou *outliers*) [9], [10]. Pois, conforme [53], essa manutenção da qualidade dos dados é tão importante quanto a escolha de quais técnicas e/ou algoritmos aplicar no conjunto de dados realizada na etapa de mineração de dados.

A Transformação é a terceira etapa do processo de KDD. É nela que os dados pré-processados passam por operações, tais como, seleção e/ou criação de características, agregação e redução ou sintetização de dados. Essas operações buscam criar um conjunto de dados organizado de tal forma que ao

final do processo estará pronto para a etapa seguinte: a mineração de dados [9]. Pode-se enxergar esse conjunto de dados como uma estrutura formada por dois elementos, a saber: a lista ou vetor de características e os dados. O primeiro elemento define quais características (ou variáveis) do problema estão presentes no conjunto de dados – normalmente referenciado também como lista de atributos ou vetor de características; o segundo elemento é o conjunto de linhas ou instâncias do problema (ou ainda, tuplas), em que cada uma dela representa um objeto, observação, transação ou registro com os valores das respectivas características [15]. Os arquivos de entrada de (conjunto de) dados no formato ARFF (ou, em inglês, Attribute-Relation File Format – ou Arquivo no Formato Atributo Relação) da ferramenta WEKA são exemplos típicos de conjuntos de dados organizados para mineração de dados. É fácil identificar neles as duas partes citadas anteriormente, lista de características ou atributos (cabeçalho) e os dados em si – linhas ou instâncias com os valores [17].

Na prática, o domínio de cada característica (ou atributo) deve ser analisado e, se necessário, adequado conforme o objetivo do processo de análise de dados e a técnica de mineração de dados selecionada para tal processo. Pois, há técnicas e/ou algoritmos que aceitam apenas valores numéricos (0, 1, 2,5, 0,35 etc) e/ou valores categóricos ou nominais (sim, não, adulto, idoso, curitiba etc). Por exemplo, as implementações (ou algoritmos) das técnicas de Árvores de Decisão e Floresta Aleatória aceitam tanto valores numéricos quanto categóricos, porém as implementações de Regras de Associação não aceitam valores numéricos contínuos, e as três não aceitam dados textuais [17], [19], [20], [54]. É necessário, então, transformar esses dados. De acordo com [15], [16], para transformar dados textuais pode-se empregar o método *Bag of Words*, mais detalhes a seguir.

O *Bag of Words* é, de acordo com [15], [16], um método simples e amplamente utilizado para transformar um texto em um vetor de números de ocorrência de palavra ou palavras no texto. Ele é um dos métodos de Processamento de Linguagem Natural (PLN) que tipicamente envolve *tokenização*, transformação, contagem das ocorrências, definição do dicionário e vetorização – a seguir a descrição de cada etapa.

- *Tokenização*: o texto é dividido em pedaços menores, em que a forma mais comum é palavras ou *unigrams*, mas é possível trabalhar com grupos de palavras ou *n-grams*. Expandir para dois ou três grupos de palavras pode ajudar no resultado do modelo, porém aumenta consideravelmente o número de características criadas podendo gerar vetores esparsos, algo que demanda cuidado ainda maior na escolha da técnica e/ou algoritmo de mineração de dados. Os vetores esparsos são estruturas com vários elementos e valores iguais que normalmente consumem muito recurso computacional (memória e processamento) durante o processo de aprendizado do modelo.
- Transformação: dependendo do problema de mineração de dados ela pode envolver operações, tais como, (i) converter todas as palavras para minúsculas (*lowercase*), (ii) extrair e usar apenas os sufixos das palavras, por exemplo, trabalha-se apenas com sufixo das palavras "salto", "saltando", "saltou" e não com todas elas – estratégia chamada de *stemming*, (iii) tratar ou remover números, pontuação, caracteres de especiais e *stop words* – são palavras definidas como palavras muito comuns em uma determinada linguagem, em português pode-se citar "de", "para", "que", "muito" e outras. Hoje algumas linguagens de programação trazem em suas bibliotecas de PLN uma lista de *stop words* para diferentes linguagens, o Python é uma dessas linguagens de programação.
- Contagem das palavras e definição do dicionário: após a *tokenização* e as transformações necessárias, é realizado a contagem das palavras gerando uma lista ou conjunto de palavras presentes no texto mais o número de vezes que aquela palavra aparece no texto – o dicionário (ou *bag of words*) do texto. Por exemplo, dado o texto "definição texto palavras dividido texto palavras transformado" o seu dicionário é "definicao=1, texto=2, palavras=2, dividido=1, transformado=1". Em geral, até este ponto e para construir o dicionário trabalha-se com um "corpo de texto" formado pelo conteúdo de uma determinada característica ou atributo de todas as instâncias do conjunto de dados, contudo na maioria das vezes não são todas as palavras do dicionário que se transformam em (novas) características, normalmente as *N-primeiras* palavras com mais ocorrências.
- Vetorização: uma vez definido o dicionário de uma característica com dados textuais, é nesta etapa que o texto da característica de cada instância do conjunto de dados é convertido para um vetor com o número de ocorrências das palavras correspondentes as palavras do dicionário. Ressalta-se que é este vetor que é incorporado ao conjunto de dados como (novas) características extraídas a partir uma ou mais característica com dados textuais. Agora, os respectivos valores dessas características são aceitos pelas técnicas e/ou algoritmos de mineração de dados.

Um breve exemplo das últimas etapas de aplicação do método *BoW*, a saber. Há uma determinada característica em um conjunto de dados com duas instâncias apenas que é "descrição do evento" e o conteúdo dela para essas duas instâncias é o seguinte: conteúdo da "descrição do evento" da instância 1 igual a "definição de texto e de palavras dividido" e conteúdo da "descrição do evento" da instância 2 igual a "texto transformado de palavras". Conforme utilizado como exemplo na etapa anterior, o "corpo de texto" utilizado na contagem e na definição do dicionário foi "definição texto palavras dividido texto palavras transformado" e o dicionário é "definicao=?, texto=?, palavras=?, dividido=?, transformado=?" (neste momento do processamento pode-se considerar como o vetor das características criadas a partir da caracteristica "descrição do evento", ou seja, o *BoW* desse atributo). Então, o texto de cada instância é convertido em número de ocorrências das palavras conforme o BoW da caracteristica "descrição do evento". O resultado é o seguinte: vetorização da "descrição do evento" da instância 1

é "definicao=1, texto=1, palavras=1, dividido=1, transformado=0" e a vetorização da "descrição do evento" da instância 2 é "definicao=0, texto=1, palavras=1, dividido=0, transformado=1".

Conforme descrito anteriormente, a aplicação do método de transformação *Bag of Words* modela dados textuais em um vetor numérico de características, o qual é incorporado ao conjunto de dados. Na prática, com ou sem dados textuais presentes no conjunto de dados, inicialmente a quantidade de características envolvidas em um problema de análise de dados pode ser grande ou começar com um número pequeno e crescer consideravelmente. Deve-se, então, selecionar quais características ou atributos são relevantes ou pertinentes ao problema e ao modelo em si. O esforço de seleção ocorre principalmente nas etapas iniciais do processo de KDD, mas pode ocorrer também nesta etapa, na de Transformação. E, independentemente do momento no processo e ao contrário do que se possa imaginar, o importante não é a quantidade de características ou atributos e sim o quanto elas impactam positivamente no desempenho do modelo [10], [12].

A melhor seleção de características ou atributos é a seleção manual realizada pelo responsável pelo processo de análise de dados a partir de um entendimento mais profundo do problema e de seu contexto. No entanto, métodos automáticos podem ser úteis principalmente perante um grande número de variáveis. A redução da quantidade de características (ou redução da dimensionalidade) do conjunto de dados permite que os interessados no processo de análise de dados foquem a atenção nos atributos relevantes ou importantes. Ressalta-se que mesmo em técnicas ou algoritmos que possuem dentro deles a seleção de atributos (ou seleção *embeded*), tal como, Árvore de Decisão o desempenho de classificação pode sofrer uma redução de 5% a 10% dada a presença (ou inclusão) de atributos não relevantes ao modelo [12].

Além da abordagem *embeded* citada anteriormente, em que a seleção de atributos faz parte do processo de indução da técnica ou algoritmo de aprendizado de máquina, há outras duas abordagens a *wrapper* e a por filtro. Na abordagem *wrapper* a seleção dos atributos é realizada a partir do resultado da aplicação de uma técnica ou algoritmo de aprendizado de máquina no conjunto de dados. Ou seja, o processo de seleção fica mais custoso por depender da execução da técnica ou algoritmo – por isso ela é normalmente utilizada quando há uma quantidade pequena de atributos. Diferente das abordagens anteriores, a abordagem por filtro não envolve uma técnica ou algoritmo de aprendizado de máquina. Ela analisa informações gerais dos dados para selecionar um (sub)conjunto de atributos. A abordagem por filtro utiliza métricas, tal como, correlação para realizar tal análise [55]. Essa última abordagem de seleção de atributos é mais simples e rápida e traz resultados mais generalizados que as outras duas, por isso a abordagem por filtro é mais utilizada tanto na indústria quanto em pesquisas acadêmicas [56].

O método CFS (ou, em inglês, *Correlation-based Feature Subset Selection*) é um dos métodos de seleção de atributos por filtro que é baseado na relevância dos atributos e na (não) redundância entre eles. A relevância de um atributo é obtida a partir da correlação entre ele e o resultado (classe) do modelo. A (não) redundância ocorre no momento que atributos com relevância menor não são adicionados ao subconjunto de atributos. Ou seja, ao final da busca há um subconjunto reduzido de atributos mais relevantes para o desempenho preditivo do modelo. A busca pode começar com o subconjunto vazio (forward) ou completo (backward) ou em qualquer ponto (bidirectional) e prosseguir [18]. Uma implementação desse método de seleção e busca de atributos pode ser encontrada, por exemplo, de acordo com [17], na ferramenta WEKA na seção *Select attributes* escolhendo o avaliador *CfsSubsetEval*.

A etapa de Mineração de Dados (ou *Data Mining*) é a quarta etapa do *KDD-process*. Nela tarefas de mineração de dados são realizadas a partir da aplicação de técnicas e/ou algoritmos de mineração de dados escolhidos conforme o objetivo/meta do processo de análise de dados e tipos de dados presentes no conjunto de dados. O objetivo da Mineração de Dados é construir modelos para verificar uma hipótese e/ou descobrir novos padrões; e dependendo do que se está investigando e dos resultados obtidos durante o processo, pode-se realizar uma análise exploratória selecionando diferentes técnicas e algoritmos de mineração ou variando seus parâmetros ou conjecturando novas hipóteses [9], [10]. As tarefas de mineração de dados podem ser divididas em dois grupos principais, a saber: preditivas e descritivas. As tarefas do primeiro grupo buscam criar modelos capazes de prever um resultado para dados categorizados que não estavam no conjunto de dados treinamento – classificação e regressão são exemplos específicos e típicos de tarefas desse grupo. Agora, nas tarefas do segundo grupo busca-se explorar e/ou descobrir padrões (ocultos) em dados não categorizados ou "rotulados" – Agrupamento e Regras de associação são exemplos específicos e típicos de tarefas desse grupo [21], [25].

A realização das tarefas preditivas de mineração de dados envolve, de acordo com [15], os seguintes passos: (1) preparar o conjunto de dados conforme a técnica escolhida, características do problema e tipos de dados envolvidos; (2) separa aleatoriamente o conjunto de dados em dois, dados de treinamento e dados de teste; (3) construir (ou aprender) o modelo preditivo usando o conjunto de dados de treinamento; (4) aplicar o modelo no conjunto de dados de teste; e, por fim, (5) avaliar o desempenho preditivo do modelo. Agora, na realização das tarefas descritivas de mineração de dados os passos podem ser resumidos em (1) preparação do conjunto de dados, (2) aplicação da técnica/algoritmo no conjunto de dados e (3) avaliação ou interpretação dos eventuais padrões ou informações extraídos dos dados.

Os modelos gerados pelas tarefas preditivas de mineração de dados devem ser validados. A validação do modelo indicado no parágrafo anterior é o *holdout* – ver passos de dois a quatro. Nela, segundo [22], [25], o conjunto de dados além de ser dividido aleatoriamente em dois – dados de treino e dados de teste –, é recomendado que isso ocorra diversas vezes e que o resultado seja avaliado por meio da média de diferentes métricas de desempenho, tais como, erro absoluto médio absoluto e acurácia (ou porcentagem de instâncias classificadas corretamente) e área

sob a curva ROC (ou, em inglês, *Receiver Operating Characteristic*).

De outro modo, a validação mais indicada para tarefas preditivas de classificação é o método k-partições (ou *k-folds* ou ainda *cross-validation folds* na ferramenta WEKA). Ele é semelhante ao método anterior, pois, há a divisão do conjunto de dados em treinamento e teste mas nele a quantidade de divisões é diferente, a saber: (1) o conjunto é inicialmente divido aleatoriamente em k subconjuntos disjuntos ou partições (ou folds) – pode-se usar 5, 10 e 20, mas o valor 10 é recomendado para estimar a acurácia dado o seu viés e variância relativamente baixo; (2) separa-se uma partição (ou subconjunto) e é treinado um modelo com todos os dados exceto os dados da partição separada; (3) aplica-se o modelo gerado nos dados da partição separada e guarda o resultado ou predições; os passos 2 e 3 se repetem k vezes e, só ao final desse processo e a partir da agregação das métricas de desempenho, que (4) o resultado da tarefa preditiva é avaliada e/ou interpretada, considerando a média das métricas de desempenho dos modelos gerados [11], [12], [15]

A quinta e última etapa do *KDD-process* é a de Interpretação e Avaliação. Nessa etapa os modelos inferidos/ gerados e/ou eventuais padrões descobertos pelas técnicas de mineração de dados são visualizados, avaliados e interpretados [10], [15]. A avaliação e/ou interpretação ocorre normalmente por meio de métricas. Elas, que podem ser subjetivas e objetivas, devem ser previamente selecionadas ou definidas considerando a técnica de mineração escolhido, tipos de dados manipulados e o objetivo do projeto ou processo de análise de dados e terem seus valores obtidos neste momento do processo [10], [22], [54].

Ressalta-se que algumas dessas métricas podem auxiliar na identificação de dois problemas comuns quando se trabalha com modelos preditivos, são eles: *overfiting* e *underfiting*. Um modelo com *overfiting* trabalha muito bem com o conjunto de dados de treinamento, mas não com um conjunto novo de dados. Por outro lado, no *underfiting* o desempenho do modelo fica aquém do esperado com o próprio conjunto de dado de treinamento, isso normalmente sugere que o algoritmo escolhido não é adequado para o conjunto de dados – ou vice-versa. Destaca-se que a validação descrita em parágrafos anteriores é extensivamente utilizada para evitar tanto *overfiting* quanto *underfiting* [25].

Enfim, uma vez descoberto o conhecimento, segundo [10], [15], deve-se agir sobre ele: usá-lo diretamente, incorporá-lo em algum sistema para estender sua ação, confrontar com conhecimentos (ou crenças) preexistentes e/ou simplesmente documentá-lo para comunicar as partes interessadas.

C. Mineração de Dados: Árvore de Decisão e Floresta Aleatória

A Mineração de Dados (ou, em inglês, *Data Mining* – DM) é vista, conforme [10], como um componente do processo de Descoberta de Conhecimento em Base de Dados (ou KDD). Por outro lado, de acordo com [11], ela é para muitas pessoas um termo sinônimo, por exemplo, de "mineração de conhecimento a partir de dados", "extração de conhecimento" e até mesmo do processo de KDD.

O objetivo, todavia, tanto da Mineração de Dados vista como processo ou parte dele é extrair conhecimento a partir de um grande volume de dados para verificação de uma hipótese definida pelo usuário e/ou para descobrir (novos) padrões. A natureza do processo de extração de conhecimento é interdisciplinar, ou seja, ela pode envolver áreas, tais como, Estatística, Aprendizado de Máquina, Banco de Dados e *Data Warehouse*, entre outras [10], [11].

O Aprendizado de Máquina (ou, em inglês, *Machine Learning*) investiga como programas de computador podem aprender a tomar decisões se baseando em dados. Por exemplo, um programa de computador que reconhece dígitos escritos à mão após aprender a partir de um conjunto de exemplos de dígitos [11], [53]. Há quatro tipos de aprendizado de máquina, no entanto os principais são: Aprendizado Supervisionado e Aprendizado Não supervisionado [11], [15].

O Aprendizado Supervisionado é bem semelhante – ou normalmente é associado – a tarefa preditiva de mineração de dados. Esse tipo aprendizado envolve a aplicação de um algoritmo em um conjunto de dados de treinamento gerando um modelo preditivo que faz o mapeamento entre características ou atributos de entrada e característica ou atributo de saída – a classe ou atributo alvo. Uma vez inferido ou treinado o modelo, ele é capaz de prever a saída a partir de novos valores de entrada. A supervisão é consequência dos dados estarem previamente categorizados ou "rotulados" – o valor/classe de cada instância do conjunto [11], [53]. Diferentemente do tipo anterior, no Aprendizado Não Supervisionado se trabalha com dados não categorizados ou "rotulados", pois, o objetivo desse segundo tipo de aprendizado é aplicar algoritmos capazes de categorizar os dados e/ou encontrar padrões "ocultos" no conjunto. Esse segundo tipo é semelhante – ou normalmente é associado – a tarefa descritiva de mineração de dados [11], [15].

De acordo com [12], [21], há várias técnicas que podem ser utilizadas em cada um dos tipos de aprendizados descritos anteriormente, por exemplo, Regressão Linear, Redes Neurais Artificiais, Árvore de Decisão e Floresta Aleatória – no aprendizado supervisionado – e Agrupamento e Regras de Associação – no aprendizado não supervisionado. Porém, dado o foco deste trabalho, são descritas a seguir as seguintes técnicas: Árvore de Decisão e Floresta Aleatória.

A Árvore de Decisão é amplamente utilizada em Aprendizado de Máquina devido aos seguintes fatores: suporta diferentes tipos de dados (categórico e numérico) e, comparada com outras técnicas, o seu tempo de treinamento é menor e a representação do conhecimento (o modelo gerado) é mais facilmente interpretado por humanos. Essa técnica pode ser aplicada em tarefas de classificação e de regressão – o tipo depende basicamente da saída do modelo, conforme descrito anteriormente. Por fim, essa técnica tem sido utilizada para resolver problemas em diversas áreas, por exemplo, diagnóstico médico, análise de crédito e projeto de software [12], [13], [15].

A estrutura do modelo preditivo gerado pela técnica citada anteriormente é constituída por um conjunto hierárquico de elementos em formato de árvore. Os elementos são nós interligados por arcos – ou chamados também de ramos. Há basicamente dois tipos de nós, os internos e os terminais (ou nó folha). Os nós internos representam características (ou atributos) do conjunto de dados e estão associados a entrada do modelo e os arcos associados a esse tipo de nó representam os valores dessas características nas instâncias [21]. Por exemplo, a figura a seguir a apresenta uma árvore de decisão relacionada com a compra (ou não) de pacote de viagem dentro de uma agência de turismo. Os nós internos são A1, A2 e A3. O nó folha é Comprar (ou Não Comprar). Os arcos (ou valores) do primeiro nó interno são 1, 2, 3 e 4, do segundo nó são Fluente (F) e Não fluente (NF) e, por fim, do terceiro nó são Internacional e Local.

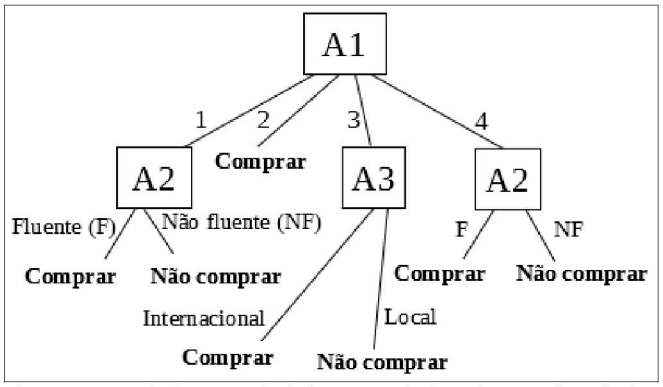


Figura 2. Exemplo de árvore de decisão (agência de turismo). Adaptado de [21, p. 393]

Para verificar uma (nova) instância a estrutura é percorrida por por meio de uma sequência de testes realizados nas características (nós internos) em função de seus valores (arcos) até chegar na decisão do modelo ou saída – um nó terminal, o qual representa também uma característica do conjunto de dados, a classe ou valor predito. Essa sequência de testes é realizada de cima para baixo, do nó raiz até a extremidade da árvore – nós terminais ou folhas [21].

A lógica de construção da Árvore de Decisão se baseia na estratégia recursiva de *dividir-e-conquistar* e no conjunto de dados ou exemplos (conjunto de dados de treinamento). Veja o algoritmo de aprendizagem em árvore de decisão apresentada a seguir – ela foi extraída de [53, p. 815].

```
função APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO(exemplos, atributos, exemplos-pais)
retorna uma árvore de decisão
  se exemplos é vazio então retornar VALOR-DA-MAIORIA (exemplos_pais)
  senão se todos os exemplos têm a mesma classificação então retornar a classificação
  senão se atributos é vazio então retornar VALOR-DA-MAIORIA(exemplos)
  senão
    A ← argmax_{a ∈ atributos} IMPORTÂNCIA (a, exemplos)
    árvore ← uma nova árvore de decisão com teste de raiz A
    para cada valor v_k de A faça
      exs ← {e : e ∈ exemplos e e.A = v_k}
      subárvore ← APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO (exs, atributos — A, exemplos)
      adicionar uma ramificação à árvore com rótulo (A = v_k) e subárvore subárvore
    retornar árvore
```

Figura 3. Algoritmo de aprendizagem em árvore de decisão. Extraído de [53, p. 815].

A seleção da característica mais significativa ou discriminatória – finalidade das funções IMPORTÂNCIA e VALOR-DA-MAIORIA presentes no algoritmo apresentado anteriormente – envolve, na prática, verificar qual característica que uma vez testada dividirá melhor o (sub)conjunto de dados (ou instâncias), e fazendo isso é incorporada como um nó da árvore. Um dos critérios mais utilizados é o Ganho de Informação, quanto maior melhor – veja a seguir o cálculo do ganho de informação dado a participação do conjunto a partir de um atributo A com v valores distintos ($a_1, a_2, \ldots a_v$) [11].

$$Ganho(A) = Info(D) - Info_A(D),$$

$$Info(D) = -\sum_{i=1}^{m} p_i \cdot \log_2(p_i),$$

$$Info_A(D) = \sum_{j=1}^{v} |D_j|/|D| \cdot Info(D_j),$$

onde D é um conjunto de dados de treinamento de instâncias ou tuplas rotuladas por classe, m os valores distintos das classes, ou seja, C_i (para $i = 1, \ldots, m$). Então, $C_i D$ é o conjunto de instâncias da classe C_i em D. E mais, $|D|$ e $|C_i D|$ denota o número de instâncias em D e $C_i D$, respectivamente. Logo, p_i é estimado por $|C_i D| \ / \ |D|$. $Info(D)$ é também conhecida como entropia de D. Por fim, D_j contém as instâncias de D que têm o resultado a_j de A.

Ela é utilizado no algoritmo chamado ID3. Essa medida nos dá a característica que reduz a Entropia do (sub)conjunto de dados quando ele for dividido. A Entropia é uma medida da falta de homogeneidade dos dados. Há, entretanto, um viés na medida de Ganho de Informação quando as características possuem muitos valores, por exemplo, uma característica em que praticamente identifica cada instância de maneira única. Em situação assim, a divisão do (sub)conjunto tenderia para uma divisão ineficiente ou indesejada. Então, algoritmos de Árvore de Decisão sucessores ao ID3, tal como o C4.5, estendem a medida anterior usando a Razão de Ganho, quando o viés se faz presente, caso contrário ele usa somente o Ganho de Informação – veja a seguir o cálculo da razão de ganho [11].

$$Razão\,de\,Ganho(A) = Ganho(A) / DivInfo_A(D),$$

$$DivInfo_A(D) = -\sum_{j=1}^{v} |D_j|/|D| \cdot \log_2(|D_j|/|D|),$$

onde a razão de ganho aplica um tipo de normalização ao ganho de informação usando um valor de "divisão de informação", ou $DivInfo$, definido analogamente com $Info(D)$

Uma Árvore de Decisão pode crescer muito de tamanho, se tornando de difícil compreensão e potencialmente gerando *overfitting*. Então, ela deve ser "podada" mas de tal forma que não haja perda no desempenho do modelo resultante. As técnicas de poda podem ser divididas em duas, dependendo do momento que a poda ocorre durante a construção da árvore, a saber: pré-poda ou pós-poda. Na primeira, o crescimento da árvore é interrompido quando é determinado que nenhuma característica aumentará significativamente o ganho de informação no processo de divisão dos (sub)conjuntos de dados. Por outro lado, na pós-poda como a árvore já está construída são avaliadas diferentes configurações através da poda de ramos e/ou subárvores, se a estimativa de erro da nova configuração é menor ou igual a estrutura original, então a poda é efetivamente realizada. O algoritmo C4.5 realiza pós-poda, mas se chama Poda Pessimista e ele calcula a estimativa de erro a partir do

conjunto de dados de treinamento, adicionando uma penalidade ao cálculo devido ao fato de se trabalhar com tal conjunto de dados [11], [21].

ID3 (*Iterative Dichotomiser*) e C4.5 são algoritmos de Árvore de Decisão desenvolvidos por John Ross Quinlan entre o final dos anos de 1970 e início dos anos de 1980, em que o segundo é uma evolução do primeiro. O C4.5, desde então, se tornou uma referência com a qual os algoritmos de aprendizado supervisionado mais recentes são frequentemente comparados [11], mais detalhes em [20].

A Floresta Aleatória (ou, em inglês, *Random Forest*) pode ser utilizada tanto para tarefas de classificação quanto para tarefas de regressão, tal como, a técnica descrita anteriormente. O poder de generalização da Floresta Aleatória, entretanto, é maior que a da Árvore de Decisão – inclusive o erro de generalização costuma convergir conforme o tamanho da floresta cresce. O desempenho da Floresta Aleatória, na realidade, é comparável ao de outra técnica popular, o AdaBoost[4] (ou *Adaptive Boosting*), no entanto a Floresta Aleatória lida melhor com erros e valores extremos (ou anomalias ou *outliers*). A Floresta Aleatória é uma técnica *ensemble*[5] de aprendizado supervisionado de máquina, ou seja, o modelo gerado por ela é uma combinação (*bagging*) de vários aprendizes base – "floresta" de Árvores de Decisão. O resultado, então, de cada aprendiz do conjunto se torna um "voto" para uma determinada classe no contexto do problema. Logo, o resultado do modelo é a classe mais popular dentre os aprendizes base ou a média dos valores – no caso das tarefas de regressão [11].

A lógica de construção da "floresta" de árvores de decisão envolve aleatoriedade tanto da seleção dos subconjuntos de dados para treinar cada aprendiz base quanto na construção de cada um deles, a saber: primeiro, é retirado aleatoriamente um subconjunto dos dados de treinamento para treinar uma árvore de decisão. Depois, para cada nó dessa árvore é selecionado aleatoriamente um subconjunto de características e a partir dele definido a característica que melhor divide o subconjunto. Por exemplo, são selecionadas aleatoriamente 10 características de 100 características disponíveis no subconjunto de dados e então usa-se a melhor característica dentre as 10 para dividir o subconjunto de dados. Então, esse processo se repete: é retirado aleatoriamente um segundo subconjunto dos dados de treinamento para treinar outra árvore de decisão. As previsões realizadas por essa segunda árvore são diferentes da primeira. O processo poderia continuar indefinidamente, mas normalmente o limite inicial é de 100 árvores [11], [19].

Diferente da técnica de Árvore de Decisão, na Floresta Aleatória não há poda das árvores de decisão e mesmo assim ela é capaz de lidar bem com um grande volume de dados e/ou quantidade de características (ou variáveis). Outra diferença, na Árvore de Decisão as características mais relevantes estão na própria estrutura do modelo (a árvore em si), são as que estão mais próximo do topo da estrutura; agora, na Floresta Aleatória são as com melhores estimativas internas de importância [11], [15], [19].

Na prática, a aplicação das técnicas de mineração de dados, tais como, Árvore de Decisão e Floresta Aleatória, é viabilizada por implementações (ou programas de computador) normalmente em ferramentas (ou plataformas) para tal finalidade. Por exemplo, o J48 é uma implementação da ferramenta WEKA para o algoritmo C4.5 de indução de árvore de decisão. Nessa implementação, o J48 é capaz de trabalhar com atributos categóricos, numéricos, binários e valores faltantes. Em relação a classe (resultante), ele pode lidar com classes categóricas e binárias e, inclusive, com valores faltantes. E, o *RandomForest* é a implementação da ferramenta em questão para a técnica Floresta Aleatória. A capacidade dessa implementação é semelhante a implementação descrita anteriormente com exceção que esta é capaz de lidar com valor numérico como classe [17], [19], [20].

D. Métricas utilizadas na avaliação dos modelos preditivos

O desempenho dos modelos preditivos de classificação inferidos pelas técnicas de aprendizado supervisionado de máquina pode ser avaliado e interpretado utilizando diferentes métricas [22], porém, neste contexto, baseado em [5], [22], [23], destacam-se as seguintes: Acurácia, Erro Absoluto Médio (ou *Mean Absolute Error* – MAE) e Área sob a Curva ROC (ou *Area under of ROC Curve* – AUROC). Nesse cenário ressalta-se a Matriz de Confusão. Conforme [22], nela são representadas as quantidades de classificações corretas e incorretas realizadas pelo modelo preditivo, veja o esquema na figura a seguir.

		Classificado como (ou predito)	
		Verdadeiro	Falso
Valor real (ou observado)	Verdadeiro	Verdadeiro Positivo (VP)	Falso Negativo (FN)
	Falso	Falso Positivo (FP)	Verdadeiro Negativo (VN)

Figura 4. Exemplo de Matriz de Confusão (duas classes). Figura do autor.

A Acurácia é, conforme [22, p. 18], "...a medida comum entre as métricas. Seu valor define o quão próximo da classificação correta o algoritmo está". Ainda conforme [22, p. 29], a fórmula da acurácia é:

$$\frac{(VP+VN)}{(VP+VN+FP+FN)},$$

onde VP é número de verdadeiros positivos, VN é número de verdadeiros negativos, FP é número de falsos positivos e, por fim, FN é número de falsos negativos. Em outras palavras, é a razão entre o número de classificações realizadas corretamente e o número total de classificações. A ferramenta WEKA, conforme [17], apresenta a acurácia como "% de instâncias classificadas corretamente" pelo modelo (preditivo).

O valor médio dos erros absolutos (MAE) de predição do modelo é calculado da seguinte forma, baseado em [23]:

$$\frac{1}{n}\sum_{i=1}^{n}|f(x_i)-y_i|,$$

onde n é número de previsões, f(xi) é o valor predito e yi é valor observado (ou rotulado). Neste contexto ressalta-se que os valores categóricos precisam ser transformados em valores numéricos para a realização desse cálculo, por

4 De acordo com [11], é um algoritmo de aprendizado de máquina supervisionado que usa a abordagem de *ensemble*, mas envolve atribuição de pesos aos resultados dos aprendizes base utilizando esse peso na avaliação final do modelo – ou *Boosting*.

5 É uma abordagem que combina (*bagging*) uma série de aprendizes base para criar um modelo com acurácia melhor que a gerada pelos aprendizes base insoladamente, mais detalhes em [11] e [15].

exemplo, valor 0 para "categoria A" e valor 1 para "categoria B".

A curva ROC (ou, em inglês, *Receiver Operating Characteristics Curve*) é uma representação bidimensional do desempenho do modelo preditivo, em que a taxa de verdadeiros positivos (ou TVP) está no eixo vertical e a taxa de falsos positivos (ou TFP) está no eixo horizontal – vejam as fórmulas e figura a seguir [22].

$$TVP = \frac{VP}{(VP + FN)},$$

$$TFP = \frac{FP}{(FP + VN)},$$

onde VP é número de verdadeiros positivos, VN é número de verdadeiros negativos, FP é número de falsos positivos e, por fim, FN é número de falsos negativos.

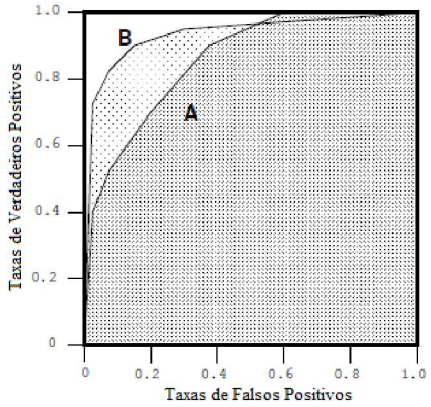


Figura 5. Exemplo Área sob a Curva ROC. Adaptado de [57, p. 15]

A curva é obtida por meio da plotagem dos valores gerados pelo modelo para cada instância [22]. Pode-se, entretanto, comparar o desempenho de modelos preditivos reduzindo o desempenho da ROC a um único valor escalar. Um método comum para isso é calcular a área sob a curva ROC. Esse valor pode variar entre 0 e 1, mas se espera de um modelo preditivo realista um valor maior que 0,5 [57]. A ferramenta WEKA usa a estatística de Mann-Whitney para calcular tal valor, mais detalhes em [17].

As métricas são utilizadas na avaliação do desempenho de um modelo em particular ou na comparação entre modelos, nos quais um deles é o modelo de linha de base. Conforme [4], uma linha de base é um auxílio eficaz na melhoria do processo e/ou estimativas trazendo benefícios nos níveis (técnicos) de processo, projeto e produto. Para [5], [58], [59], o modelo de linha de base pode ser inferido por técnicas mais simples, tal como, *ZeroR*. Espera-se, portanto, que o desempenho dos modelos propostos ou em estudo supere o desempenho dos modelos de linha de base e, pelo menos, se aproxime do desempenho de modelos gerados em estudos semelhantes.

E. Linguagem de Programação e Ferramenta: Python, Spyder e Weka

A busca por possibilitar que profissionais de diferentes áreas do conhecimento possam fazer uso mais facilmente de algoritmos e/ou técnicas de Mineração de Dados e a intensificação das pesquisas no desenvolvimento e/ou evolução desses procedimentos corroboraram com a atual diversidade de ferramentas e de linguagens de programação voltadas para tal atividade [24], [26], [27], [60]. Dentre essa diversidade destacam-se a linguagem Python e a ferramenta WEKA, pois foram os principais recursos utilizados neste estudo.

Python é uma linguagem de programação de propósito geral e de alto nível criada por Guido van Rossum em 1989. Ela é interpretada e suporta diferentes paradigmas de programação por meio de seu paradigma orientado a objeto. O Python trabalha com exceções, tipagem dinâmica, tipos de dados dinâmicos de alto nível e classes. Além de trazer em sua biblioteca padrão recursos que agilizam e/ou facilitam o desenvolvimento de diferentes tipos de programas por parte dos programadores, ele também permite estender os recursos da linguagem por meio da instalação e importação de módulos (ou bibliotecas) desenvolvidos pela comunidade Python em geral para as mais diversas finalidades – por exemplo, construção de sistemas web, aplicativos desktop ou mobile, aprendizado de máquina, processamento de linguagem natural etc [16], [28], [29].

Python é uma das linguagens mais populares para análise de dados devido à variedade de módulos disponíveis para instalação desenvolvidos por desenvolvedores renomados e pela comunidade de código aberto [30]. Perante essa variedade ressaltam-se estes dois módulos: o *Natural Language Toolkit* (ou NLTK) e o *Scikit-learn* (ou apenas *sklearn*) porque ambos foram utilizados na etapa de transformação (de dados) deste trabalho. O NLTK é praticamente a principal plataforma para desenvolver programas Python que precisem trabalhar com dados relacionados com linguagem natural, inclusive oferecendo suporte ao português brasileiro [31]. O *sklearn* é um módulo de código aberto para facilitar o desenvolvimento de programas Python envolvendo tanto o aprendizado de máquina supervisionado quanto o não supervisionado. Ele fornece vários recursos, tais como, ajuste, seleção e avaliação de modelos, pré-processamento de dados e muitos outros [32]. Por exemplo, a classe *CountVectorizer* do módulo *sklearn* e a função *stopwords.words('portuguese')* do módulo *nltk* juntos possibilitam converter uma coleção de textos (português brasileiro) em uma matriz de palavras – ou *tokens* – com a quantidade de ocorrência de cada uma delas – recursos esses utilizados na implementação do método *Bag of Words* descrito anteriormente neste trabalho. Por fim, tanto o primeiro quanto o segundo módulo podem ser instalados em diferentes Sistemas Operacionais (ou SOs) – *MacOS*, *Windows* e variações *Unix* [31], [32].

O Python no geral apresenta uma portabilidade abrangente em termos de SOs – ou seja, um programa desenvolvido nessa linguagem pode ser executado, por exemplo, em computadores com *MacOS*, *Windows* e variações *Unix*. Uma vez instalado o Python é possível desenvolver e executar programas utilizando apenas o seu interpretador e um editor de texto simples, por exemplo, o Notepad++. Pode-se usar, entretanto, tanto um interpretador com mais recursos visuais e de auxílio, tal como, o *ipython* quanto um ambiente de desenvolvimento mais interativo, tal como, *Spyder* [28], [29]. O *Spyder* é um ambiente de desenvolvimento Python de código aberto e livre desenvolvido para analistas de dados, engenheiros e cientistas que permite, por exemplo, edição de código fonte,

depuração de programas, customização do ambiente de trabalho e visualização e exploração de dados. É possível instalar esse ambiente de desenvolvimento nos três SOs citados anteriormente [33]. Para mais detalhes sobre instalação e uso tanto do Python quanto do Spyder ver [28], [29] e [33], respectivamente.

O *Waikato Environment for Knowledge Analysis* (ou WEKA) é uma ferramenta de análise de dados testada, experimentada e de código aberto desenvolvida em 1999 pela Universidade de Waikato da Nova Zelândia. Ela conéem inúmeras implementações de algoritmos de aprendizado de máquina e de mineração de dados, e é amplamente utilizada no ensino, pesquisa e na indústria. É possível instalar essa ferramenta em computadores com *MacOS*, *Windows* e variações *Unix*, e uma vez feito isso é possível interagir com ela por meio de linha de comando, de uma interface gráfica (*WEKA GUI Chooser*) ou de uma *Application Interface Programming* (ou API) Java [22], [24], [34], [35]. Além da funcionalidade de visualização e edição de arquivos ARFF (menu/opção *Tools* → *ArffViewer*), no contexto deste estudo também destacam-se as seguintes funcionalidades disponíveis no *WEKA GUI Chooser*: *Explorer* e *Experimenter* – mais detalhes nos parágrafos a seguir. De acordo com [17], os arquivos ARFF são uma das formas de entrada de (conjunto de) dados na ferramenta. Eles são formados basicamente por três partes: nome do arquivo/conjunto de dados, lista de características (ou atributos) e os dados em si. Por exemplo, na figura a seguir é possível identificar essas três partes por meio dos seguintes metadados @RELATION, @ATTRIBUTE e @DATA. Na prática, é utilizada a funcionalidade citada anteriormente para visualizar tais arquivos – inclusive, se necessário, é possível utiliza-la também para converte-los para o formato CSV.

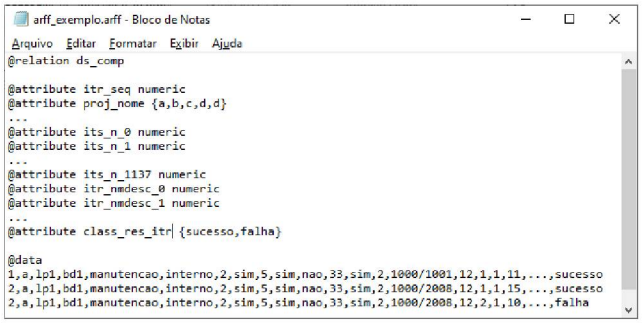


Figura 6. Exemplo de arquivo ARFF. Figura do autor.

Na funcionalidade *Explorer* da interface gráfica do WEKA pode-se (aba/seção na respectiva interface/funciolidade): selecionar os dados e modificá-los – se necessário por meio de *filter* (*Preprocess*); treinar e testar modelos que classificam ou realizam regressão (*Classify*); realizar clusterização[6] dos dados (*Cluster*); descobrir regras de associação (*Associate*); selecionar/identificar as características (ou atributos) mais relevantes no conjunto de dados (*Select attributes*); e, por fim, visualizar em duas dimensões os dados (*Visualize*) [17]. Por outro lado, a funcionalidade *Experimenter* da mesma interface gráfica é um ambiente de realização de experimentos. Ela permite o usuário criar, executar (ou "rodar"), modificar e analisar experimentos de uma maneira mais prática e produtiva que se eles fossem realizados individualmente ou um por vez [17]. Neste estudo essa funcionalidade foi utilizada na etapa de mineração de dados especificamente no treinamento e na etapa de avaliação e interpretação dos modelos preditivos gerados com a aplicação das implementações/algoritmos de Árvore de Decisão e de Floresta Aleatória.

Para mais detalhes sobre instalação e uso do WEKA ver [35] e [17]. Em [12] pode-se encontrar indicação de implementações/algoritmos da ferramenta em questão conforme a etapa e/ou tarefa do processo de análise de dados. Ou, em [22] há uma comparação entre implementações/algoritmos presentes no WEKA. Enfim, para tomar conhecimento de outras ferramentas de análise de dados ver [24] e/ou [60].

F. Processo de Software: Desenvolvimento Iterativo

O software está presente em várias categorias de sistemas, tais como, sites web, aplicativos em celulares, sistemas embutidos em um forno de micro-ondas e ambientes ou ferramentas de desenvolvimento de sistemas. O software é tanto o produto – de um processo de elaboração conduzido por profissionais – quanto o meio utilizado para sua entrega – por exemplo, linguagens de programação e ambientes de desenvolvimento [1]. Ele está cada vez mais incorporado aos aspectos do dia a dia das pessoas e das empresas – as quais dependem dele, de maneira crescente, para controle de tarefas, operações e/ou decisões estratégias e táticas. Os cenários que envolvem software estão se tornando mais complexos a cada período que passa demandando cada vez mais sistemas escaláveis e de fácil adaptação. Nesse cenário, a Engenharia de Software estuda e define processos, métodos e ferramentas que orientam e auxiliam os profissionais de software para o desenvolvimento de sistemas de alta qualidade [4].

Os processos são a base da engenharia de software e do controle gerencial dos projetos envolvendo sistemas, tecnologia da informação e comunicação. Os processos de software estabelecem os marcos do desenvolvimento do produto, o conjunto de ações ou atividades e seus responsáveis (ou papéis) e os artefatos de software gerados – ou produtos de trabalho gerados, tais como, documentos, modelos, estruturas de dados, código etc. A qualidade do software é assegurada e as alterações e riscos são adequadamente geridos tanto por meio das atividades ou tarefas de engenharia, tais como, engenharia de requisitos e projeto (ou design) de software, quanto por meio das atividades de apoio que também devem ocorrer ao longo do processo, tais como, gestão de projeto e da configuração de software [1]. Em outras palavras, um processo de software resumidamente é um conjunto de ações e atividades cujo objetivo é o desenvolvimento e evolução de um software com qualidade, e a representação simplificada desses processos sob uma perspectiva específica é o modelo de processo de software [2].

A maioria dos modelos de processo de software se baseia em um destes três modelos, a saber: Engenharia de Software baseada em Componentes, Modelo em Cascata e Desenvolvimento Iterativo. Os profissionais de software

6 A clusterização (ou agrupamento) se aplica, conforme [12, p. 135], "...quando não há classe a ser predita, mas sim quando as instâncias devem ser divididas em grupos naturais".

estão sujeitos a problemas ou situações no decorrer do seu desenvolvimento e esses modelos são soluções experimentadas para tais problemas [2].

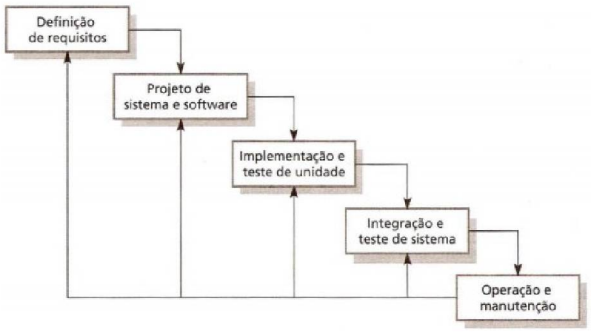


Figura 7. Modelo em Cascata. Extraído de [2, p. 44]

A Figura 7 apresenta o primeiro modelo de processo de software, o Modelo em Cascata. A origem dele foi os processos mais gerais de engenharia de sistemas, a qual não se preocupa apenas com software, mas com todos os elementos de um sistema sociotécnico – desde hardware às interações do sistema com usuário e seu ambiente. O Modelo em Cascata, algumas vezes chamado de ciclo de vida clássico, sugere que o processo de elaboração do software é sequencial, começando pela fase de definição de requisitos, avançando pelas fases de projeto, implementação, integração e culminando na operação e manutenção da solução. O resultado de cada fase consiste de um ou mais artefatos de software elaborados e aprovados. Pode ocorrer, entretanto, o retorno a fases anteriores devido a eventuais problemas nesses artefatos repetindo com isso algumas fases (anteriores) do ciclo – por exemplo, na fase de implementação e teste de unidade são identificados problemas decorrentes do projeto ou até mesmo da definição de requisitos. Destaca-se, entretanto, que esse caminho não linear não é muito explorado pelos profissionais que utilizam tal modelo, em geral, eles o percorrem sequencialmente – fase após fase [2]. A seguir uma breve descrição de cada fase do modelo em questão, pois, conforme [4], [36], [37], elas e/ou ações relacionadas a cada uma delas estão presentes de alguma forma em outros modelos de processo e/ou práticas de software ou de gestão e organização de projetos de software.

Na fase de definição de requisitos do modelo em cascata é elaborada em conjunto com cliente do sistema/software a especificação de sistema, que é o artefato que descreve detalhadamente os requisitos do sistema – características e/ou restrições do sistema e funcionalidades ou serviços que ele deve disponibilizar aos seus usuários e interessados. Na fase de projeto de sistema e software é elaborada e modelada a arquitetura do sistema/software. Ela envolve a elaboração e modelagem de seus elementos estruturantes e eventuais relações dados aspectos de hardware e entre eles e abstrações fundamentais de software, ou seja, tudo que é necessário para a implementação e operação do sistema. Na fase de implementação e teste de unidade a estrutura de hardware é criada, por exemplo, servidores de aplicação e de banco de dado e as abstrações são codificadas (ou programadas) por meio, por exemplo, de classes – depende do paradigma ou da linguagem de programação escolhida no início do processo. O código, então, é testado individualmente para avaliar comportamento esperado (especificação de sistema/software) versus apresentado [2].

Na fase de integração e teste de sistema os programas ou código programados na fase anterior são integrados e testados também para avaliar comportamento esperado versus apresentado, todavia do ponto de vista do sistema como um todo. Uma vez aprovado pelo cliente o sistema está liberado. Na fase de operação e manutenção o sistema é instalado no hardware anteriormente definido. O sistema, então, deve ser mantido em operação e isso envolve, por exemplo, o acompanhamento do desempenho do sistema, a correção de erros ou problemas decorrentes de fases anteriores – mas só detectados agora – e o aprimoramento e/ou ampliação das funcionalidades ou serviços do sistema a medida que mudanças ou novos requisitos aparecem. Na prática, o esforço para manter o sistema em operação pode implicar na execução de ações ou atividades de uma ou mais fases anteriores [2].

O modelo em cascata é útil quando as necessidades, ou requisitos, são bem compreendidos e não há expectativa de grandes mudanças – em outras palavras, eles são praticamente fixos – e o fluxo de trabalho do desenvolvimento de software ou do projeto pode ser linear como o do modelo em si sem comprometer o resultado esperado pelo cliente. Por outro lado, a eficácia desse modelo sofre críticas há décadas dados os seguintes aspectos inerentes ao desenvolvimento de software: a incerteza natural que existe no início de muitos projetos de software ou de tecnologia – é difícil estabelecer todas as necessidades do cliente e/ou requisitos logo no início do projeto; o cliente ou o mercado não pode esperar o término de todas as fases para se ter uma versão operacional do sistema, e ainda com chance de um eventual erro propagado dentre as fases e descoberto apenas no final do processo; e, por fim, os projetos (reais) não seguem um fluxo sequencial ou linear, o ritmo deles é cada vez mais acelerado e sujeito a inúmeras mudanças [2], [4].

Na prática, diferentes organizações, projetos e equipes necessitam de diferentes formas de trabalho. Há situações que a prioridade é a previsibilidade (de datas ou de custos), então se recomenda usar ou adaptar os (modelos de) processos prescritivos, tal como o descrito anteriormente. Agora, se a prioridade da situação é flexibilidade e adaptabilidade, então se pode usar ou adaptar os (modelos de) processos ágeis, por exemplo, o Scrum e a Extreme Programming (ou XP) [4].

Os processos ou métodos ágeis valorizam mais indivíduos e interações, software em funcionamento, colaboração com o cliente e responder a mudanças mais que processos e ferramentas, documentação abrangente, negociação de contratos e seguir um plano, respectivamente. Ou seja, há valor nesses últimos quatro elementos, porém, estes métodos valorizam mais os quatro primeiros. Os signatários do manifesto ágil descreveram também doze princípios que esses métodos seguem [38]. Mas, dois deles merecem destaque aqui, nas palavras de [39]: "entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo" e "as melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis". O primeiro princípio destacado é

descrito a seguir quando se apresenta o Desenvolvimento Iterativo e os processos ágeis citados anteriormente. O segundo, entretanto, que é a auto-organização (ou equipes auto-organizáveis), de acordo com [3], significa que a empresa no nível gerencial se comprometerá em facilitar a evolução do comportamento que emerge da interação dos integrantes da equipe do projeto, em vez de prescrever ou especificar de maneira detalhada e antecipada seu comportamento. A influência na gerência nesse tipo de equipe ocorre de outra forma, por exemplo, seleção de seus integrantes e definição de quais produtos e/ou resultados devem ser gerados.

Os métodos ágeis possuem características que trazem dinamismo para o desenvolvimento de produto, motivação para equipe e cooperação entre seus integrantes, flexibilidade e adaptabilidade ao projeto e informações mais eficazes sobre a verdadeira situação do projeto [38]. Entretanto, os processos ágeis possuem limitações, por exemplo, conforme indicado por [40]: a insustentável regra ou prática por um longo período do cliente presente e a dificuldade de introduzir métodos ágeis em projetos complexos e grandes. Segundo [41], para se praticar métodos ágeis é necessário estar atento a alguns pré-requisitos, tais como, a equipe deve estar de acordo com sua utilização e também se deve buscar apoio da gerência o quanto antes.

Na prática, a necessidade de lidar com diferentes tipos de projetos dentro de uma mesma organização e ao mesmo tempo com os novos desafios do mercado, como complexidade e inovação, impulsionam o estudo e a utilização de modelos híbridos por parte dos profissionais. Esses modelos combinam princípios, práticas, técnicas e ferramentas de diferentes abordagens em um processo sistemático adequado ao contexto de negócio e tipo específico de projetos com intuito de melhorar o desempenho tanto do projeto quanto do seu produto em si proporcionando um equilíbrio entre previsibilidade e flexibilidade [13].

No cenário descrito no parágrafo anterior, destaca-se o (modelo de) processo Desenvolvimento Iterativo, pois ele é a base tanto de modelos híbridos quanto de alguns dos princípios dos processos ágeis, tais como, os dois processos flexíveis e adaptáveis citados anteriormente. Com base em [2], [4], uma iteração de software é um ciclo de trabalho que envolve a realização de ações e/ou atividades de engenharia de software e de apoio de tal maneira que ao final dele uma parte do sistema está em condições de ser entregue ao cliente. Essa parte ou incremento de software é resultado, por exemplo, de ações e atividades de definição de requisitos, projeto de sistema e software, implementação e teste de unidade, integração e teste de sistema e operação. No desenvolvimento iterativo, então, o software ou solução se torna mais e mais completo ao final de cada ciclo.

O modelo descrito anteriormente requer um esforço maior de gerenciamento e acompanhamento por parte da equipe do projeto, e de envolvimento por parte do cliente. Isso se deve a não linearidade das atividades e os vários ciclos de desenvolvimento e atividades necessários para o término do projeto. Esse tipo de modelo de processo requer um novo formato de contrato ou aquisição de serviços de desenvolvimento de software, pois não há uma especificação detalhada e completa para servir de base para uma negociação prévia. Essa ausência de especificação pode dificultar o trabalho de uma equipe independente de verificação e de validação ou de garantia da qualidade, principalmente porque este tipo de modelo de processo busca minimizar a documentação e aproximar a definição de requisitos, projeto e implementação. No desenvolvimento iterativo e incremental as decisões de projeto (design) são postergadas, o foco está em atender os requisitos prioritários normalmente, e isso pode levar a um software mal estruturado e difícil de manter [2], [4].

Por outro lado, o desenvolvimento iterativo e incremental tem várias vantagens, nas palavras de [2, p. 48]:

> "1. Os clientes não precisam esperar até a entrega do sistema inteiro para se beneficiarem dele. O primeiro incremento satisfaz os requisitos mais críticos e, dessa forma, é possível usar o software imediatamente. 2. Os clientes podem usar os incrementos iniciais como protótipos e ganhar experiência, obtendo informações sobre os requisitos dos incrementos posteriores do sistema. 3. Existe um risco menor de falha geral do projeto. Embora possam ser encontrados problemas em alguns incrementos, é provável que alguns sejam entregues com sucesso aos clientes. 4. Como os serviços de prioridade mais alta são entregues primeiro, e os incrementos posteriores são integrados a eles, é inevitável que os serviços mais importantes de sistema recebem mais testes. Isso significa que os clientes têm menor probabilidade de encontrar falhas de software nas partes mais importantes do sistema."

De acordo com [42], o Scrum, a XP e modelos híbridos que combinam os dois estão entre os processos ágeis mais utilizados dentre diferentes indústrias do mercado e governo. Segundo [37], eles trabalham bem juntos porque são iterativos e incrementais e se complementam considerando o foco de cada um: o primeiro traz ações e/ou atividades de apoio ao processo de software, tais como, gerenciamento e organização do projeto; enquanto, o segundo traz práticas mais voltadas para a engenharia de software em si.

A eXtreme Programing (ou XP) foi criada no final da década de 1980 por Kent Beck após um projeto crítico na Chrysler. A XP é estruturada em valores, princípios e práticas, em que o segundo elemento é a ligação entre o primeiro e o último. Os valores e princípios são muito semelhantes aos descritos no parágrafo sobre métodos ágeis. O conjunto de práticas da XP é um conjunto de práticas engenharia de software antigas e testadas que se apoiam, criando sinergia e com isso endereçam os problemas e/ou desafios inerentes ao desenvolvimento de software. Esse conjunto é dividido em dois grupos, as práticas primárias e as corolárias. As práticas do primeiro grupo são independentes do que está sendo realizado, e a melhoria no processo de desenvolvimento é alcançada mais rapidamente. Pode-se começar por qualquer uma delas. Agora, é interessante dominar as práticas primárias antes de utilizar as práticas do segundo grupo. O uso e a interação entre as práticas aumentam seus efeitos [36], [38], [43]. Para [1], a revisão do software é algo que auxilia na sua (garantia de) qualidade, então, conforme [36], deve-se revisar o trabalho sempre que possível. Na prática, todo e qualquer trabalho que é entregue ao cliente deve ser realizado em dupla – prática primária *Pair Programming*.

Outras práticas primárias da XP que merecem destaque: a) ciclo trimestral: uma vez por trimestre reflita e planeje detalhadamente o trabalho considerando aspectos, tais como,

a equipe, o projeto e seu progresso e alinhamento com objetivos maiores; b) ciclo semanal uma vez por semana planeje detalhadamente o trabalho focando as histórias que serão desenvolvidas durante uma iteração; c) histórias de usuário: os requisitos ou funcionalidades do sistema devem ser especificados considerando o ponto de vista do cliente. Os requisitos devem ser, na realidade, descritos idealmente pelo próprio cliente em cartões (de papel), os quais podem ser manuseados facilmente tanto pelo usuário quanto pelo desenvolvedor. Há também histórias técnicas, por exemplo, criação e preparação de um servidor de aplicação ou de banco de dados para o sistema; d) integração contínua: toda vez que um desenvolvedor terminar uma história (de usuário ou técnica), ele deve conseguir executar, testar e, se a implementação está correta, integrar o código ao código do projeto que está no repositório do projeto – e isso pode ocorrer várias vezes durante o dia de trabalho; e) equipe completa: incluir na equipe do projeto todas as pessoas com as habilidades e perspectivas necessárias para o sucesso do projeto; e, por fim, f) sentar junto: os integrantes da equipe – idealmente inclusive o cliente – precisam estar bem próximos e em um (g) espaço de trabalho aberto e informativo – o ambiente de trabalho deve fornecer informações relevantes sobre o projeto e/ou sistema e permitir uma comunicação fácil e sem barreiras [36], [41].

A XP e o Scrum aplicam o conceito de *timebox* em vários elementos dos seus processos ou métodos. Por exemplo, as ações e/ou atividades de apoio, tal como o planejamento do trabalho, consideram um período de tempo predeterminado para se realizar tal trabalho: a equipe tem uma iteração para desenvolver ou manter um incremento do produto. O *timebox* de uma iteração na XP é semanal e no Scrum, por outro lado, o período de tempo predeterminado de uma iteração – ou Sprint – pode ser de até um mês [36], [38], [44]. O desenvolvimento e manutenção de um produto (complexo) no Scrum, segundo [38, p. 31], "...é feito de uma forma iterativa e incremental – ciclos completos de desenvolvimento de duração fixa que, ao final, resultem em incrementos potencialmente entregáveis do produto. Essas iterações são chamadas de Sprints",

O Scrum é um *framework* leve que considera que o conhecimento vem da experiência e da tomada de decisões baseada naquilo que é observado – empirismo – e que se deve reduzir o desperdício e se concentrar no essencial (*lean thinking*). O Scrum ajuda pessoas, times e organizações a desenvolver e manter produtos complexos, o qual funciona bem como um contêiner para outros processos e/ou práticas. Então, por exemplo, se o produto a ser desenvolvido ou mantido é software, então ele poderia ser "preenchido" com práticas XP de desenvolvimento de software de tal forma que a equipe scrum poderia realizar seu trabalho. Por fim, o processo (ou *framework*) Scrum é composto basicamente por três papéis, três artefatos e quatro eventos (ou cerimônias), mais detalhes de cada um deles nos parágrafos a seguir [37], [38], [44].

A equipe scrum, ou *Scrum Team*, deve ser suficiente pequena para se manter ágil mas grande o suficiente para comportar todas as habilidades necessárias para desenvolver o incremento do produto – não deve ultrapassar o tamanho de 10 integrantes. A equipe é responsável por todas as atividades necessárias para se alcançar o objetivo ou meta da Sprint, o qual é norteado pelo (meta do) produto. A *Scrum Team* é auto-gerenciada (ou auto-organizada) e formada basicamente por três papéis, os *Developers* (ou desenvolvedores), o *Product Owner* (ou dono do produto, ou apenas PO) e o *Scrum Master* (especialista scrum, ou apenas SM) [44]. Outro exemplo de semelhança entre XP e Scrum está na estratégia de composição da equipe de trabalho: a equipe deve comportar todas as habilidades ou pessoas necessárias para criar um incremento do produto a cada iteração. Na XP a prática que implementa tal estratégia é prática primária chamada de Equipe Completa. No Scrum isso é referenciado quando se fala da maneira de compor o Scrum Team, em que ele deve ser multifuncional [36], [44].

O produto é um meio para entregar valor ao cliente, e isso pode ser um serviço, algo físico e/ou abstrato. O *Backlog* do Produto é uma lista priorizada e emergente de itens de trabalho – por exemplo, características ou requisitos de sistema/software. Essa lista deve ser elaborada e mantida tendo em mente a meta de produto desenvolvida junto ao cliente. O *Backlog* da Sprint é a lista de itens de trabalho de uma Sprint, a qual é criada durante o evento de *Sprint Planning* – mais detalhes a seguir. Esses itens são selecionados a partir do *backlog* do produto, contudo a lista é complementada com um plano de ação para entregá-los como um ou mais resultados da Sprint – ou um ou mais incremento utilizável e pronto do produto que agrega valor ao cliente. O incremento deve atender uma descrição formal ou uma série de medidas de qualidade para ser considerado pronto – definição de pronto [3], [44]. Os itens do *Backlog* do Produto são considerados preparados para seleção durante o evento de *Sprint Planning*. Deixar os itens preparados envolve uma ação ou atividade contínua de refinamento, a qual em geral busca "quebrar" os itens em elementos menores e mais precisos e incluir informações adicionais neles. A granularidade e as informações desses itens dependem da categoria do produto em desenvolvimento [44].

Os eventos Scrum são o *Sprint Planning* (ou Reunião de planejamento da Sprint), *Daily Scrums* (ou Scrums diário), *Sprint Review* (ou Revisão da Sprint) e *Sprint Retrospective* (ou Retrospectiva da Sprint). Os eventos são oportunidades formais de inspeção e adaptação dos artefatos e manutenção da transparência necessária, ou seja, os pilares para o funcionamento do processo são realizados e mantidos [38], [44]. As ações contínuas de preparação e manutenção dos *Backlogs* não são citadas explicitamente no guia do scrum como eventos. No entanto, é um esforço que deve ser previsto pela equipe porque será algo que ela deverá realizar durante o processo de desenvolvimento do produto – ou seja, antes, durante e após a iteração/sprint [3], [38]. Os eventos citados anteriormente são descritos a seguir, mas considerando o que foi descrito em parágrafos anteriores que uma Sprint do Scrum é na prática uma Iteração.

Uma vez preparado o *Backlog* do Produto, é na Reunião de planejamento da Iteração que a equipe de maneira colaborativa define o porquê (meta) da Iteração, seleciona a partir do *Backlog* do Produto os itens da Iteração – ou seja, o *Backlog* da Iteração – e define como eles serão transformados em um ou mais incremento do produto – os

itens são "quebrados" em tarefas. Durante o desenvolvimento do incremento do produto ocorre os Scrums Diários. Neles os integrantes que estão trabalhando nos itens do *Backlog* da Iteração avaliam o progresso do trabalho do dia anterior e definem um plano de ação para o dia seguinte. Na Revisão da Iteração o que foi desenvolvido pela equipe é inspecionado com o cliente e eventuais adaptações e/ou melhorias entram no plano da próxima Iteração. O objetivo do último evento, a Reunião de Retrospectiva, é a melhoria do processo. A equipe deve avaliar ou questionar o funcionamento dele própria e do processo como um todo e, se necessário, traçar um plano de ação (de melhoria de processo). Uma iteração se repete após a outra até que o produto a ser entregue ao cliente fique completo. Por fim, os *timebox* sugeridos para cada evento citado anteriormente em média são: oito horas para um planejamento de uma iteração/sprint de um mês, 15 minutos para os scrums diários e de duas a três horas para a revisão e retrospectiva respetivamente [38], [44].

G. Base de Dados Relacionais: MySQL, SQL e MySQL Workbench

O MySQL é um Sistema Gerenciador de Banco de Dados (SGBD) relacional *Open Source*[7] desenvolvido, distribuído e suportado pela *Oracle Corporation* [45]. Um SGBD busca disponibilizar um ambiente tanto conveniente quanto eficiente para o armazenamento e recuperação de informações, e para isso é constituído por um conjunto de dados associados a um conjunto de programas para acesso a esses dados [46]. A interação com o SGBD MySQL pode ser, conforme [45], através de um sistema cliente-servidor, o qual consiste em um servidor preparado para receber e executar comandos SQL e um cliente que permite a escrita e o envio de tais comandos.

A intenção de um SGBD com isso é proporcionar ao usuário (pessoa ou programa) uma abstração ou visão mais simplificada do banco de dados ocultando determinados detalhes sobre como os dados são armazenados e mantidos pelo sistema, e para isso ele trabalha com diferentes formas ou modelos de se enxergar os dados, neste trabalho destaca-se um deles, o Modelo Entidade-Relacionamento – ou Modelo Relacional [47]. Esse modelo baseia-se na abstração em que os problemas do mundo real são representados por meio de um ou mais conjuntos de objetos chamados entidades (ou tabelas) e eventuais relacionamentos entre elas. Essas entidades possuem suas tuplas (ou linhas) com seus respectivos atributos (ou colunas) [46]. Uma iteração de desenvolvimento de software, por exemplo, possui nome e uma lista de itens de trabalho – em que cada item possui também um nome. Esse cenário reduzido e simplificado do mundo real poderia ser modelado da seguinte forma: banco de dados iterações formado pela tabela/entidade "iteração" (colunas: nome) e pela entidade/tabela "itens de trabalho" (colunas: nome e informação que permite identificar a qual iteração o item pertence).

As entidades ou tabelas são acessadas e/ou manipuladas a partir de comandos SQL, ou *Structured Query Language*. Ela é uma linguagem-padrão declarativa de alto nível que proporciona uma interface com SGBDs relacionais, que por meio dela, o usuário especifica somente o resultado (esperado) e não o como o sistema de fato deve executar o comando, facilitando assim a interação usuário-SGBD [47]. Por exemplo, utilizando a situação meramente ilustrativa do parágrafo anterior e uma versão reduzida e simplificada da sintaxe SQL para MySQL disponível em [45], um usuário poderia criar o banco de dados iterações citado anteriormente enviando os seguintes comandos SQL a um SGBD MySQL:

```
"create database iteracoes;" e "create table iteracao (`id` int(10) unsigned NOT NULL AUTO_INCREMENT, 'nome' varchar(255) NOT NULL,PRIMARY KEY ('id'));" e "create table item_trabalho ('id' int(10) unsigned NOT NULL AUTO_INCREMENT, 'nome' varchar(255) NOT NULL, 'id_iteracao' int(10) unsigned NOT NULL, PRIMARY KEY ('id'));".
```

Os tipos de comandos SQL podem ser divididos em subconjuntos de linguagens conforme o comando (ou operação) que é realizado junto ao banco de dados: (1) subconjunto da linguagem de definição de dados (ou, em inglês, *Data Definition Language*, DDL) que é utilizado para definir a estrutura do banco de dados em si; e (2) subconjunto da linguagem de manipulação de dados (ou, em inglês, *Data Manipulation Language*, DML) que traz meios para manipular os dados do banco [47]. No parágrafo anterior foram apresentados comandos pertencentes ao primeiro subconjunto, o DDL, e no parágrafo a seguir comandos do segundo subconjunto, o DML.

Uma vez criada a estrutura (ou esquema) do banco por meio dos comandos DDL, pode-se inserir os dados e recuperá-los conforme a necessidade do usuário usando comandos DML. Voltando ao exemplo do banco de dados iterações, para inserir dados de uma iteração com dois itens de trabalho nas respectivas tabelas os comandos SQL seriam:

```
"insert into iteracao (id, nome) values ('1', 'Iteração 1 - Exemplo de inclusão na tabela iteracao');"; "insert into item_trabalho (id, nome, id_iteracao) values (10, 'Item de trabalho 1 - Exemplo 1', 1);" e "insert into item_trabalho (id, nome, id_iteracao) values (11, 'Item de trabalho 2 - Exemplo 2', 1);".
```

De outro modo, conforme [47], para recuperar (ou buscar ou selecionar) dados de uma ou mais tabela do banco utilizam-se comandos do subconjunto DML – especificamente chamados de *query language* –, tal como, o comando *select* que seleciona um (sub)conjunto de linhas de uma ou mais tabela dada uma determinada condição de seleção.

Considera-se que há mais iterações cadastradas no banco de dados fictício chamado iterações e que o usuário quer recuperar em uma seleção (ou consulta) o nome das iterações cadastradas e em outra o nome das iterações mais a quantidade de itens de trabalho de cada uma delas. Então, os comandos *select* para essas duas seleções seriam respectivamente:

```
"select nome from iteracao" e "select nome, count(it.id_iteracao) from iteracao as i inner join item_trabalho as it on i.id = it.id_iteracao group by nome;".
```

7 Código aberto (MySQL): qualquer pessoa pode baixar o software da Internet, usá-lo e modificá-lo sem pagar nada, mais detalhes em https://dev.mysql.com/.

Ressalta-se da última seleção que, conforme [46], uma seleção pode envolver diferentes funcionalidades, tais como, a possibilidade de junção de dados de diferentes tabelas de maneira eficiente e de agregação de dados – a operação *inner join* e a função agregadora *count*, respectivamente. E mais, pode-se criar uma tabela na base de dados a partir do resultado de uma seleção [45]. Por exemplo:

```
"create table numeros_iteracoes as select nome,
count(it.id_iteracao) from iteracao as i inner
join item_trabalho as it on i.id =
it.id_iteracao group by nome;".
```

Os comandos SQL apresentados até aqui podem ser escritos e enviados ao SGBD por meio de um outro programa (ou cliente no sistema cliente-servidor), ou mais especificamente, conforme [45], uma ferramenta administrativa – ou uma interface de programação de aplicativos (ou, em inglês, *Application Programming Interface*, API) –, tal como, o MySQL Workbench. Ela é uma ferramenta visual que permite modelar dados e/ou banco de dados, escrever e executar comandos SQL, configurar servidores, administrar usuários, fazer *backup* e *restore* de bancos de dados e demais atividades típicas da interação com um SGBD (MySQL) [45]. É possível consultar outros tipos de comandos SQL e recursos tanto de junção de tabelas quanto de funções de agregação de dados em [45]–[48].

III. MATERIAIS E MÉTODOS

O método de estudo deste trabalho foi baseado no processo de análise de dados *KDD-process*, o qual foi descrito em detalhes no capítulo anterior. Ele envolve ciclos ou repetições (iterativo) e é interativo. Ou seja, o ciclo de etapas desse processo pode se repetir total ou parcialmente e o resultado de uma ou mais etapas serve de entrada para a etapa ou ciclo seguinte. Na prática, a quantidade e a configuração desses ciclos dependem do objetivo ou meta do projeto ou processo de análise de dados e dos resultados encontrados no decorrer do trabalho. Portanto, foram planejados e realizados dois ciclos de execução de tarefas do *KDD-process* para criação e avaliação de diferentes modelos preditivos envolvendo iterações de desenvolvimento de software, mais detalhes na figura e nos parágrafos a seguir.

Os dados utilizados nos ciclos citados anteriormente foram os dados históricos de iterações de desenvolvimento de software de cinco projetos reais de uma empresa de Tecnologia de Informação e Comunicação (ou TIC) de um órgão governamental. Foram utilizados, na prática, apenas os dados dos projetos autorizados pela empresa por meio de processo/protocolo oficial[8], e não necessariamente todos os projetos solicitados pelos pesquisadores. Uma parte deles estava armazenada no banco de dados de uma das ferramentas que suporta o PDS (ou Processos de Desenvolvimento de Sistemas) da empresa e a outra parte dos dados foi obtida por meio de questionamentos junto aos integrantes ou responsáveis dos respectivos projetos. Esses dados estão diretamente relacionados com planejamento e execução de iterações de software, então, neles há informações, tais como, total de horas planejadas e executadas, total de pontos, quantidade de desenvolvedores, *backlog* da iteração (quantidade e itens), duração etc – mais detalhes sobre esses dados em tabelas descritivas a seguir. O processo de desenvolvimento de software/sistema[9] utilizado pelas equipes dos projetos é baseado no Scrum e XP. Na realidade, ele faz parte do PDS da empresa, o qual serve de guia ou referência para as equipes de desenvolvimento e manutenção de sistemas/software. Ele é um modelo híbrido porque ele combina princípios e práticas de diferentes modelos de desenvolvimento de software e de gestão de projetos para criar processos ou abordagens distintas e aumentar a "caixa de ferramenta" que fica disponível para as equipes em termos de processo de software.

A Figura 3 apresenta os passos dos dois ciclos de execução de etapas do *KDD-process* necessários para criar e avaliar os modelos preditivos de iterações de software.

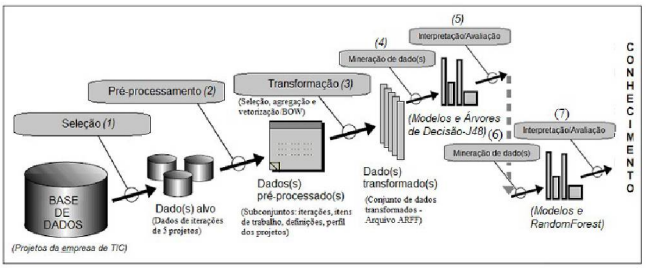


Figura 8. Passos do método de estudo/trabalho. Adaptado de [10]

Na Figura 8, o primeiro ciclo de criação e avaliação de modelos preditivos utilizou a técnica/algoritmo de Árvore de Decisão/J48 e envolveu cinco passos – de 1 a 5. O segundo ciclo de criação e avaliação de modelos preditivos utilizou a técnica/algoritmo de Floresta Aleatória/*RandomForest* e envolveu dois passos apenas – 6 e 7. Esse segundo ciclo utilizou o mesmo conjunto de dados que foi criado durante os passos 1, 2 e 3.

As informações do *passo 1 (seleção)* já foram descritas no início deste capítulo, porém, destaca-se novamente que o objetivo tanto deste primeiro ciclo quanto do segundo é a criação e avaliação de modelos preditivos envolvendo o resultado (sucesso ou não) de iterações de desenvolvimento de software.

No *passo 2 (pré-processamento)*, foram criados quatro subconjuntos de dados alvo, em que a origem dos três primeiros foi a base de dados citada anteriormente e a do último respostas a questionamentos realizados junto aos responsáveis e/ou integrantes das equipes dos projetos selecionados. Os três primeiros subconjuntos de dados alvo são: (i) informações gerais (ou macro) das iterações de desenvolvimento e/ou de manutenção de software; (ii) informações dos itens de trabalho associados a cada uma das iterações; e (iii) as descrições das definições de preparado e de pronto utilizadas em cada projeto de desenvolvimento e/ou esforço de manutenção de software. Os subconjuntos

8 Trâmite realizado por meio do sistema https://www.eprotocolo.pr.gov.br/ (número do protocolo é 16.704.458-1). Resumidamente, o processo envolveu a elaboração e entrega de um ofício de parceria entre instituição de ensino e empresa (001/2020-UFPR/SEPT/EIAA), o preenchimento e envio de um formulário de coleta de dados, apresentações da pesquisa perante o(s) responsável(is) pelo projeto dentro da empresa, assinatura(s) do processo (termo de compromisso, por exemplo) e, por fim, disponibilização/coleta dos dados. Esse processo levou aproximadamente 5 meses.

9 Este processo foi elaborado e oficialmente disponibilizado para toda a empresa em Julho/2016 e está em uso até o momento de elaboração deste estudo. Ele é internamento conhecido como Abordagem Ágil e o resultado do primeiro projeto usando um piloto de tal abordagem pode ser acessado em [61].

foram construídos por meio da escrita e execução de comandos SQL (*create* e *select* com *join*) envolvendo diferentes tabelas da base de dados relacionais MySQL (versão 8) de uma das ferramentas que suporta PDS da empresa. Esses comandos foram enviados ao banco de dados através da ferramenta MySQL Workbench (versão 8), e com isso tabelas (temporárias) foram geradas dentro do respectivo banco para cada subconjunto de dados ficando eles disponíveis para as próximas etapas e/ou ciclos.

A descrição dos atributos de cada um dos quatro subconjuntos é apresentada em tabelas descritivas a seguir.

Tabela 3. Subconjunto informações gerais das iterações de software

Atributo	Descrição (domínio)
proj_id	Identificador do projeto (número)
itr_id	Identificador da iteração (número)
itr_seq	Sequência de execução da iteração no contexto do projeto ou manutenção de software (número)
itr_nome	Nome da iteração (texto)
itr_descricao	Descrição da iteração (texto)
itr_tem_descricao	"sim" caso a descrição da iteração foi preenchida, "nao" no caso contrário (texto).
itr_duracao	Quantidade de dias entre a data de início e a data de término (número) da iteração.
itr_burndown_criado	"sim" se foi criado um burndown da solução/projeto, "nao" no caso contrário (texto).
itr_equipe_ttl	Quantidade de integrantes que participaram da realização da iteração (número).
itr_equipe_ids	Composição da equipe (texto).

Ressalta-se que para cada projeto há um *backlog* do produto/projeto e um ou mais *backlog* da iteração/sprint. Foram coletadas e preprocessadas, então, informações gerais de cada iteração de software dos *backlogs*/projetos selecionados para a realização deste estudo, ver Tabela 3 para descrição de tais informações. O valor do atributo *itr_equipe_ids* ou composição da equipe citado na tabela em questão é definido da seguinte forma: identificador ou código único de cada desenvolvedor – ou id do desenvolvedor – que participou da iteração executando ou desenvolvendo um ou mais item de trabalho, em que na situação de mais de um desenvolvedor os ids foram separados por "/". Por exemplo, equipe com um desenvolvedor com identificador igual a 100 o valor desse atributo seria "100"; de outro modo, equipe com três desenvolvedores com ids iguais a 200, 220 e 400 o valor desse atributo seria "200/220/400".

Tabela 4. Subconjunto informações dos itens de trabalho das iterações

Atributo	Descrição (domínio)
proj_id	Identificador do projeto (número)
itr_id	Identificador da iteração (número).
it_id	Identificador do item de trabalho (número).
it_nome	Nome do item de trabalho (texto).
it_status	Situação do item de trabalho ao final/fechamento da iteração (texto): "atribuído", "resolvido" e "fechado".
it_pontos	Quantidade de pontos do item de trabalho (número).
it_qtd_ttl_tarefas	Total de tarefas associadas ao item de trabalho (número).
it_ttl_hrplanej	Total de horas planejadas para realização do item de trabalho (número): soma das horas planejadas para as tarefas associadas ao item.
it_ttl_hrexec	Total de horas executadas para realização do item de trabalho (número): soma das horas executadas para as tarefas associadas ao item.
it_foi_div	1 (um) caso o item de trabalho foi dividido (ou detalhado ou refinado) durante ou no final da iteração, 0 (zero) no caso contrário.
it_eh_res_div	1 (um) caso o item de trabalho é resultado da divisão (ou detalhamento ou refinamento) de outro item, 0 (zero) no caso contrário.
it_add_cf_ou_pcf	Indica se o item de trabalho foi adicionado ao *backlog* da iteração no planejamento (confirmação) da iteração ou após tal cerimônia: "cf" e "pcf", respectivamente.

No *backlog* da iteração há elementos ou itens de trabalho que podem representar, por exemplo, uma funcionalidade do sistema/solução (descrita como uma história de usuário), um ajuste a ser realizado em alguma tela ou funcionalidade do sistema, um bug a ser corrigido etc. Pode-se visualizar na Tabela 4 as informações que foram coletadas e preprocessadas para representar tais itens, mas ressaltando que as informações das tarefas de cada item de trabalho foram sumarizadas em atributos e não consideradas em (mais) um subconjunto.

Tabela 5. Subconjunto definições de preparado e de pronto

Atributo	Descrição (domínio)
proj_id	Identificador do projeto (número)
def_id	Identificador do registro da definição (número)
def_tipo	Indica se a descrição/registro é da definição de preparado ou da definição de pronto: "preparado" ou "pronto", respectivamente (texto).
descricao	Descrição da definição (texto).

Na Tabela 5, finalizando a descrição das informações coletadas e preprocessadas a partir da base de dados relacionais citada no início deste capítulo, pode-se observar a descrição dos atributos da definição de preparado e da definição de pronto. No cenário das equipes dos projetos selecionados, a primeira registra os critérios para avaliar se um item de trabalho está preparado para fazer parte do *backlog* da iteração e a segunda o se resultado produzido por ele está pronto para ser liberado ao usuário/cliente.

Tabela 6. Subconjunto informações perfil do projeto/*backlog*

Atributo	Descrição
proj_id	Identificador do projeto (número).
proj_nome	Nome projeto ou manutenção (texto).
ling_prog	Linguagem/plataforma de desenvolvimento (texto).
tipo_bd	SGBD (texto).
proj_ou_sust	Indica a natureza do esforço predominante das iterações (texto): "projeto" (desenvolvimento ou projeto de software) ou "manutencao" (manutenção de sotware).
publico_alvo	Indica o público-alvo predominante do esforço das iterações (texto): "interno" (cliente interno, na prática são os próprios colaboradores da em-

	presa) ou “externo”.

Encerrando a descrição dos quatro subconjuntos de dados alvo, na Tabela 6 é possível visualizar informações que foram coletadas e preprocessadas para auxiliar na definição do perfil de cada projeto/*backlog*. Ao final desse passo, o de número 2, os quatro subconjuntos estavam preparados para a execução do próximo passo do ciclo de criação do conjunto de dados base/inicial e variações a partir dele.

No *passo 3 (transformação)* foram criados diferentes conjuntos de dados para a aplicação no passo seguinte das técnicas/algoritmos Árvore de Decisão/J48 e Floresta Aleatória/*RandomForest*. Na prática, foi criado um conjunto de dados base e a partir dele gerados cinco conjuntos de dados com diferentes configurações de características ou atributos, veja a seguir mais detalhes desse processo.

Tabela 7. Conjunto de dados (base): versão inicial

Atributo(s)	Origem e indicativo de transformação
id geral	Sequência gerada para identificar cada instância (ou linha) do conjunto de maneira única.
itr_seq, itr_tem_descricao, itr_duracao, itr_burndown_criado, itr_equipe_ttl, itr_equipe_ids	Subconjunto de origem: iterações. Não houve operação/transformação envolvendo as informações.
proj_nome, ling_prog, tipo_bd, proj_ou_sust, publico_alvo	Subconjunto de origem: perfil do projeto/*backlog*. Não houve operação/transformação envolvendo as informações.
proj_def_prep_qtd, proj_tem_def_prep, proj_def_pronto_qtd, proj_tem_def_pronto	Subconjunto de origem: definições de preparado e pronto. Não houve operação/transformação envolvendo as informações.

A primeira atividade foi criar a versão inicial do conjunto de dados base considerando os atributos que não precisaram de nenhum processo de transformação, ver Tabela 7 para lista de tais atributos.

Tabela 8. Conjunto de dados (base): atributos não textuais gerados a partir do subconjunto itens de trabalho

Atributo (operação)
itr_its_ttl_cf (contagem dos itens), itr_tarefas_ttl_cf (soma it_qtd_ttl_tarefas), itr_hrplanej_ttl_cf (soma it_ttl_hrplanej), itr_pts_ttl_cf (soma it_pontos), itr_its_div_ttl (soma it_foi_div), itr_its_ehresdiv_ttl (soma it_eh_res_div), itr_tarefas_ttl (soma it_qtd_ttl_tarefas), itr_hrplanej_ttl (soma it_ttl_hrplanej), itr_hrexec_ttl (soma it_ttl_hrexec), itr_pts_ttl (soma it_pontos), itr_tarefas_ttl_res (soma it_qtd_ttl_tarefas), itr_pts_ttl_res (soma it_pontos), itr_pts_itsdiv_ttl (soma it_pontos com filtro it_foi_div = 1)
class_res_itr (detalhes no parágrafo a seguir)

Agora, na segunda atividade foram acrescentados ao conjunto de dados os atributos numéricos e categóricos criados a partir de operações de agregação, tais como, contagem e soma envolvendo o subconjunto itens de trabalho, ver Tabela 8. O atributo “class_res_itr” da respectiva tabela é base ou referência para etapa de mineração de dados envolvendo aprendizado supervisionado. O atributo informa a classe (resultante) de cada instância (ou linha). Ele é o atributo que define se uma iteração terminou com sucesso (ou não), ou seja, seu domínio é “sucesso” e “falha”. O valor desse atributo foi definido da seguinte forma: valor categórico “sucesso” se o desempenho da iteração foi igual ou maior a 0,90 ou “falha” se desempenho foi inferior a 0,90, em que o desempenho é o valor obtido da divisão entre a quantidade de itens de trabalho resolvidos e total de itens de trabalho da iteração. Por fim, duas observações sobre o conteúdo da Tabela 8: primeira, o sufixo “_cf” nos atributos significa que a informação contida neles está associada com o momento de confirmação do planejamento da iteração, ou seja, a equipe não começou a execução da iteração/sprint – os integrantes não começaram a executar e/ou desenvolver os itens de trabalho do *backlog* da iteração/sprint; segunda, o sufixo “_res” significa que a informação contida nos respectivos atributos está associada com itens de trabalho resolvidos.

Ressalta-se que para realizar as duas atividades anteriores foram utilizadas as mesmas tecnologias de banco de dados relacionais citadas no passo anterior, e descritas no capítulo fundamentação teórica. Na prática, foi gerado ao final dessas duas atividades uma tabela temporária em uma base de dados relacional MySQL (versão 8), a qual foi acessada pelo programa Python citado a seguir.

Na terceira atividade de construção do conjunto base foram adicionados os atributos (ou características) resultantes da transformação das informações textuais, tais como, nome e descrição da iteração, definições de preparado e de pronto e nomes dos itens de trabalho. Essas informações textuais passaram por um processo de tokenização e vetorização baseado no método *Bag of Word (ou BoW)*. Esse processo foi automatizado por meio de um programa Python em que a sua entrada é a tabela/base de dados MySQL elaborada nas atividades anteriores e a sua saída é um arquivo ARFF/WEKA com a estrutura de colunas conforme apresentado na Tabela 10. O programa Python em questão foi desenvolvido através do ambiente de desenvolvimento Spyder e utilizou diferentes recursos da linguagem de programação em questão, tais como, a CountVectorizer do módulo sklearn, a stopwords.words('portuguese') do módulo nltk e a *punctuation* do módulo string. A Tabela 9 a seguir apresenta um resumo do resultado dessa transformação de dados textuais.

Tabela 9. Conjunto de dados (base): transformação dos atributos textuais

Informação textual de referência (operação/filtro)/Subconjunto de origem	Resultado da aplicação do método *BoW*
descricao (com filtro “def_tipo = ‘preparado’”)/ Definições de Preparado e de Pronto	Foi gerado um vetor de 78 características: “proj_dprep_<n>”, onde <n> pode variar de 0 a 77.
descricao (com filtro “def_tipo = ‘pronto’”)/Definições de Preparado e de Pronto	Foi gerado um vetor de 27 características: “proj_dpron_<n>”, onde <n> pode variar de 0 a 26.
itr_nome e itr_descricao (concatenação)/Iterações.	Foi gerado um vetor de 671 características: “itr_nmdesc_<n>”, onde <n> pode variar de 0 a 670.
it_nome (com filtro “it_add_cf_ou_pcf = ‘cf’”)/ Itens de Trabalho	Foi gerado um vetor de 1029 características: “its_ncf_<n>”, onde <n> pode variar de 0 a 1028.
it_nome /Itens de Trabalho	Foi gerado um vetor de 1138 características: “its_n_<número da coluna>”, onde <n> pode variar de 0 a 1137.

Em resumo, os quatro subconjuntos selecionados e preprocessados foram transformados em um único conjunto de dados com 2972 características e 114 instâncias (ou linhas) – onde, 45 são da classe "sucesso" e 69 da classe "falha". Essas instâncias trazem os valores das características ou atributos das iterações de desenvolvimento de software dos cinco projetos selecionados para a realização deste estudo. Por fim, a Tabela 10 apresenta a lista completa de características.

Tabela 10. Conjunto de dados (base): características (nº coluna)

itr_seq (1)	itr_burndown_criado (13)	itr_pts_ttl_cf (25)
proj_nome (2)	itr_equipe_ttl (14)	itr_pts_ttl (26)
ling_prog (3)	itr_equipe_ids (15)	itr_pts_ttl_res (27)
tipo_bd (4)	itr_its_ttl_cf (16)	itr_pts_itsdiv_ttl (28)
proj_ou_sust (5)	itr_its_div_ttl (17)	its_n_[0 a 1137] (29-1166)
publico_alvo (6)	itr_its_ehresdiv_ttl (18)	its_ncf_[0 a 1028] (1167-2195)
proj_def_prep_qtd (7)	itr_tarefas_ttl_cf (19)	itr_nmdesc_[0 a 670] (2196-2866)
proj_tem_def_prep (8)	itr_tarefas_ttl (20)	proj_dpron_[0 a 26] (2867-2893)
proj_def_pronto_qtd (9)	itr_tarefas_ttl_res (21)	proj_dprep_[0 a 77] (2894-2971)
proj_tem_def_pronto (10)	itr_hrplanej_ttl_cf (22)	class_res_itr (2972)
itr_tem_descricao (11)	itr_hrplanej_ttl (23)	
itr_duracao (12)	itr_hrexec_ttl (24)	

Neste ponto do passo 3 (transformação) foram preparados cinco variações de conjuntos de dados a partir do conjunto de dados base, e foram essas variações que serviram de entrada nos passos a seguir envolvendo mineração de dados. As variações buscaram explorar diferentes composições ou configurações das características do conjunto base, tais como, todos os atributos possíveis, somente os que estão disponíveis no momento do planejamento (confirmação) da iteração, somente os atributos "não textuais" – ou seja, os dados que passaram pela aplicação do método BoW foram ignorados – e, por fim, variações geradas a partir da aplicação do algoritmo CFS nos conjuntos "ds_comp" e "ds_planej" – para mais detalhes ver Tabela 11 a seguir.

Tabela 11. Conjuntos de dados (variações utilizadas): nome, estratégia e atributos (resumo)

Conjunto de dados	Estratégia de composição do conjunto de dados	Atributos (resumo)
ds_comp	Inclusão de todos os atributos do conjunto de dados base com exceção do its_ncf_* (ele é um subconjunto do its_n_*).	Tabela 10 menos: its_ncf_[0 a 1028].
ds_planej	Somente os atributos disponíveis na confirmação do planejamento da iteração.	Tabela 10 menos: 15, 17, 20, 21, 23, 24, 26, 27, 28 e 29-1166 (ou, its_n_[0 a 1137]).
ds_naotxt	Somente os atributos numéricos e categóricos – ou os "não textuais".	Tabela 10 menos os atributos 29 a 2971.
ds_alg_comp	Aplicação do algoritmo CFS no conjunto "ds_comp".	32 atributos: itr_duracao, itr_equipe_ids, itr_its_div_ttl, itr_its_ehresdiv_ttl, itr_pts_ttl_res, itr_pts_itsdiv_ttl, its_n_(22 diferentes colunas) e itr_nmdesc_(4 diferentes colunas).
ds_alg_planej	Aplicação do algoritmo CFS no conjunto no conjunto "ds_planej".	55 atributos: itr_seq, itr_duracao, itr_its_ttl_cf, itr_its_ehresdiv_ttl, its_ncf_(43 diferentes colunas) e itr_nmdesc_(8 diferentes colunas).

Destaca-se que o atributo que informa a classe de cada instância estava presente no conjunto de dados independente da variação de composição ou configuração. Pois, ele é o atributo que rotula ou identifica o resultado de cada instância de iteração de software, o "class_res_itr" ("sucesso" ou "falha") e as técnicas de mineração de dados selecionadas para este estudo e aplicadas na sequência do processo em questão dependem dele para criação dos modelos preditivos.

A preparação ou organização dos diferentes conjuntos descritos na Tabela 11 foi realizada através da ferramenta WEKA utilizando (i) a funcionalidade de visualização e edição de arquivos ARFF disponível no menu/opção *Tools, ArffViewer* do *WEKA GUI Chooser* e (ii) a funcionalidade *Explorer* da mesma interface gráfica especificamente as abas/seções *Preprocess* e *Select atributes*. A aplicação do algoritmo CfsSubsetEval (CFS) foi realizada por meio da aba/seção *Select atributes* selecionando o método de busca *Best First* e o modo *Cross-validation* com *folds* igual a 10 e *seed* igual a 1. Conforme [17], nesse modo de seleção o algoritmo não apresenta diretamente uma lista de atributos (selecionados) e sim o número (%) de *folds* que o atributo esteve presente durante a tarefa de busca ou seleção de característica, ou seja, de acordo com [18], quanto maior o valor mais o atributo está (cor)relacionado com a classe resultante de classificação. Portanto, os atributos com valor igual a 0 (%) não foram selecionados para compor os modelos.

No *passo 4 (mineração de dados)* foram aplicadas as seguintes técnicas/algoritmos nos conjuntos de dados: Árvore de Decisão especificamente o algoritmo J48, selecionada baseado em [5], [51], [62] e a técnica/algoritmo *ZeroR* como linha de base (inicial).

A aplicação da técnica/algoritmo Árvore de Decisão/J48 foi realizada por meio da funcionalidade *Experimenter* do *WEKA GUI Chooser* em que as configurações na seção *Setup* foram baseadas em [22], a saber: a) *Experiment Type*: *Cross-validation/Number of folds* igual a 10 e modo *Classification*; b) *Iteration Control*: *Number of repetions* igual a 10 e modo *Data set first*; c) *Datasets*: foram selecionados os cinco arquivos ou conjuntos ARFF identificados na Tabela 11; d) *Algorithms* (e parâmetros): (1) "*rules.ZeroR '*" e (2) "*trees.J48 '-C 0.25 -M 2*". Nessa configuração, segundo [17], pode-se observar que os valores das métricas de desempenho dos modelos gerados são a média dos valores obtidos nas dez aplicações realizadas em cada conjunto de dados.

Os resultados ou modelos gerados durante o passo anterior e a *interpretação/avaliação (passo 5)* deles estão descritos no capítulo a seguir. Portanto, neste ponto do processo de análise de dados encerra-se o primeiro e

começa-se o segundo ciclo de execução de etapas do KDD-process instanciado para este estudo. Conforme descrito no início deste capítulo, no segundo ciclo foi utilizado as mesmas variações de conjuntos de dados utilizadas no primeiro ciclo, então, não foi necessário repetir as etapas de seleção, pré-processamento e transformação, mais detalhes a seguir.

No *passo 6 (repetição da etapa de mineração de dados)* foi aplicada a técnica Floresta Aleatória especificamente o algoritmo *RandomForest*, selecionada com base em [5], [51], [62]. De acordo com [17], [19], essa técnica trabalha com uma coleção de árvores de decisão/classificação, então, normalmente não é possível visualizar a estrutura de árvore gerada, tal como, no J48 – e, por consequência, não é possível identificar e/ou avaliar facilmente eventuais padrões existentes dentre os atributos e/ou dados. Mas, baseado em [63], essa técnica/algoritmo tem bons resultados perante um espectro abrangente de problemas. Então, no lugar de focar na estrutura do modelo, o objetivo deste ciclo foi verificar se o desempenho dos modelos gerados pelo *RandomForest* seria melhor que o do J48.

A aplicação da técnica/algoritmo foi realizada também por meio da funcionalidade *Experimenter* do *WEKA GUI Chooser* utilizando as mesmas configurações informadas no passo 4 com exceção do item (d) *Algorithms* (e parâmetros), em que seu valor neste passo foi o seguinte: "*trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1*". Não foi necessário, portanto, selecionar outros algoritmos porque a linha de base a partir deste ponto do processo de análise de dados se tornou os modelos gerados no passo anterior com a aplicação do J48.

O *hardware* utilizado neste estudo foi um *notebook Dell Inspiron* 15500 com processador *Intel Core* i7 (7500U/2,70 GHz), memória RAM de 8 GB, armazenamento SSD[10] de 500 GB, sistema operacional *Windows* 10 *Home Single Language*. As versões dos softwares/ferramentas: WEKA 3.9.3, *Python* 3.7, *Spyder* 3.3.3, *MySQL Community Server* 8.0.13 e, por fim, *MySQL Wokbench* 8.0 *CE*.

Enfim, os resultados ou modelos gerados durante o passo 6 e a *interpretação/avaliação deles – o passo 7* – estão descritos no capítulo a seguir.

IV. RESULTADOS E DISCUSSÕES

Neste capítulo são apresentados os modelos preditivos e padrões encontrados com os dois ciclos de execução de tarefas de mineração e análise de dados do método de estudo descrito no capítulo anterior. O desempenho dos modelos gerados durante os ciclos foram avaliados e/ou interpretados considerando os valores das seguintes métricas: Acurácia, MAE e AUROC. Os valores foram obtidos por meio da aba/seção *Analyse* da funcionalidade *Experimenter* da ferramenta WEKA em que o "*" indica que o valor mensurado apresenta uma diferença significativa[11] se comparado com o seu correspondente na linha de base (ou LB).

A. Árvore de Decisão/J48: Modelos preditivos e Padrões

No primeiro ciclo de execução os modelos preditivos foram gerados a partir da aplicação da técnica/algoritmo ZeroR (linha de base) e Árvore de Decisão/J48.

Tabela 12. Conjuntos de Dados versus *ZeroR* e J48: Modelos Preditivos

Conjunto de dados (nº atributos)	*ZeroR* (LB)			Árvore de Decisão/J48		
	Acurácia (% correta)	MAE	AUROC	Acurácia (% correta)	MAE	AUROC
ds_comp (1943)	60,61	0,48	0,50	88,77*	0,12*	0,88*
ds_planej (1825)	60,61	0,48	0,50	53,65	0,46	0,52
ds_naotxt (29)	60,61	0,48	0,50	86,65*	0,13*	0,87*
ds_alg_comp (33)	60,61	0,48	0,50	92,75*	0,09*	0,93*
ds_alg_planej (56)	60,61	0,48	0,50	63,87	0,40*	0,62*

Os modelos inferidos pelo J48 apresentaram acurácias que superaram a linha de base, tais como, 86,65% do modelo ds_naotxt-J48 e 92,75% do modelo ds_alg_comp-J48. O modelo ds_alg_comp-J48 superou a linha de base também nas duas outras métricas, a MAE (0,09) e a AUROC (0,93), tanto esse modelo quanto o ds_naotxt-J48 tiveram um bom desempenho mesmo com uma quantidade reduzida de características ou atributos – apenas 33 e 29 atributos, respectivamente. Agora, alguns modelos, tais como, ds_planej-J48 (53,65%) e ds_alg_planej-J48 (63,87%) apresentaram desempenho pior ou semelhante a linha de base. Dado esse cenário, o desempenho dos modelos ds_comp-J48 e ds_alg_comp-J48 demonstraram que é possível predizer o resultado de uma iteração de desenvolvimento de software a partir de dados históricos. Esses resultados corroboraram com [51], o qual enfatiza a possibilidade de uso de resultados de iterações ou projetos anteriores como base para o planejamento do que está em andamento; esses resultados também corroboraram com [5], pois eles utilizaram dados históricos para elaborar modelos capazes de prever tanto a capacidade (de desenvolvimento) de uma equipe quanto o resultado de uma iteração de software.

A acurácia média (89,39%) dos modelos que usaram atributos tanto do momento do planejamento quanto da execução da iteração de software (ds_comp-J48, ds_naotxt-J48 e ds_alg_comp-J48) foi maior que a média (58,76%) dos modelos que usuram apenas atributos do momento de planejamento/confirmação da iteração (ds_planej-J48 e ds_alg_planej-J48), aproximadamente 31% a mais. Ou seja, esse cenário, mais o descrito no parágrafo anterior envolvendo o bom desempenho dos modelos mesmo com uma quantidade reduzida de atributos, reforçam, por

10 Disco de Estado Sólido (ou, em inglês, *Solid State Drive - SSD*), mas detalhes em https://www.dcce.ibilce.unesp.br/~aleardo/cursos/arqcomp/Semin_SSD.pdf

11 Conforme [17], a funcionalidade *Experiment* da ferramenta WEKA traz esse indicativo em função de testes estatísticos aplicados nos resultados/métricas de desempenho dos modelos – o teste é o *Paired T-Tester (corrected)* com significação 0,05. Para mais detalhes sobre esse recurso ver [17] e sobre teste estatístico e significação estatística ver [64, p. 225].

exemplo, conforme [4], [44], que não é necessário um sistema de métricas ou variáveis complexo para melhorar o processo (local) de software da equipe e/ou qualidade do produto gerado por ela – comece simples, porém, se certifique que as variáveis selecionadas e avaliadas agregam valor ao sistema. Ou ainda, conforme [3, p. 449], "[...]não precisamos de medidas perfeitas; precisamos de medidas que nos ajudem a responder perguntas".

O desempenho do modelo formado apenas por atributos numéricos e categóricos ficou próximo do desempenho do modelo completo: ds_naotxt-J48 (86,65/0,13/0,87) versus a do modelo ds_comp-J48 (88,77/0,12/0,88). Pode-se, então, em determinadas situações escolher o primeiro modelo que é mais simples de ser treinado e implementado do que o segundo, que requer o custo adicional de transformação dos dados textuais dos itens de trabalho das iterações. A diferença de desempenho entre eles é pequena mas o custo de elaboração não.

Os modelos inferidos utilizando os conjuntos de dados formados a partir da aplicação do algoritmo CFS obtiveram um desempenho melhor em termos de acurácia e MAE que seus respectivos modelos com a quantidade original de atributos, a saber (modelo/nº de atributos [acurácia]): ds_comp-J48/1943 [88,77%] versus ds_alg_comp-J48/33 [92,75%] e ds_planej/1825[53,65%] versus ds_alg_planej/56[63,87%]. Essa melhora era de certa forma esperada dado que, conforme [11], [65], a técnica implementada pelo algoritmo em questão busca os (subconjuntos de) atributos mais relevantes para o resultado do modelo e isso tende a melhorar seu desempenho.

Nos parágrafos a seguir as estruturas ou árvores de decisão dos modelos com melhor acurácia descritos anteriormente. Essas árvores de decisão foram avaliadas com intuito de identificar eventuais padrões. Veja a seguir tanto as estruturas ou árvores de decisão inferidas pelo algoritmo J48 quanto o resultado dessa avaliação. As imagens ou figuras das árvores de decisão foram exportadas por meio de funcionalidade específica do software WEKA, mais detalhes em [17].

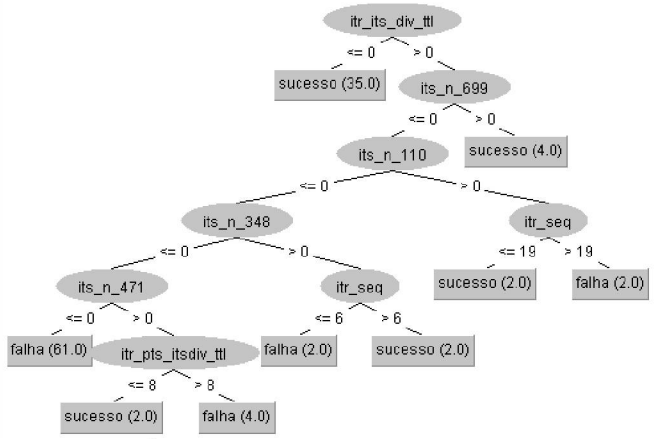


Figura 9. Árvore de Decisão do ds_comp-J48. Figura do autor.

A Figura 9 apresenta a árvore de decisão inferida pelo J48 utilizando o conjunto de dados ds_comp. Esse conjunto tinha 1943 atributos mas a estrutura inferida ficou com apenas 8 elementos – mas 7 atributos distintos – desconsiderando a classe ou rotulo (elemento retangular). O atributo mais relevante dentre eles é o itr_its_div_ttl. Pode-se observar a presença de atributos relacionados com os dados textuais do nome dos itens de trabalho das iterações, por exemplo, its_n_699 e its_n_110.

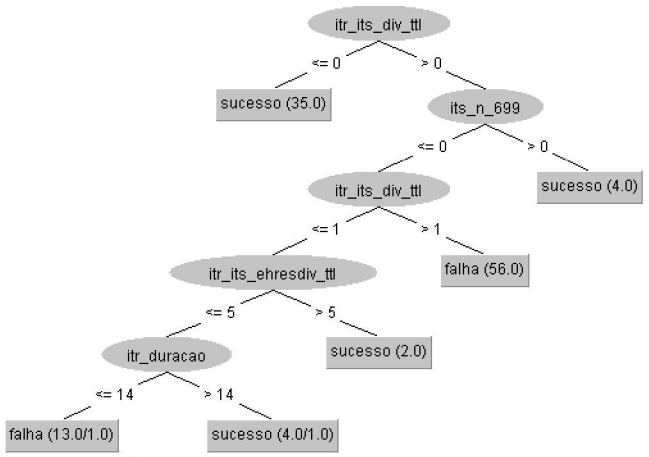


Figura 10. Árvore de Decisão do ds_alg_comp-J48. Figura do autor.

Na Figura 10, a árvore de decisão inferida pelo J48 utilizando o conjunto de dados ds_comp. Esse conjunto de dados tinha 33 atributos mas a estrutura inferida ficou com apenas 5 elementos/atributos desconsiderando a classe. O atributo mais relevante e a presença de atributo relacionado com dados textuais dos itens de trabalho da iteração são características semelhantes a estrutura anterior (Figura 4).

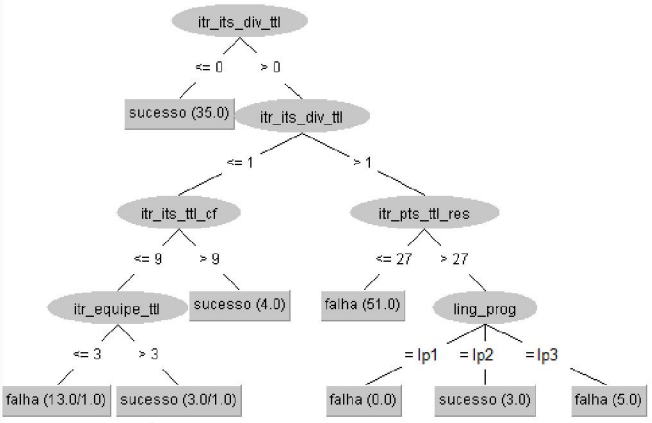


Figura 11. Árvore de Decisão do ds_naotxt-J48. Figura do autor.

A Figura 11 apresenta a árvore de decisão inferida pelo J48 utilizando o conjunto de dados ds_naotxt. Esse conjunto de dados tinha 29 atributos mas a estrutura inferida ficou com apenas 6 elementos – mas 5 atributos distintos – desconsiderando a classe. O atributo mais relevante novamente é o itr_its_div_ttl.

Em resumo, os seguintes atributos foram considerados relevantes pelos algoritmos para prever o resultado das iterações de software: itr_its_div_ttl, itr_its_ttl_cf, itr_pts_ttl_res, itr_equipe_ttl, ling_prog, itr_its_ehresdiv_tll, itr_duracao, itr_seq, itr_pts_itsdiv_ttl e its_n_[110, 348, 471, 699].

Os atributos listados no parágrafo anterior destacaram na prática em nível iteração-incremento o que é apresentado pela teoria em nível projeto quanto a aspectos importantes para o processo de planejamento de software, são eles:

escopo (requisitos que devem ser fornecidos aos clientes ou usuários), recursos (pessoas e tecnologias necessários para executar o trabalho de desenvolvimento de software) e, por fim, estimativas de software e/ou dados históricos (tamanho, complexidade, custo e/ou esforço) [4, p. 606–609]. Nesse cenário, pode-se dizer que os atributos itr_equipe_ttl e ling_prog estão associados com recursos; os atributos itr_its_ttl_cf (expectativa de itens da iteração), itr_duracao e itr_seq (momento do projeto) estão relacionados com estimativas e itr_pts_ttl_res com dados históricos; e, por fim, os atributos itr_its_ttl_cf (tamanho inicial do *backlog* da iteração), its_n_[110, 348, 471, 699], itr_its_ehresdiv_tll e itr_its_div_ttl associados com escopo.

Ainda sobre os atributos considerados relevantes pelos algoritmos, o atributo itr_its_div_ttl merece destaque porque ele está presente em todas as estruturas de árvore como nó raiz. Conforme pode-se observar nas estruturas de decisão apresentadas anteriormente, apenas avaliando o valor desse atributo os respectivos modelos foram capazes de classificar de imediato aproximadamente 31% das iterações – 35 de 114 instâncias foram imediatamente classificadas como "sucesso" apenas em função da análise do valor dele. Relembrando, o seu valor é o total de itens de trabalho que foram divididos durante ou no final da iteração. Essa divisão ocorre normalmente porque, de acordo com [3], o elemento era "grande demais" e/ou não é compreendido pela equipe.

B. Floresta Aleatória/RandomForest: Modelos preditivos

No segundo ciclo de execução método de estudo deste trabalho, foram inferidos modelos pelo *RandomForest* utilizando as mesmas variações de conjunto de dados utilizadas no ciclo anterior, porém, aqui a linha de base passou a ser o desempenho dos modelos anteriormente inferidos pelo J48, mais detalhes na tabela e parágrafo a seguir.

Tabela 13. Conjuntos de Dados versus J48 e *RandomForest:* Modelos Preditivos

Conjunto de dados (nº atributos)	J48 (LB)			*RandomForest (RF)*		
	Acurácia (% correta)	MAE	AUROC	Acurácia (% correta)	MAE	AUROC
ds_comp (1943)	88,77	0,12	0,88	76,36*	0,40*	0,83
ds_planej (1825)	53,65	0,46	0,52	60,72	0,45	0,61
ds_naotxt (29)	86,65	0,13	0,87	91,10*	0,23*	0,98*
ds_alg_comp (33)	92,75	0,09	0,93	93,38	0,19*	0,98*
ds_alg_planej (56)	63,87	0,40	0,62	76,67*	0,35	0,80*

Há, pelo menos, três diferenças de resultados dos ciclos que merecem destaque: (1) a melhora da acurácia e da AUROC no ds_naotxt, de 86,65% para 91,10%, todavia o MAE aumentou, de 0,13 para 0,23; (2) o desempenho do modelo ds_comp piorou tanto em acurácia quanto em MAE, de 88,77% para 76,36% e de 0,12 para 0,40, respectivamente; e, por fim, (3) a AUROC melhorou nos modelos ds_naotxt-RF, ds_alg_comp-RF e ds_alg_planej-RF.

Os bons resultados, portanto, apresentados com a aplicação do *RandomForest* em um problema envolvendo predição e classificação de iterações de desenvolvimento de software corroboraram com o que é dito em [63] – o potencial desse algoritmo perante um especto abrangente de problemas.

Enfim, em termos de desempenho (Acurácia/MAE/AUROC) os modelos preditivos ds_alg_comp-J48 (92,75%/0,09/0,93) e ds_alg_comp-RF (93,38%/0,19/0,98) foram os melhores avaliados. O resultado do segundo modelo, inclusive, também corroborou com [51] e [5] – no sentido de utilizar dados históricos de iterações ou projetos para prever o resultado de uma iteração de software.

C. Limitações

Os dados utilizados foram informações de projetos reais de uma empresa de TIC, mas o número de projetos (ou *backlogs*) de desenvolvimento de software autorizado para uso foi menor que o esperado. Houve, então, uma redução na variedade de projetos algo que pode reduzir a representatividade dos resultados e/ou conclusões do estudo em questão. Essa redução no número de projetos/*backlogs* ocorreu por questões de segurança; e, pelo mesmo motivo e também pelo termo de compromisso assinado pelos pesquisadores junta a empresa parceira, os (conjuntos de) dados não foram disponibilizados ao público, entretanto o resultado do estudo em si – este artigo.

Os conjuntos de dados foram formados por dados que representam basicamente dois momentos da iteração/sprint, o planejamento ou confirmação e o resultado final da execução dela.

Os atributos utilizados para representar as iterações de desenvolvimento de software durante o processo de análise de dados foram os elementos disponíveis na base de dados da ferramenta da empresa mais complementação realizada por meio de questionamentos junto aos integrantes das equipes dos projetos/*backlogs* autorizados para uso dos dados. Esses atributos, entretanto, se limitaram basicamente a aspectos tanto do processo quanto do projeto de software, ou seja, não foram coletados dados relacionados, por exemplo, ao código do software desenvolvido pelas equipes dos projetos/*backlogs* selecionados e avaliados.

As técnicas de mineração de dados foram aplicadas a um número determinado de variações de conjuntos de dados considerando momentos do planejamento da iteração, e não "todas" as variações possíveis de composição desses conjuntos em termos de combinação das características ou atributos disponíveis.

Enfim, como não havia um atributo na base de dados de origem que informasse a situação final (classe) de cada iteração de desenvolvimento de software dos projetos/*backlogs* foi necessário derivar tal informação a partir do conteúdo de diferentes atributos do (sub)conjunto de dados Itens de trabalho – a quantidade de itens resolvidos versus total de itens da iteração de software considerando sucesso iterações que alcançaram uma resolução de itens igual ou superior a 90%.

V. CONSIDERAÇÕES FINAIS

O estudo realizado durante este trabalho teve como objetivo identificar modelos e padrões que possam auxiliar as equipes de software no processo de tomada de decisão inerente à realização de iterações ou sprints de desenvolvimento de software. Para alcançar esse objetivo, foi aplicado um método de estudo baseado no KDD-process. Esse método utilizou dados reais de desenvolvimento e as técnicas de mineração de dados Árvore de Decisão e Floresta Aleatória.

Os modelos com o melhor desempenho em termos de Acurácia, MAE e AUROC foram o ds_alg_comp-J48 (92,75%, 0,09 e 0,93) e ds_alg_comp-RF (93,38%, 0,19 e 0,98). Este estudo corroborou com o estudo [5] no sentido de demonstrar a possibilidade de predição do resultado de iterações de desenvolvimento de software se baseando em dados históricos do processo de software. E mais, o bom desempenho do segundo modelo corroborou com o fato de que o *RandomForest (RF)* tem bons resultados perante um espectro abrangente de problemas, algo descrito de certa forma em [63].

Conforme [12], nem sempre ter mais características ou atributos nos conjuntos de dados necessariamente resulta em maior poder de classificação ou discriminação por parte do modelo inferido, pelo contrário, adicionar elementos não relevantes ao conjunto de dados pode confundir o sistema de aprendizado de máquina. Os resultados obtidos neste trabalho demonstraram isso, veja o desempenho (acurácia) dos modelos com um número reduzido de características ou atributos (lembrando que o conjunto de dados completo/base tem 1943 atributos): ds_naotxt com 29 atributos 86,65% de acurácia com aplicação do J48 e 91,10% no RF, ds_alg_comp com 33 atributos 92,75% de acurácia no J48 e 93,38% no RF e, por fim, ds_alg_planej com 56 atributos 76,67% de acurácia no RF. Esses dois últimos conjuntos de dados foram gerados com o auxílio do algoritmo CFS. Eles obtiveram um desempenho melhor em termos de Acurácia e MAE que seus respectivos modelos com uma quantidade maior de atributos.

Perante o bom desempenho dos modelos preditivos, mesmo os que não consideraram dados textuais em suas estruturas, as equipes poderiam utilizá-los para realizar um "diagnóstico" da iteração durante ou ao final do seu planejamento e, dependendo do resultado, tomar uma ação buscando diminuir o risco de falha da iteração. Por exemplo, ao final do planejamento da iteração ou da confirmação dele a equipe informa os valores de entrada do modelo e avalia o resultado. Se positivo, então segue com a execução da iteração, caso contrário ela poderia variar alguns valores simulando diferentes cenários e com isso tomar alguma ação conforme os atributos utilizados nas simulações. Ressalta-se, entretanto, que dependendo do modelo escolhido a equipe terá que estimar alguns valores de atributos para poder utilizá-lo porque eles estão associados com o final da execução da iteração. Essa forma pode ser uma maneira de inserir previsibilidade em processos ou métodos de desenvolvimento software, tais como, Scrum e XP, os quais, segundo [4], priorizam mais flexibilidade e adaptabilidade.

Os atributos relevantes apresentados pelas estruturas de árvore de decisão dos melhores modelos gerados pela aplicação do algoritmo J48 se demostraram um padrão dados os aspectos importantes no processo de planejamento de software apontados em [4, p. 606–609] que são escopo, recursos e estimativas de software. Nesse cenário, destacou-se o atributo itr_its_div_ttl porque ele esteve presente em todas as estruturas dos melhores modelos, inclusive como o nó ou informação mais relevante. Ele está relacionado com o aspecto escopo, pois traz o total de itens de trabalho que foram divididos durante ou no final de uma determinada iteração/sprint. Observou-se, então, que iterações/sprints que não precisaram dividir nenhum item de trabalho durante ou ao final alcançaram o sucesso da iteração/sprint mais facilmente. Diante disso, conforme [44], a equipe deve realizar o *Product Backlog* refinement para ter itens menores e mais precisos.

Dada a premissa de desenvolvimento iterativo e incremental, ou seja, os requisitos não serão todos detalhados antes de iniciar o desenvolvimento, a equipe deve, portanto, manter o *backlog* do produto organizado de tal forma que há nele tanto itens que estão preparados para serem selecionados e desenvolvidos no curto prazo (as histórias de usuários pequenas) como itens que o horizonte é de médio e longo prazo (épicos) – o "iceberg" do *backlog* do produto [3]. A responsabilidade de manutenção dos itens do *backlog* do produto/iteração é do Dono do Produto – por exemplo, desenvolver e comunicar a meta/visão do produto, criar/refinar e ordenar itens [44]. Entretanto, pode ser necessário envolver (outros) especialistas, tais como, designers de experiência do usuário para, por exemplo, auxiliar na identificação do (real) problema e possíveis soluções e/ou na elaboração/refinamento dos requisitos [3].

Por outro lado, houve equipes que selecionaram consciente ou inconscientemente itens com um certo grau de incerteza, mas conseguiram lidar com isso, tanto que houve iterações que terminaram com sucesso mesmo com itens divididos durante ou no final da iteração. Nesse cenário, a equipe poderia, por exemplo, calcular a quantidade ou porcentagem média de itens divididos a partir de dados de iterações anteriores finalizadas com sucesso e utilizar isso no momento do planejamento da iteração corrente como um indicador de "grau médio/máximo de incerteza" que a equipe suporta – até porque, conforme [2], dada a natureza do produto em desenvolvimento, mudanças nos requisitos ocorrem e, segundo [39], são bem-vindas.

As considerações descritas até aqui podem fazer parte das reflexões e/ou discussões realizadas pelas equipes no início do projeto, antes do início do desenvolvimento em si do produto e/ou durante o evento de (Reunião de) Retrospectiva, no qual, segundo [44], os integrantes buscam a melhoria do processo de desenvolvimento do produto como um todo. Pois, melhorando o processo mais iterações ou sprints terminarão com sucesso, ou seja, mais software de qualidade entregue ao cliente no mesmo período de tempo – ou *timebox*. Para mais práticas técnicas ou de gestão no contexto de Desenvolvimento iterativo e incremental, Scrum e XP, consultar [3], [36], [37], [41], [66].

O processo para obter a autorização de uso de dados de uma empresa pode levar semanas ou até meses para ser deferido (ou indeferido), então, sugerimos que comece esse trâmite o quanto antes. E mais, saiba que todas as etapas do

processo de análise de dados são trabalhosas e complexas, mas as etapas iniciais (seleção, pré-processamento e transformação) podem demandar um esforço e/ou tempo razoável, principalmente se o tema em estudo e/ou a estrutura da(s) base(s) de dados selecionada não for de domínio de quem está conduzindo o respectivo processo. Por fim, como diz um provérbio chinês, "se quiser derrubar uma árvore na metade do tempo, passe o dobro do tempo amolando o machado", portanto, pesquise, selecione e prepare um ambiente (tecnologias, linguagens de programação e ferramentas) para a análise e mineração de dados antes de iniciar efetivamente o projeto ou processo de análise.

Por fim, apesar de conseguir alcançar o objetivo definido neste trabalho, alguns trabalhos futuros podem ser sugeridos. Um trabalho futuro poderia envolver um ou mais dos seguintes aspectos: dados de projetos/*backlogs* diferentes dos utilizados aqui; dados relacionados com o produto em si (métricas de código, por exemplo); variações dos parâmetros dos algoritmos J48 e *RandomForest*; uso de outras técnicas ou algoritmos de mineração dados, tal como, Redes Neurais voltado para indução de modelos preditivos e Regras de Associação com intuito de identificar práticas de desenvolvimento de software que possam melhorar o desempenho da equipe que planeja e executa as iterações de software.

REFERÊNCIAS

[1] R. S. Pressman, *Engenharia de software*, 6º ed. Porto Alegre (RS): AMGH, 2010.

[2] I. Sommerville, *Engenharia de software*, 8º ed. São Paulo: Pearson Addison-Wesley, 2007.

[3] M. Cohn, *Desenvolvimento de software com scrum: aplicando métodos ágeis com sucesso*. Porto Alegre: Bookman, 2011.

[4] R. S. Pressman, *Engenharia de software: uma abordagem profissional*, 7º ed. Porto Alegre (RS): AMGH, 2011.

[5] M. Choetkiertikul, H. K. Dam, T. Tran, A. Ghose, e J. Grundy, "Predicting Delivery Capability in Iterative Software Development", *IEEE Transactions on Software Engineering*, vol. 44, nº 6, p. 551–573, jun. 2018, doi: 10.1109/TSE.2017.2693989.

[6] V. T. Faria, J. F. B. Facó, e A. A. de Andrade, "Mineração de Dados para Análise de Bancos de Dados Empresariais", p. 6, 2017.

[7] F. Provost e T. Fawcett, *Data Science para Negócios*. Rio de Janeiro, Brasil: Alta Books, 2016.

[8] A. C. Beltrão, "Aplicação de Software Analytics e o Algoritmo de Ranqueamento de Páginas Para Apoiar Decisões de Planejamento de Sprint", Monografia, Faculdade UnB Gama, Brasília/DF, 2016.

[9] K. V. de Moura, "Data Science : um estudo dos métodos no mercado e na academia", 2018, Acessado: dez. 14, 2020. [Online]. Disponível em: https://lume.ufrgs.br/handle/10183/195011.

[10] U. Fayyad, G. Piatetsky-Shapiro, e P. Smyth, "From Data Mining to Knowledge Discovery in Databases", *AIMag*, vol. 17, nº 3, Art. nº 3, mar. 1996, doi: 10.1609/aimag.v17i3.1230.

[11] J. Han, J. Pei, M. Kamber, e an O. M. C. Safari, *Data Mining: Concepts and Techniques, 3rd Edition*, 3º ed. Amsterdam et al: Morgan Kaufmann Publishers, 2011.

[12] I. H. Witten, E. Frank, M. A. Hall, e C. J. Pal, *Data mining: practical machine learning tools and techniques*, Fourth Edition. Amsterdam: Elsevier, 2017.

[13] M. J. Bianchi e D. C. Amaral, "O potencial do Data Mining para tratamento da complexidade no contexto do Gerenciamento Híbrido de projetos de novos produtos", in *Blucher Engineering Proceedings*, set. 2019, vol. 2, nº 6, p. 83–93, Acessado: jan. 12, 2021. [Online]. Disponível em: https://www.proceedings.blucher.com.br/article-details/o-potencial-do-data-mining-para-tratamento-da-complexidade-no-contexto-do-gerenciamento-hbrido-de-projetos-de-novos-produtos-33540.

[14] J. L. R. de Almeida, R. Balancieri, M. N. Roecker, G. C. L. Leal, e P. H. Bermejo, "Extração de Conhecimento a partir de Regras de Associação entre Métricas de Código Fonte", *Revista de Sistemas e Computação - RSC*, vol. 7, nº 1, ago. 2017, Acessado: jan. 12, 2021. [Online]. Disponível em: https://revistas.unifacs.br/index.php/rsc/article/view/4717.

[15] H. Brink, J. W. Richards, e M. Fetherolf, *Real-world machine learning*. Shelter Island: Manning, 2017.

[16] T. Beysolow II, *Applied Natural Language Processing with Python: Implementing Machine Learning and Deep Learning Algorithms for Natural Language Processing*. Berkeley, CA: Apress, 2018.

[17] WEKA, "WEKA General documentation", *WEKA General documentation*, 2021. https://waikato.github.io/weka-wiki/documentation/#general-documentation (acessado jan. 10, 2021).

[18] M. A. Hall, "Correlation-based Feature Selection for Machine Learning", Doutorado, University of Waikato, Hamilton - Nova Zelândia, 1998.

[19] L. Breiman, "Random Forests", *Machine learning*, vol. 45, nº 1, p. 5–32, 2001.

[20] J. R. Quinlan, *C4.5: Programs for Machine Learning*. 1993.

[21] K. J. Cios, W. Pedrycz, R. W. Swiniarski, e L. A. Kurgan, Orgs., *Data mining: a knowledge discovery approach*. New York, NY: Springer, 2007.

[22] G. de O. Santos, "Benchmarking em algoritmos de classificação na mineração de dados", *Benchmarking in mining classification algorithms of data*, jul. 2018, Acessado: dez. 28, 2020. [Online]. Disponível em: http://rosario.ufma.br:8080/jspui/handle/123456789/3500.

[23] S. Dragicevic, S. Celar, e M. Turic, "Bayesian network model for task effort estimation in agile software development", *Journal of Systems and Software*, vol. 127, p. 109–119, maio 2017, doi: 10.1016/j.jss.2017.01.027.

[24] C. O. Camilo, "Mineração de Dados: Conceitos, Tarefas, Métodos e Ferramentas", p. 29, 2009.

[25] H. M. Carvalho, "Aprendizado de Máquina voltado para Mineração de Dados: Árvores de Decisão", Trabalho de Conclusão de Curso, Universidade de Brasília - UnB, Brasília, DF, 2014.

[26] H. M. Justino, L. G. D. O. MORIMOTO, e L. GOBBATO, "Aleteia : sistema de classificação de notícias", Trabalho de Conclusão de Curso, Universidade Federal do Paraná, Curitiba, 2019.

[27] V. A. Padilha e A. C. P. de L. F. de Carvalho, *Mineração de Dados em Python*. São Paulo, 2017.

[28] Python, "Python", *Welcome to Python.org*, 2021. https://www.python.org/ (acessado mar. 06, 2021).

[29] Python, "Python (Brasil)", 2021. https://python.org.br/ (acessado mar. 06, 2021).

[30] S. Raschka e V. Mirjalili, *Python machine learning: machine learning and deep learning with Python, scikit-learn, and TensorFlow*, 2º ed. Birmingham Mumbai: Packt Publishing, 2017.

[31] NLTK, "Natural Language Toolkit", 2021. http://www.nltk.org/ (acessado mar. 06, 2021).

[32] scikit-learn, "Getting Started — scikit-learn documentation", 2021. https://scikit-learn.org/stable/getting_started.html (acessado mar. 06, 2021).

[33] Spyder, "Spyder IDE", *Home — Spyder IDE*, 2021. https://www.spyder-ide.org/ (acessado mar. 06, 2021).

[34] R. F. V. de Castro, "Análise de desempenho dos algoritmos Apriori e Fuzzy Apriori na extração de regras de associação aplicados a um Sistema de Detecção de Intrusos", Dissertação, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2014.

[35] WEKA, "WEKA The workbench for machine learning", *WEKA The workbench for machine learning*, 2021. https://www.cs.waikato.ac.nz/ml/weka/ (acessado jan. 10, 2021).

[36] K. Beck e C. Andres, *Extreme programming explained: embrace change*, 2nd ed. Boston, MA: Addison-Wesley, 2005.

[37] H. Kniberg, M. Cohn, e J. Sutherland, *Scrum and XP from the Trenches: how we do Scrum*. C4Media, 2015.

[38] R. Prikladnicki, R. Willi, e F. Milani, *Métodos Ágeis para Desenvolvimento de Software*. Porto Alegre: Bookman, 2014.

[39] K. Beck *et al.*, "Manifesto for Agile Software Development", 2001. https://agilemanifesto.org/ (acessado mar. 14, 2021).

[40] T. Dyba e T. Dingsoyr, "What Do We Know about Agile Software Development?", *IEEE Software*, vol. 26, nº 5, p. 6–9, set. 2009, doi: 10.1109/MS.2009.145.

[41] J. Shore e S. Warden, *A Arte do Desenvolvimento Ágil*. Rio de Janeiro: Alta Books, 2008.

[42] Digital.ai, "State of Agile Survey", 2019. https://stateofagile.com/ (acessado mar. 12, 2021).

[43] M. Fowler, "The New Methodology", *martinfowler.com*, 2005. https://martinfowler.com/articles/newMethodology.html (acessado mar. 12, 2021).

[44] K. Schwaber e J. Sutherland, *O Guia do Scrum*. 2020.

[45] Oracle Corporation, "MySQL :: MySQL 8.0", *Reference Manual :: 1.2 Overview of the MySQL Database Management System*, 2021. https://dev.mysql.com/doc/refman/8.0/en/what-is.html (acessado jan. 14, 2021).

[46] A. Silberschatz, H. F. Korth, e S. Sudarshan, *Sistema de bancos de dados*, 3º ed. São Paulo: Makron Books, 1999.

[47] R. Elmasri, S. B. Navathe, M. G. Pinheiro, e L. R. de Figueiredo, *Sistemas de banco de dados*. São Paulo: Pearson Addison Wesley, 2005.

[48] Oracle Corporation, "MySQL :: MySQL Workbench", 2021. https://www.mysql.com/products/workbench/ (acessado jan. 16, 2021).

[49] C. A. de S. Junior e H. F. Villela, "Um estudo sobre Publicações Relacionadas a Mineração de Dados", *Computação & Sociedade*, vol. 1, nº 1, Art. nº 1, set. 2019, Acessado: dez. 13, 2020. [Online]. Disponível em: http://www.fumec.br/revistas/computacaoesociedade/article/view/7301.

[50] M. Choetkiertikul, H. K. Dam, T. Tran, e A. Ghose, "Characterization and Prediction of Issue-Related Risks in Software Projects", in *IEEE/ACM 12th Working Conference on Mining Software Repositories*, maio 2015, p. 280–291, doi: 10.1109/MSR.2015.33.

[51] A. L. C. Nascimento, "Mineração de dados aplicada ao controle de prazos e prioridades em projetos de software", Monografia, Universidade Tecnológica Federal do Paraná, 2017.

[52] E. Turban, R. K. Rainer, e R. E. Potter, *Introdução a sistemas de informação uma abordagem gerencial*, 6º ed. Rio de Janeiro: Campus, 2007.

[53] S. Russell e P. Norvig, *Inteligência artificial*. Rio de Janeiro, Brazil: Elsevier, 2013.

[54] R. Agrawal e R. Srikant, "Fast Algorithms for Mining Association Rules", Santiago, Chile, 1994, p. 478–499, [Online]. Disponível em: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.2474&rep=rep1&type=pdf.

[55] R. A. F. de Lima, "Estratégias de seleção de atributos para detecção de anomalias em transações eletrônicas", Dissertação, Universidade Federal de Minas Gerais - Departamento de Ciência da Computação, Belo Horizonte, 2016.

[56] S. Goswami e A. Chakrabarti, "Feature Selection: A Practitioner View", *IJITCS*, vol. 6, nº 11, p. 66–77, out. 2014, doi: 10.5815/ijitcs.2014.11.10.

[57] T. Fawcett, "ROC graphs: Notes and practical considerations for researchers", 2003.

[58] J. Brownlee, "Do Not Use Random Guessing As Your Baseline Classifier", *Machine Learning Mastery*, mar. 03, 2016. https://machinelearningmastery.com/dont-use-random-guessing-as-your-baseline-classifier/ (acessado nov. 30, 2020).

[59] J. Granatyr, "Classificador ZeroR no Weka", *IA Expert Academy*, 2016. https://iaexpert.academy/2016/12/29/classificador-zeror-no-weka/ (acessado jan. 08, 2021).

[60] D. R. Carvalho e M. R. Dallagassa, "Mineração de dados: aplicações, ferramentas, tipos de aprendizado e outros subtemas", *AtoZ: novas práticas em informação e conhecimento*, vol. 3, nº 2, Art. nº 2, dez. 2014, doi: 10.5380/atoz.v3i2.41340.

[61] V. C. M. Lima, A. G. S. Seca Neto, e M. C. F. P. Emer, "Experiência bem-sucedida de adoção de Métodos Ágeis em uma Empresa Pública de Tecnologia da Informação e Comunicação: um relato preliminar", apresentado em Workshop Brasileiro de Métodos Ágeis, São Paulo, 2013, [Online]. Disponível em: http://www2.dainf.ct.utfpr.edu.br/Members/adolfo/pesquisa/agile-methods.

[62] B. N. B. Paixão, L. B. da Silva, M. V. das N. Toffano, H. R. M. da Hora, e R. A. de Carvalho, "Descoberta de Conhecimento em Base de Dados de Projetos de Inovação: Um Estudo em um Polo Embrapi", in *Simpósio de Pesquisa Operacional e Logística da Marinha - Publicação Online*, Rio de Janeiro, Brasil, maio 2020, p. 1047–1057, doi: 10.5151/spolm2019-077.

[63] S. Lessmann, B. Baesens, C. Mues, e S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings", *IIEEE Trans. Software Eng.*, vol. 34, nº 4, p. 485–496, jul. 2008, doi: 10.1109/TSE.2008.35.

[64] D. Rumsey, *Estatística Para Leigos*. Rio de Janeiro, Brasil: Alta Books, 2010.

[65] V. C. G. Coelho, J. P. C. L. da Costa, D. A. da Silva, R. T. de Sousa Jr., F. L. L. de Mendonça, e D. G. Silva, "Mineração de Dados Educacionais no Ensino à Distância Governamental", apresentado em Conferências Ibero-Americanas WWW/Internet e Computação Aplicada, 2016.

[66] V. C. M. Lima, "Programação em par: investigando sua eficácia perante tarefas de modelagem e construção de software", Dissertação, Universidade Tecnológica Federal do Paraná, Curitiba, 2013.