UNIVERSIDADE FEEVALE

LUCAS ADAMS CAMARGO

DESCOBERTA DE CONHECIMENTO EM BASE DE DADOS DE ALTERAÇÕES EM SISTEMA DE GESTÃO

Novo Hamburgo

2019

LUCAS ADAMS CAMARGO

DESCOBERTA DE CONHECIMENTO EM BASE DE DADOS DE ALTERAÇÕES EM SISTEMA DE GESTÃO

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação pela Universidade Feevale

Orientador: Gabriel da Silva Simões

Novo Hamburgo

2019

LUCAS ADAMS CAMARGO

DESCOBERTA DE CONHECIMENTO EM BASE DE DADOS DE ALTERAÇÕES EM SISTEMA DE GESTÃO

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação pela Universidade Feevale

APROVADO EM: ___ / ___ / _____

GABRIEL DA SILVA SIMÕES
Orientador – Feevale

EDVAR BERGMANN ARAUJO
Examinador interno – Feevale

RODRIGO RAFAEL VILLARREAL GOULART
Examinador interno – Feevale

Novo Hamburgo
2019

AGRADECIMENTOS

Gostaria de agradecer a todos os que, de alguma maneira, contribuíram para a realização desse trabalho de conclusão, em especial: À minha família, sempre presente provendo apoio e incentivo. Ao meu orientador Gabriel da Silva Simões que compartilhou seu conhecimento e sua dedicação para enriquecimento deste trabalho. A todos vocês minha mais sincera gratidão!

RESUMO

No processo de desenvolvimento de software frequentemente são inseridas inconsistências no sistema, o que acaba gerando diversos transtornos. Estes defeitos no sistema impactam tanto a empresa, que precisa investir tempo em correções, quanto o cliente, que não consegue utilizar o sistema apropriadamente. A empresa Rech Informática possui todas as alterações em seu sistema ERP documentadas de forma que é possível identificar se a modificação gerou alguma inconsistência ou não. Entretanto, não é possível analisar manualmente todos esses dados para extrair informações relevantes sobre essas inconsistências e evitar que elas ocorram. Dessa forma, esse trabalho trata da aplicação de técnicas de mineração de dados e aprendizado de máquina sobre essa base de dados objetivando a geração de um modelo capaz de prever se determinada alteração no sistema irá gerar inconsistência. O processo de treinamento de modelos preditivos foi desenvolvido utilizando a linguagem de programação Python, que possui bibliotecas voltadas para a Ciência de Dados. Os resultados obtidos confirmam a viabilidade da geração de predições, porém com ressalvas quanto à acurácia obtida. Além disso, foi observada a praticidade e eficiência da utilização da linguagem Python para fins de geração de conhecimento.

Palavras-chave: Inconsistências. Mineração de dados. Aprendizado de máquina. Modelo preditivo. Aprendizado supervisionado.

ABSTRACT

In software development process often defects are inserted in the system, which leads to generating several inconveniences. These system defects affect the business, which needs to invest time in patches, and the customer, who can not use the system properly. The company Rech Informática has all the changes in its ERP system documented, this makes it possible to identify if the change has caused some inconsistency or not. However, it is not viable to manually analyze all data to extract relevant information about these inconsistencies and prevent them from occurring. Thus, this work is related to the application of techniques of data mining and machine learning on this database aiming at the generation of models capable of predicting if certain change in the system will generate some defect. The predictive modeling training process will be developed using the Python programming language, which has libraries focused on Data Science.

Keywords: Defect. Data Mining. Machine learning. Predictive model. Supervised learning

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CSV	*Comma-Separated Values*
ERP	*Enterprise Resourse Planning*
KNN	*K-Nearest Neighbors*
MLP	*Multilayer Perceptron*
RNA	Rede Neural Artificial
RNC	Registro de Não Conformidade
RNS	Registro de Necessidade e Sugestão
SQL	*Structured Query Language*
SVC	*C-Support Vector Classification*
SVM	*Support Vector Machine*
SIBC	Sistema de Informação Baseado em Computador
SIGER	Sistema Integrado de Gestão Empresarial Rech

SUMÁRIO

1 INTRODUÇÃO

Cada vez mais a tecnologia faz parte do cotidiano, o que inclui o dia-a-dia das empresas, que frequentemente sustentam suas atividades utilizando sistemas de gestão para controlar o negócio. Esses sistemas são conhecidos pela sigla em inglês ERP (*Enterprise Resourse Planning*), que se refere a um software de Planejamento de Recursos Empresariais. Segundo Davenport (2002), sistemas ERP podem ser definidos como aplicativos separados em pacotes que dão apoio às decisões da empresa fornecendo informações em um sistema integrado.

Uma das empresas que atua na área de desenvolvimento de sistema ERP é a Rech Informática, que está alocada na cidade Novo Hamburgo no estado do Rio Grande do Sul. O principal software desenvolvido pela empresa é o SIGER® (Sistema Integrado de Gestão Empresarial Rech), que atende um total de aproximadamente 11.000 usuários (Rech Informática, 2019a). Conforme Haberkorn (1999) propõe, este tipo de empresa possui alguns processos centrais, como levantamento de requisitos e validação por exemplo, porém se destaca a atividade de desenvolvimento de software. No cenário da Rech Informática, todas as alterações no sistema estão registradas em uma ferramenta interna de controle chamada Sicla. Essa ferramenta consolida informações sobre atendimentos telefônicos realizados, compromissos em agendas dos técnicos, além do próprio registro de alterações no sistema ERP, que será alvo de estudo neste trabalho.

A base de dados do Sicla possui diversos dados sobre as alterações realizadas no sistema, e entre essas informações é possível verificar quais alterações levaram ao surgimento de inconsistências no sistema. Na realidade da empresa esses dados são aproveitados apenas para fins estatísticos de monitoramento, de forma que se tem o controle, por exemplo, se a quantidade total de horas investidas em correção de defeitos aumentou ou diminuiu em relação à anos anteriores. Sendo assim, existe um grande volume de informações relativas ao processo de desenvolvimento de software que poderiam ser aproveitadas para gerar conhecimento sobre essa atividade e, assim, possibilitar que ela ocorra de forma mais eficiente. Além disso, essas informações podem ser utilizadas não somente para monitorar o passado, mas também para prever futuras ocorrências de inconsistências.

Para solucionar esse tipo de situação que existe a área nomeada Descoberta de Conhecimento em Base de Dados ou, na sigla em inglês, KDD (*Knowledge Discovery in Databases*). Uma definição para este termo elaborada em 1996 é a seguinte: KDD é um processo, de várias etapas, não trivial, interativo e iterativo, para identificação de padrões compreensíveis, válidos, novos e potencialmente úteis a partir de grandes conjuntos de dados (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996 apud GOLDSCHMIDT; PASSOS, 2005).

Como salientado anteriormente, o processo de descoberta de conhecimento é composto por várias etapas e, segundo Goldschmidt e Passos (2005), destacam-se três principais: pré-processamento, mineração de dados e pós-processamento. O pré-processamento abrange as operações de obtenção e preparação dos dados, enquanto que a etapa de mineração se refere à etapa na qual são obtidos os conhecimentos acerca da base de dados. A última etapa, o pós-processamento, objetiva avaliar a utilidade dos conhecimentos obtidos, buscando encontrar aplicações para os mesmos.

A descoberta de conhecimento em bases de dados de empresas desenvolvedoras de software já foi alvo de estudo em algumas ocasiões. Pode ser citado o estudo realizado por Cruz e Ruiz (2008), no qual são analisados os apontamentos de atividades registradas durante a manutenção de software na empresa desenvolvedora. Esses apontamentos fornecem dados para a mineração de processos, que gera informações sobre como as atividades estão sequenciadas e que dependências existem entre elas. Uma questão relevante apontada por esse trabalho é a atenção para a distribuição dos dados disponíveis para análise, pois no cenário estudado existiam poucas informações sobre o processo de execução de projetos, uma vez que são demandas que ocorrem com menos frequência. Da mesma forma, na base de dados de alterações no SIGER® existe um pequeno número de alterações que geram inconsistências se comparado à quantidade total de alterações feitas.

Porém, além de extrair conhecimento da base de dados, também é possível gerar modelos capazes de prever o comportamento de novos cenários baseado nos dados antecessores através de técnicas de *machine learning*, que é traduzido para aprendizado de máquina. Como citado por Mitchell (2006), enquanto a Ciência da Computação focava inicialmente em programar manualmente computadores, o aprendizado de máquina é focado em como é possível fazer os computadores programarem a si próprios. Ou seja, esta área propõe técnicas de aprendizado que utilizam a experiência entregue à aplicação para gerar performance na execução de uma tarefa proposta (MITCHELL, 2006).

Sendo assim, o objetivo deste trabalho é analisar a base de dados de alterações realizadas para manutenção de software utilizando técnicas de *machine learning* visando gerar um modelo capaz de prever a ocorrência de inconformidades no sistema. Para isso, é necessário obter da base de dados as informações que serão analisadas, submetendo as mesmas à preparação necessária para utilização no processo de mineração de dados. A exploração dos dados foi realizada utilizando recursos da linguagem de programação Python e, finalmente, foram estudados os resultados obtidos buscando validar a acurácia das regras e conhecimentos encontrados.

Este trabalho está organizado da seguinte maneira: o Capítulo 2 estabelece o cenário que está sendo estudado. No Capítulo 3 são descritos os conceitos e técnicas relativos ao aprendizado de máquina. As formas de avaliação dos resultados são discutidas no se-

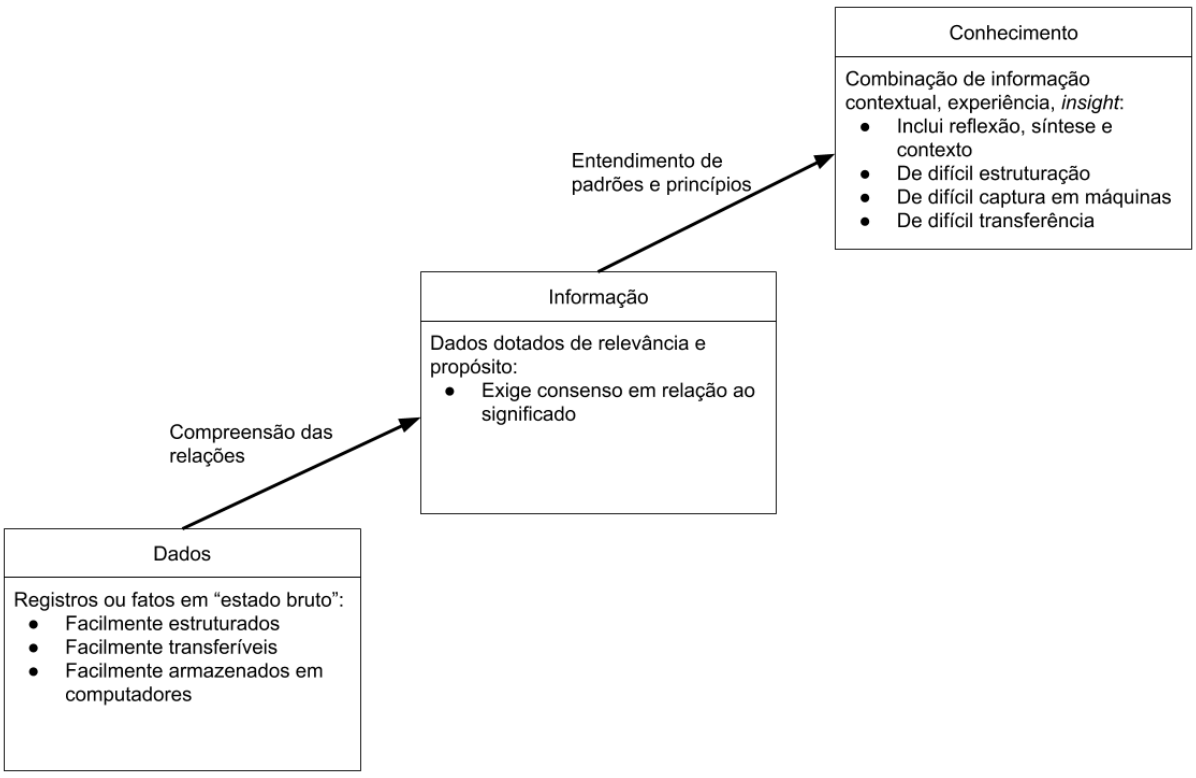
ção 3.4. No Capítulo 4 são apresentadas as atividades exploratórias realizadas com base na fundamentação teórica, assim como os primeiros resultados. O Capítulo 5 relata todas as implementações acerca da extração de dados e geração de modelos preditivos realizadas após a etapa de exploração inicial, assim como os resultados obtidos. Por fim, o Capítulo 6 relata as principais conclusões do trabalho.

2 SISTEMAS DE GESTÃO

A tecnologia no século XXI abriu várias possibilidades que eram remotas até décadas atrás. Um dos fenômenos que pode ser observado com o crescimento da tecnologia foi o aumento na capacidade dos consumidores localizarem ofertas distintas do produto que se tem interesse, obtendo uma visão clara das condições propostas por diferentes organizações. Sendo assim, o advento da tecnologia no cotidiano das pessoas acirrou a concorrência entre empresas de diversos segmentos, fazendo com que essas empresas buscassem a criação de vantagens competitivas que pudessem destacar a organização entre as demais.

Para buscar essas vantagens é necessário que os gestores tenham uma visão objetiva do negócio e das informações relativas à ele. Como Beal (2004) destaca, a informação de qualidade possibilita que a tomada de decisão seja feita de forma mais acertada, diminuindo os riscos e trazendo mais resultados para a empresa. No entanto, para compreender mais claramente essa afirmação é importante conhecer a diferença entre dados, informações e conhecimentos.

Figura 1: Níveis hierárquicos da informação

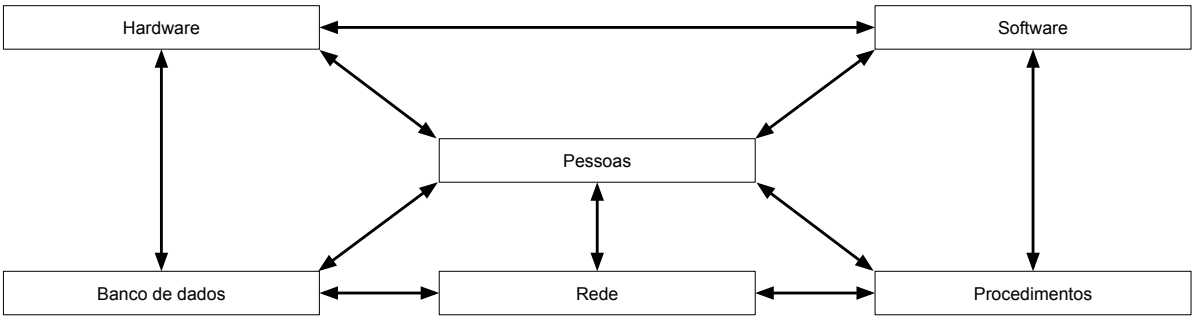


Fonte: Adaptado pelo autor segundo Beal (2004)

Conforme pode ser observado na Figura 1, dados são fatos em sua forma primária, podendo ser físicos, como textos escritos, ou até abstratos, como recordações armazenadas na memória. Ao serem providos de significado e contexto esses dados podem ser classificados como informação. Sendo assim, dados são a fonte para geração de informações. O conhecimento por sua vez também é gerado a partir de informações, porém não somente delas. Para se ter conhecimento também é necessário experiência e reflexão, de forma que seja possível a avaliação e incorporação de novas informações acerca do conhecimento. É válido observar também que obter um dado não necessariamente leva à obtenção de uma informação, assim como uma informação, por mais precisa que seja, não produz necessariamente um conhecimento.

No contexto das empresas é possível observar que a correta gestão e aproveitamento das informações reflete uma vantagem competitiva sobre as demais organizações que desconhecem o próprio negócio. Para promover esse controle existem os chamados Sistemas de Informação, que se caracterizam por manipular dados objetivando gerar informações úteis para os usuários (BEAL, 2004). Segundo Turban, Jr e Potter (2003) os Sistemas de Informação podem ser manuais, porém a maior parte dos mesmos se aproveita de recursos computacionais para funcionar. Sendo assim, esse tipo de sistema pode ser classificado ainda como um Sistema de Informação Baseado em Computador (SIBC), que pode ser segmentado em algumas partes principais destacadas na Figura 2.

Figura 2: Componentes básicos de um SIBC



Fonte: Adaptado pelo autor segundo Turban, Jr e Potter (2003)

Como pode ser observado na Figura 2, os SIBC promovem um conjunto de interrelações entre os componentes visando a geração e controle de informações de interesse das pessoas. O componente *Hardware* se refere aos aparatos físicos utilizados para interagir com o sistema, tais quais monitores, teclados, processador, etc. *Software* engloba os programas de computador que executam efetivamente o processamento dos dados. Os dados do sistema ficam armazenados no Banco de dados, que pode ser descrito como um conjunto de arquivos organizados e relacionados através de associações. Já a rede se refere à estrutura que possibilita o compartilhamento de recursos entre diferentes computadores.

Por fim, os procedimentos refletem as regras, estratégias ou métodos para utilização do sistema de informação.

2.1 SISTEMAS ERP

Uma empresa pode aproveitar diferentes Sistemas de Informação baseados em computador para alavancar o negócio, cobrindo as diversas áreas do negócio. Com essa estratégia teria-se, por exemplo, um sistema de gestão específico para a produção sendo utilizado paralelamente a outro sistema de gestão que efetua o controle financeiro da empresa. Porém, ao fazer isso a tendência é que ocorra o desencontro de informações e o retrabalho com a digitação dos mesmos dados nos vários sistemas diferentes que são utilizados no negócio. Para solucionar esse tipo de problema surgiu o conceito de Sistema de Gestão Integrada, também conhecido pela sigla em inglês ERP (*Enterprise Resource Planning*).

Segundo Leon (2014), um ERP trata-se de um sistema que apoia a gestão do negócio como um todo, promovendo a eficiência no uso dos recursos da empresa e organização nas operações cotidianas. Tipicamente esse sistema é segmentado em diferentes pacotes, cada um atendendo um determinado setor da empresa, mas seguindo sempre os princípios de apoio à gestão. Esses módulos são focados nas operações essenciais executadas pela empresa, como controle de vendas, controle financeiro e contabilidade por exemplo. Como os sistemas ERP surgiram inicialmente voltados para as industrias de manufatura, também é comum que eles ofereçam funcionalidade para controle de produção e insumos.

Apesar desta divisão do sistema, uma característica sempre presente é a capacidade de integração dos dados, de forma que não ocorram informações desencontradas ou faltantes dentro do sistema. Um exemplo disso pode ser uma rotina de emissão de notas do módulo de vendas que automaticamente atualiza informações de saldo no módulo de estoque e gera pagamentos no módulo financeiro. Com isso, uma ação dentro do sistema em um determinado módulo pode gerar informações em outros módulos, garantindo que os dados do processo estejam integrados no contexto do sistema.

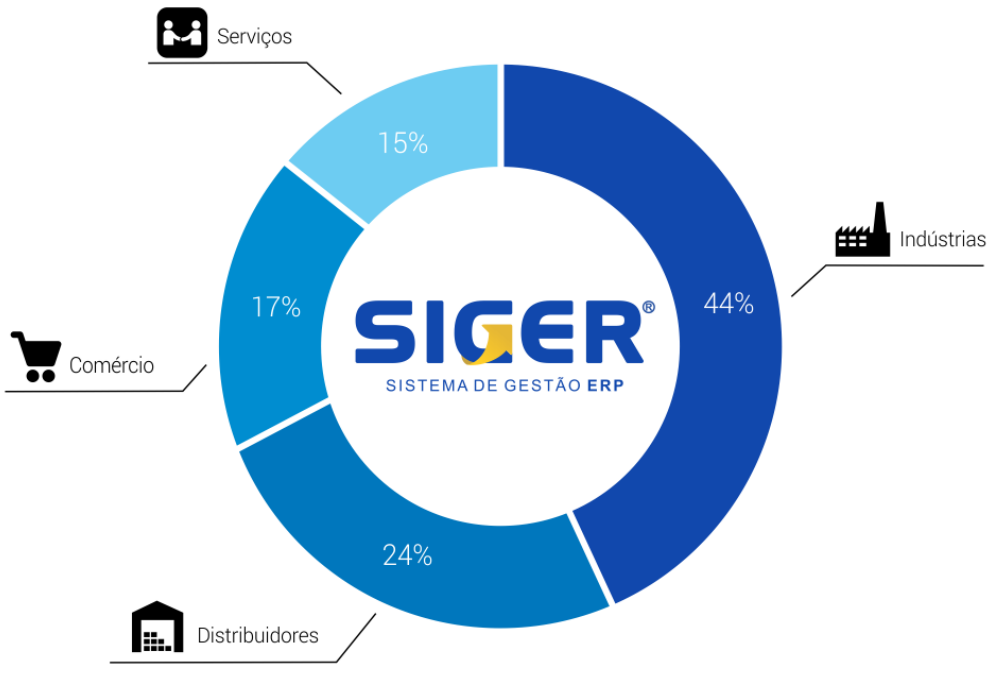
2.2 RECH INFORMÁTICA

A criação de um sistema ERP é uma tarefa trabalhosa, exigindo tanto conhecimento sobre recursos tecnológicos quanto conhecimentos de gestão, como regras de negócio. Porém, o desenvolvimento continua mesmo após o sistema estar concluído. Segundo Leon (2014) a customização é uma necessidade recorrente na implantação de sistemas ERP, haja vista que muitas vezes o processo seguido pela empresa cliente não se adapta totalmente aos recursos disponíveis no sistema. A Rech informática é uma empresa desenvolvedora de software, responsável pela manutenção e suporte do sistema ERP SIGER®.

A empresa está sediada em Novo Hamburgo, cidade do Rio Grande do Sul e atua na área de software de gestão desde o início da década de 90. Foi fundada pelos irmãos Carlos Vanderlei Rech e Rovani Marcelo Rech com o objetivo de desenvolver um software completo e integrado que eliminasse os retrabalhos e desperdícios no processo das empresas, alinhando conhecimentos de gestão empresarial e tecnologias produtivas. A missão da empresa constitui-se em fornecer produtos e serviços de alta qualidade contribuindo para que clientes e colaboradores atinjam seus objetivos estratégicos, explorando ao máximos seus potenciais e otimizando seus processos através da integração de tecnologia e sistema de informações.

Atualmente a empresa conta com uma equipe de 150 colaboradores distribuídos nos setores de desenvolvimento, suporte, manutenção e comercial. A empresa atende mais de 1.200 clientes e 11.000 usuários que utilizam o sistema ERP diariamente em seu negócio. Estes clientes concentram-se na região do Vale do Sinos e região metropolitana de Porto Alegre. São atendidos diversos ramos de negócio, como indústrias e comércio. A Figura 3 representa a distribuição atual de clientes da Rech Informática conforme essas áreas de negócio.

Figura 3: Clientes por segmento de atuação



Fonte: (Rech Informática, 2019b)

2.3 SICLA

No desenvolvimento de alterações na empresa Rech Informática são executadas várias operações internas que geram dados ao longo de todo o processo. O controle interno das demandas e atividades executadas pelo colaboradores se dá por meio de uma ferramenta chamada Sicla, que foi desenvolvida especificamente para esse fim pela própria empresa. Este programa está presente ao longo de todo o processo da empresa, armazenando contatos realizados, a posição das demandas atuais e gerando apontamentos de produção. Além disso, a ferramenta ainda possui recursos adicionais como agenda, geração de recados e painéis de monitoramento. A seguir é discutido o processo de implementação de alterações no sistema ERP assim como o uso da ferramenta Sicla e que tipo de informações é possível obter a partir dela.

A demanda da empresa inicia tipicamente através de um contato do cliente, que pode ser por meio de um telefonema, uma conversa de skype ou troca de e-mail. O técnico do suporte então registra este contato no Sicla através de uma entidade chamada atendimento. Neste registro são inseridos dados como data e hora do contato, empresa envolvida, usuário do sistema, motivo do contato e uma descrição textual dos assuntos tratados.

O atendimento pode se tratar de uma dúvida pontual que é esclarecida na hora, mas também pode se tratar de um contato para solicitação de uma alteração no sistema, que será encaminhada para o setor de desenvolvimento. Também podem ocorrer contatos que reportam defeitos no sistema e, neste caso, também haverá envolvimento do grupo de desenvolvimento da empresa. Para registrar e controlar essas demandas de alteração no sistema ERP dentro do Sicla existe o Registro de Necessidades e Sugestões, chamado de RNS. Neste registro ficam documentadas informações semelhantes ao atendimento, como criador, data e hora da necessidade, empresa e usuário que gerou a demanda e descrições textuais sobre o cenário do cliente e a alteração que está sendo solicitada. Além destes dados, a RNS também conta com informações sobre o tempo decorrido para levantamento de requisitos, ou seja, esclarecendo detalhes sobre a alteração e expectativas do cliente.

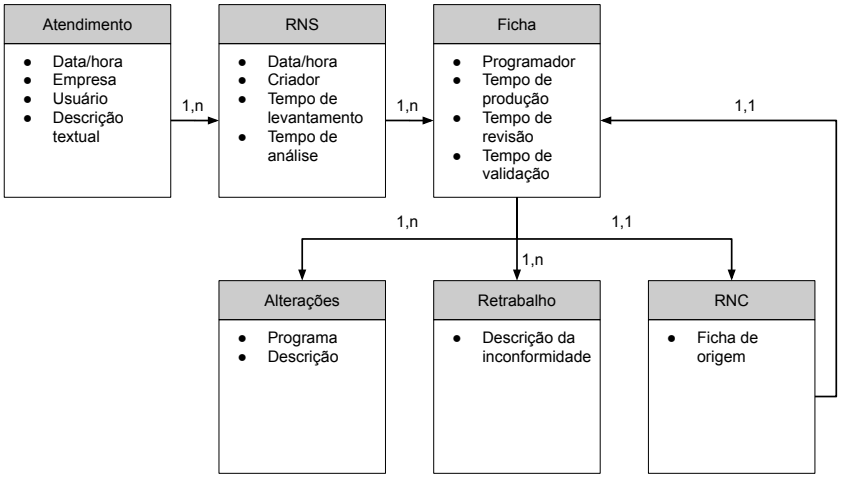
Antes de fazer efetivamente a alteração no sistema a RNS passa por um processo de análise, no qual um desenvolvedor estuda a alteração solicitada e propõe formas de atender ela. Neste processo são geradas mais informações no registro da RNS, como o analista responsável e total de tempo de análise investida. Quando a análise está concluída é gerado outro registro dentro do Sicla, a ficha, que registra detalhes da alteração como o código fonte que será alterado e código do programador que executará a alteração. Na ficha são feitos novos apontamentos de produção que geram um tempo total de implementação da alteração, além do registro de fontes alterados, que fica armazenado em uma tabela de alterações.

Com o término da produção se inicia outro processo, de revisão, que é feito por outro desenvolvedor a fim de identificar se a alteração foi feita seguindo os padrões. Após a revisão tem-se a validação, que verifica se a alteração atende adequadamente a necessidade apresentada pela RNS. Esses processos também geram apontamentos de tempo de revisão e validação vinculados à ficha. Havendo alguma inconformidade é feito um registro de retrabalho dentro da ficha apontando o que precisa ser mudado. Após o retrabalho ser sanado, ocorre novamente a revisão e validação, incrementando o tempo total destas atividades.

No caso de alterações que se tratam de uma correção de uma inconsistência do sistema a ficha ainda conta com um registro adicional chamado de RNC, que significa Registro de Não Conformidade. Esse registro documenta qual foi a ficha responsável por inserir o defeito no sistema. Portanto, a partir dessa documentação é possível identificar quando uma alteração no sistema gerou uma inconsistência. As informações do Sicla estão disponíveis em um banco de dados interno da empresa, no qual é possível fazer consultas para elaborar uma base de dados de alterações no sistema, contendo dados de interesse para o aprendizado de máquina que será estudado neste trabalho.

A Figura 4 ilustra o relacionamento entre as entidades descritas no sistema Sicla. Estes dados, até o presente momento, são utilizados para fins de monitoramento e estatística, avaliando-se periodicamente a quantidade de atendimentos realizados e de RNS atendidas, por exemplo. Este trabalho propõe a utilização destes dados de forma preventiva, visando detectar quando uma alteração insere um defeito no sistema.

Figura 4: Diagrama entidade relacionamento dos dados do Sicla



Fonte: Elaborado pelo autor

3 APRENDIZADO DE MÁQUINA

Tipicamente na computação as tarefas são resolvidas desenvolvendo programas de computador que funcionam de forma procedural, ou seja, seguem um conjunto ordenado e sequencial de instruções que visam atender a uma necessidade. Para exemplificar é possível supor um programa que atende a tarefa de escovar os dentes:

1. Pegue a escova de dentes;
2. Pegue a pasta de dentes;
3. Coloque um pouco de pasta na escova;
4. Molhe a escova;
5. Escove os dentes;
6. Enxágue a boca;
7. Lave a escova;
8. Guarde a escova de dentes;
9. Guarde a pasta de dentes;

O problema suposto acima pode ser resolvido de forma procedural pois possui um objetivo definido que pode ser alcançado através de uma sequência simples de instruções. Porém existem problemas mais complexos que não podem ser resolvidos com uma sequência de procedimentos pré-determinados, como por exemplo a ação de diferenciar a escova de dentes da escova de cabelo. Outro problema cuja solução computacional é complexa seria: qual deve ser o diagnóstico de um paciente dado uma série de resultados de exames realizados previamente com esse paciente? Quando o ser humano se depara com esse tipo de problema complexo ele se fundamenta em todo o seu conhecimento e experiências prévias para traçar uma resposta que considere correta. Visando solucionar esse tipo de problemas complexos que surgiu a área da computação denominada Inteligência Artificial.

Segundo (FACELI et al., 2011), a Inteligência Artificial inicialmente se aproveitava de conhecimentos obtidos de especialistas das áreas de interesse para criar regras e hipóteses relativas ao problema a ser resolvido. No entanto, com o passar do tempo até os dias atuais foi percebida a necessidade de automatizar esse processo até então manual de adquirir conhecimento e gerar um programa inteligente. Era necessário que o programa fosse capaz de criar uma função ou hipótese que solucionasse o problema proposto a partir da experiência entregue ao programa. Esse processo de aproveitamento de experiência

para geração de uma solução recebeu o nome *machine learning*, que pode ser traduzido como aprendizado de máquina. Conforme define Faceli et al. (2011, p.2):

> A capacidade de aprendizado é considerada essencial para um comportamento inteligente. Atividades como memorizar, observar e explorar situações para aprender fatos, melhorar habilidades motoras/cognitivas por meio de práticas e organizar conhecimento novo em representações apropriadas podem ser consideradas atividades relacionadas ao aprendizado.

A intenção geral do aprendizado de máquina é que o computador ou programa aprenda através da experiência passada, utilizando o processo de indução para obter conclusões genéricas. Essa experiência é passada por meio de um conjunto de dados, que representam instâncias de exemplo do problema a ser resolvido, contendo atributos que definem as características do problema e a resposta correspondente a esses atributos.

Como Alppaydin (2010) descreve, muitas vezes a falta de conhecimento sobre determinado assunto é compensada com uma quantidade grande de dados sobre o mesmo. Esse mesmo autor cita um exemplo para a aplicação de aprendizado de máquina que ajuda a entender o conceito: distinguir um e-mail legítimo de um e-mail que se trata de um spam, ou seja, um anúncio ou propaganda gerado automaticamente. A saída gerada para esse problema é simples, apenas um atributo "sim/não" que indica se a mensagem é um spam. A entrada corresponde a todas as características quem podem ser obtidas de um determinado e-mail, como e-mail de origem, data de envio, conteúdo da mensagem, presença de imagens de rodapé, etc. Não se sabe como transformar os atributos de entrada no atributo de saída, mas como citado anteriormente, é possível obter um grande volume de exemplos que relacionam esses atributos de entrada e saída. A partir desse conjunto de dados podem ser aplicadas técnicas de aprendizado de máquina de forma que o computador aprenda sozinho o que constitui um e-mail de spam e saiba posteriormente classificar se um e-mail qualquer trata-se de um spam ou não.

3.1 TIPOS DE APRENDIZADO

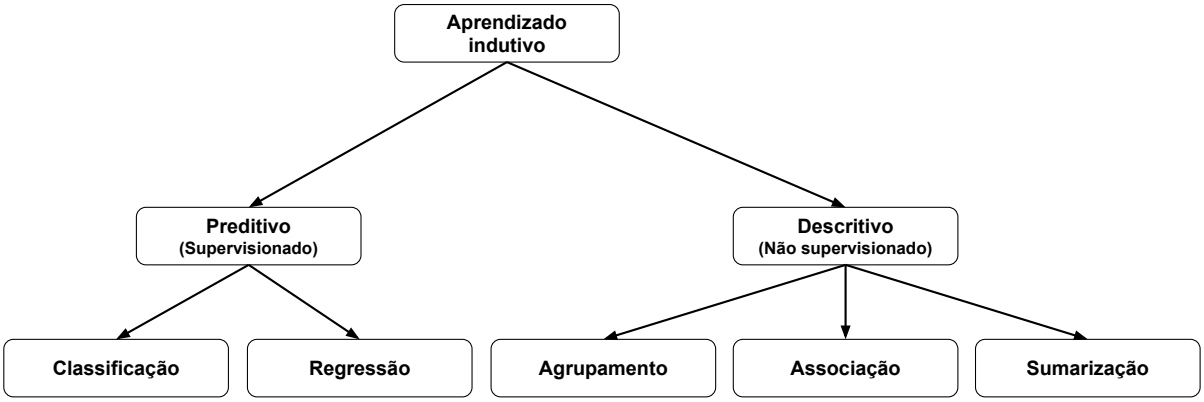
As tarefas de aprendizado de máquina podem ser organizadas em diferentes conjuntos conforme as características e objetivos que possuem. Conforme Faceli et al., a distinção mais geral que pode ser feita diz respeito ao paradigma de aprendizado, ou seja, o objetivo geral que o modelo gerado através do aprendizado deve atender. Conforme esse critério os modelos de aprendizado podem ser separados em modelos descritivos e modelos preditivos.

As tarefas de aprendizado de descrição tem como objetivo a caracterização do conjunto de dados fornecido, não necessitando portanto que seja especificado um atributo de saída dentre os atributos de cada instância de exemplo. Uma meta neste tipo de

aprendizado é identificar grupos de instâncias que se assemelham de alguma forma. Outra meta que pode ser citada é a descoberta de regras de associação entre os atributos do conjunto de dados. Como o conjunto de dados é suficiente por si só para a geração do conhecimento, os algoritmos de aprendizado utilizados para tarefas de descrição seguem o paradigma de aprendizado não supervisionado.

As tarefas de aprendizado preditivo, diferentemente das tarefas de descrição, necessitam que seja especificado um atributo de saída, o qual será alvo da predição. Sendo assim, surge a figura de um "supervisor" externo ao conjunto de dados, alguém que determina qual é o atributo de saída e que conhece o valor esperado para ele em cada instância apresentada no conjunto de dados de treinamento. Por esse motivo, os algoritmos de aprendizado para modelos preditivos seguem o paradigma de aprendizado supervisionado. A meta neste tipo de tarefa é gerar um modelo capaz de prever o valor do atributo de saída dado os valores específicos dos atributos de entrada. Sendo assim, o supervisor pode também avaliar a precisão que o modelo obteve após o treinamento. Esse é o tipo de tarefa de aprendizado que será utilizada neste estudo, haja vista que o objetivo é predizer se uma determinada alteração no sistema gerará inconsistência.

Figura 5: Hierarquia de aprendizado



Fonte: Adaptado pelo autor segundo Faceli et al. (2011)

Na Figura 5 está demonstrada uma divisão simples entre os diferentes tipos de tarefas de aprendizado. O item principal é o aprendizado indutivo, que se refere a obtenção de conhecimento ou generalizações a partir do conjunto de dados apresentado. A divisão seguinte diz respeito à natureza descritiva (não supervisionada) ou preditiva (supervisionada). Na parte de aprendizado preditivo, a divisão seguinte é dada pelo tipo de dado do atributo de saída, também chamado de rótulo ou atributo de classe. O modelo preditivo é do tipo de classificação quando os dados de saída são discretos, ou seja, possuem um conjunto de valores pré-determinados e restritos, como por exemplo "sim" ou "não". Por outro lado o atributo de saída pode ter valores contínuos, como R$ 1,23 ou 6,57m, nestes casos o modelo preditivo é de regressão. Os modelos de regressão são

capazes de responder perguntas como a seguinte: Quando irá custar uma casa com determinados atributos sabendo-se os mesmos atributos e o preço de outras casas da mesma região? Nestes casos o modelo preditivo trará como resposta um valor contínuo como os apresentados anteriormente. O objetivo deste trabalho é identificar se uma alteração irá gerar inconsistência ou não, portanto a saída para o modelo poderá ser classes definidas, possibilitando a utilização de modelos preditivos de classificação.

3.2 TÉCNICAS DE APRENDIZADO PREDITIVO

Como citado anteriormente o objetivo do aprendizado neste trabalho é fazer predições. Para exemplificar um caso de predição pode ser considerado o cenário de uma instituição financeira que quer classificar o risco que ela terá ao fornecer crédito para os futuros clientes, como ilustra Alppaydin (2010). Neste cenário, o banco tem acesso ao histórico dos seguintes dados de seus clientes: *income*, que representa a receita que o cliente tem mensalmente, *savings*, que se refere ao valor que o cliente poupa, também mensalmente, e a classificação já feita para o cliente em relação ao investimento. O objetivo é classificar se o fornecimento de crédito para novos clientes terá baixo-risco (*low-risk*) ou alto-risco (*high-risk*) de o cliente não retornar o valor integral do empréstimo para a instituição. Após estes dados passarem por técnicas de aprendizado de máquina pode ser obtida uma regra para a classificação que se assemelhe à seguinte:

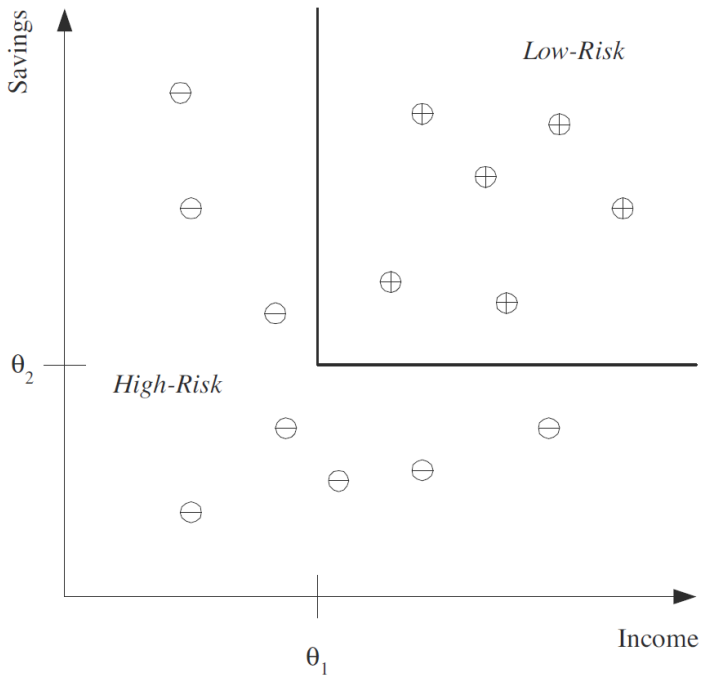
IF $income > \Theta_1$ AND $savings > \Theta_2$ THEN *low-risk* ELSE *high-risk*

Essa regra localizada explica os dados passados no conjunto de treino fornecido para o aprendizado de máquina. Nela é utilizado o termo Θ_1, que se refere a um limiar de *income*, ou seja, um valor específico da receita mensal do cliente. Além deste, também é utilizado Θ_2, que é outro limiar, porém relativo ao valor de *savings*, que é o valor que o cliente poupa mensalmente. Juntos esses dois valores de Θ_1 e Θ_2 criam uma separação no conjunto de dados que divide corretamente as instâncias conforme a classe original de cada uma. Sendo o futuro semelhante ao passado, é possível classificar novas situações de crédito a partir deste modelo gerado, obtendo predições corretas para elas.

A Figura 6 ilustra visualmente os dados fornecidos para treinamento deste modelo preditivo, assim como a regra identificada e a classificação feita. Cada círculo representa uma instância do problema, ou seja, um cliente que solicitou crédito. O símbolo '+' ou '-' em cada instância representam a classificação prévia já sabida para aquele determinado cliente, sendo respectivamente de baixo-risco e alto-risco. As linhas delimitadas por Θ_1 e Θ_2 indicam a regra de classificação obtida no modelo gerado após o processo de aprendizado, através da qual é possível enquadrar novas instâncias na categoria adequada: *Low-risk* ou *High-risk*.

O processo descrito ilustra os procedimentos relativos ao aprendizado preditivo

Figura 6: Conjunto de treinamento para aprovação de crédito



Fonte: Alppaydin (2010)

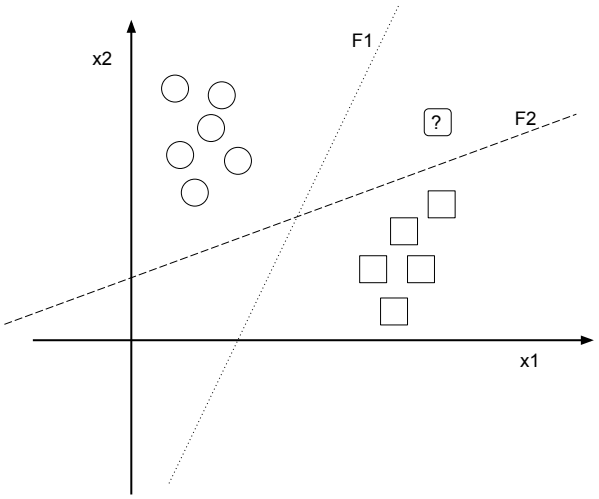
de forma breve, assim como os benefícios que podem ser obtidos através dele. A geração efetivamente do modelo preditivo pode ser feita de várias formas diferentes, através de algumas técnicas de aprendizado preditivo, as quais estão detalhadas nas próximas seções.

3.2.1 SVM e SVC

SVM é a sigla para *Support Vector Machine*, que pode ser traduzido como Máquina de Vetores de Suporte. Esta técnica de aprendizado analisa as correlações entre os atributos do conjunto de dados, procurando definir uma função que possa separar as classes de exemplo corretamente conforme o rótulo que está associado a ela. Conforme (ALPPAYDIN, 2010), essa função também é chamada de hiperplano, e permite classificar corretamente novos exemplos que contenham os mesmos atributos que o conjunto de dados de treino, gerando portanto uma predição da classificação dessa instância. No entanto, selecionar a função que melhor separa as classes não é uma tarefa simples, pois existem inúmeras soluções possíveis. A Figura 7 exemplifica uma situação que pode ocorrer conforme é analisada a correlação entre os atributos x1 e x2.

No gráfico da Figura 7 estão exemplificadas instâncias de duas classes distintas, uma representada por um círculo e outra representada por um quadrado. A posição de cada instância é definida conforme os valores dos atributos x1 e x2. Neste conjunto de dados existe uma separação clara entre as duas classes possíveis porém, como comentado anteriormente, existem várias funções que poderiam separar corretamente essas classes.

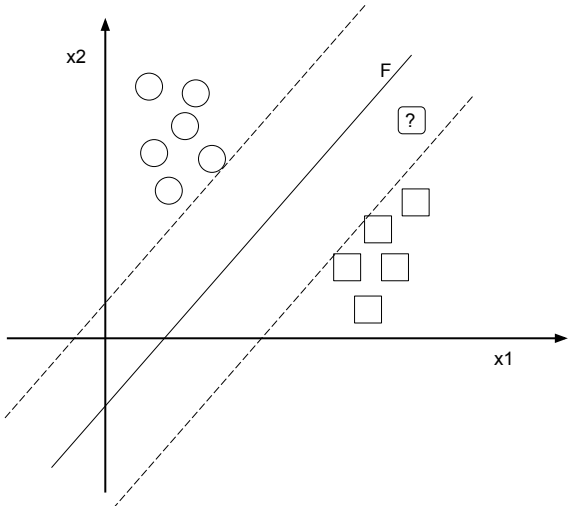
Figura 7: Classificações possíveis conforme atributos x1 e x2



Fonte: Elaborado pelo autor

Uma solução possível é a função representada pela linha pontilhada F1. Outra solução também válida é a função representada pela linha tracejada F2. No entanto, ao inserir uma nova instância é possível notar que ela pode ser classificada de formas diferentes pela função F1 ou F2. No gráfico, essa instância nova está representada pelo caractere "?". Os modelos de aprendizado SVM buscam justamente resolver esse impasse, propondo que a função mais adequada para descrição do conjunto de dados é aquela que obtém a maior distância possível entre as instâncias de classes diferentes. A Figura 8 apresenta a solução encontrada pelo aprendizado utilizando a técnica SVM.

Figura 8: Classificação com método SVC



Fonte: Elaborado pelo autor

A linha representada pela letra F na Figura 8 corresponde à função que separa as instâncias de classes diferentes obtendo a maior distância possível até as instâncias

mais próximas. Essa distância está destacada na imagem através das linhas tracejadas. A denominação *Support Vector Machine* se refere à utilização dessas instâncias como referência, ou seja, vértices de apoio para a seleção da função F. SVC é a sigla utilizada para indicar a técnica *C-Support Vector Classification*, que é uma implementação de SVM voltada para a classificação.

3.2.2 Árvore de decisão

As árvores de decisão se baseiam na estratégia dividir para conquistar, transformando um problema complexo de classificação em vários problemas menores de decisão. Conforme Faceli et al. (2011) indica, formalmente uma árvore de decisão pode ser descrita como um grafo acíclico direcionado. Sendo assim, a representação gráfica da árvore de decisão pode ser comparada à um fluxograma, o que facilita a interpretação das regras geradas pelo aprendizado de máquina pelos seres humanos.

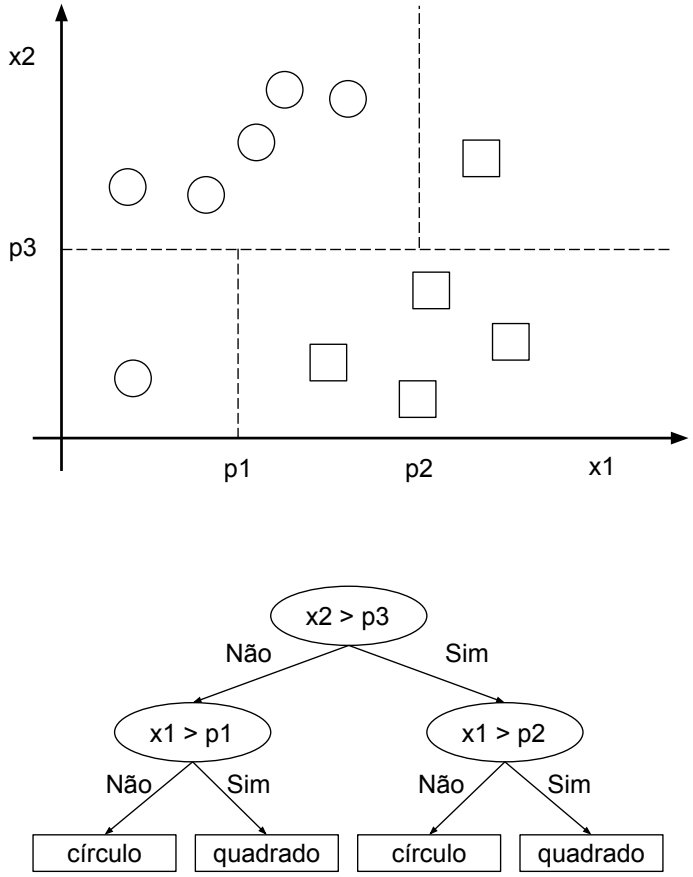
No contexto do conjunto de dados para aprendizado de máquina, cada nodo da árvore irá avaliar o conteúdo de um atributo. Segundo (GOLDSCHMIDT; PASSOS, 2005), se um atributo não aparece na árvore de decisão gerada pelo aprendizado, este atributo pode ser considerado irrelevante para o problema.

O início de uma árvore de decisão é um nodo raiz que contém a principal pergunta a ser respondida. A resposta à essa pergunta leva para uma das várias ramificações disponíveis para o nodo. Cada ramificação está ligada a um outro nodo, que por sua vez também pode possuir uma pergunta a ser respondida. Esse processo então é repetido até que se encontre um nodo folha, ou seja, um nodo terminal que não contém uma pergunta, mas sim a classe à qual a instância analisada se enquadra. Conforme (LUCAS, 2011), existe uma hierarquia nos nodos da árvore, iniciando na raiz com o nodo mais discriminante, ou seja, o que melhor separa o conjunto. Após este, são selecionados para os próximos nodos os próximos atributos mais discriminantes.

A Figura 9, na parte superior, apresenta um exemplo de conjunto de instâncias de duas classes: círculo e quadrado. Essas instâncias estão dispostas conforme os atributos x1 e x2. A técnica de árvore de decisão busca selecionar os atributos cujos valores podem separar corretamente as classes do conjunto de dados. Neste mesmo exemplo, os pontos p1, p2 e p3 são utilizados para delimitar as classes dentro do conjunto. Utilizando esses limites, pode ser gerada a árvore de decisão apresentada na parte inferior da Figura 9.

Uma vantagem obtida pela estrutura hierárquica da árvore de decisão é a agilidade com que se pode encontrar a classificação para determinado conjunto de atributos de entrada (ALPPAYDIN, 2010). Cada decisão resolvida na estrutura da árvore elimina uma parte dos resultados possíveis. No caso de uma árvore como a apresentada na Figura 9, que possui decisões binárias, cada resposta elimina metade dos resultados restantes.

Figura 9: Classificação com método Árvore de decisão



Fonte: Elaborado pelo autor

3.2.3 Random Forest

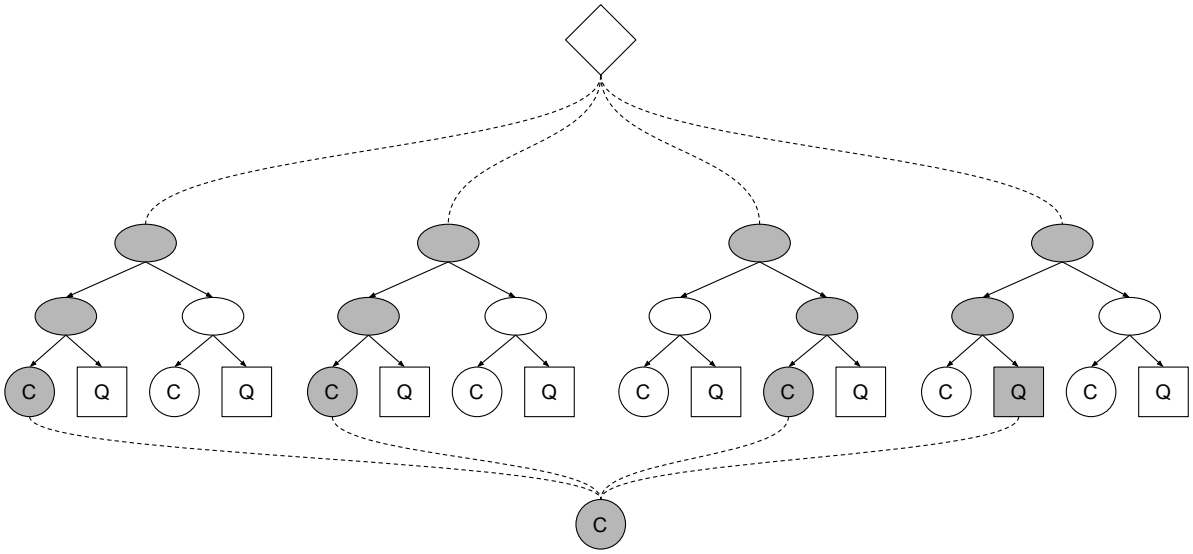
O método *Random Forest* é um aperfeiçoamento da árvore de decisão, buscando aumentar o poder preditivo e adaptabilidade do modelo gerado. A árvore de decisão seleciona a ordem dos nodos com base nos atributos que melhor separam as classes, no entanto, isso é feito com base no conjunto de dados de treinamento ao qual o modelo preditivo tem acesso. Sendo assim, o modelo de árvore de decisão gerado pode sofrer o *overfitting*, que significa que o modelo ficou superajustado ao conjunto de dados treinamento, tendo um desempenho bom dentro deste conjunto porém obtendo um desempenho ruim para novos cenários ao qual o modelo é exposto. Ainda analisando o cenário dos modelos de árvore de decisão, é possível observar que um mesmo conjunto de dados pode gerar várias árvores diferentes, com desempenho semelhante. É possível que a árvore que melhor explica o cenário geral analisado não seja a que gera as melhores predições no conjunto de dados de treinamento e, sendo assim, esta árvore não seria utilizada, pois o que determina a árvore selecionada do modelo é o desempenho no conjunto de dados de teste.

Visando gerar mais adaptabilidade na aprendizagem dos modelos preditivos surgiu

o conceito de *ensemble learning*, ou aprendizado de conjunto. Segundo Dietterich (2002), a abordagem neste tipo de aprendizado não é encontrar uma única hipótese que melhor descreve os dados, mas sim construir um conjunto de hipóteses para este fim. Para a predição, cada uma das hipóteses gera uma classificação que então é combinada com a classificação das demais hipóteses gerando a resposta definitiva. O funcionamento é semelhante à uma votação, em que cada hipótese tem um peso diferente e gera uma classificação própria. Dietterich (2002) ainda afirma que as técnicas que utilizam *ensemble learning* tipicamente são mais eficientes que técnicas que geram apenas uma hipótese.

O aprendizado com *Random Forest* nada mais é do que o método *ensemble learning* aplicado ao conceito de árvore de decisão. Dessa forma, o conjunto de hipóteses se apresenta na forma de um conjunto de árvores de decisão, que justificam o nome *Random Forest* que pode ser traduzido como "Floresta aleatória". Os atributos das árvores de decisão são selecionados de forma aleatória, garantindo que as hipóteses geradas sejam diferentes. O peso que a classificação dada por cada árvore de decisão terá é baseado no desempenho que a árvore teve nas predições dentro do conjunto de treino. Seguindo o conceito do *ensemble learning*, a classificação majoritariamente votada pelas árvores de decisão da floresta será a classificação definitiva gerada pelo modelo.

Figura 10: Classificação com método *Random Forest*



Fonte: Elaborado pelo autor

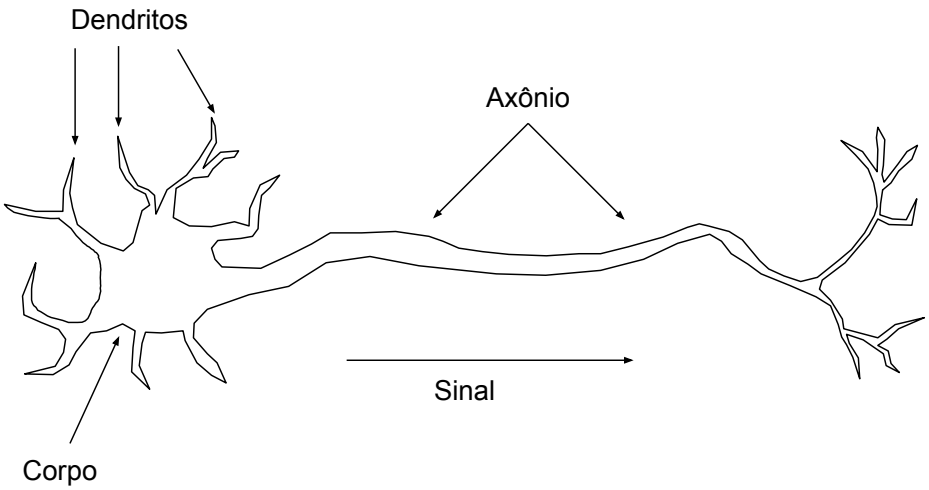
Na Figura 10 está apresentado um exemplo de classificação utilizando o modelo de *Random Forest*. Este modelo de exemplo possui quatro árvores de decisão, cada qual com distribuições únicas de atributos nos nodos. O resultado de cada árvore de decisão é a classificação conforme o rótulo "Q" ou "C". O fluxo de decisões executado por cada árvore está destacado em cinza na imagem. Pode ser observado que não existiu consenso

na classificação de todas as árvores de decisão, sendo portanto gerado o resultado a partir da classificação predominante, neste caso a classe “C”.

3.2.4 Rede Neural do tipo MLP

MLP é a sigla para o termo em inglês *multilayer perceptron*, que é uma implementação de Rede Neural Artificial (RNA), conceito este que precisa ser compreendido primeiramente antes de caracterizar um *perceptron*. Inspiradas no sistema nervoso, as RNAs surgiram com a intenção de aproximar o aprendizado de máquina do comportamento do cérebro humano. Para isso, foi estudado o comportamento dos neurônios e da dinâmica de interação deles quando o sistema nervoso recebe algum estímulo.

Figura 11: Componentes de um neurônio



Fonte: Elaborado pelo autor

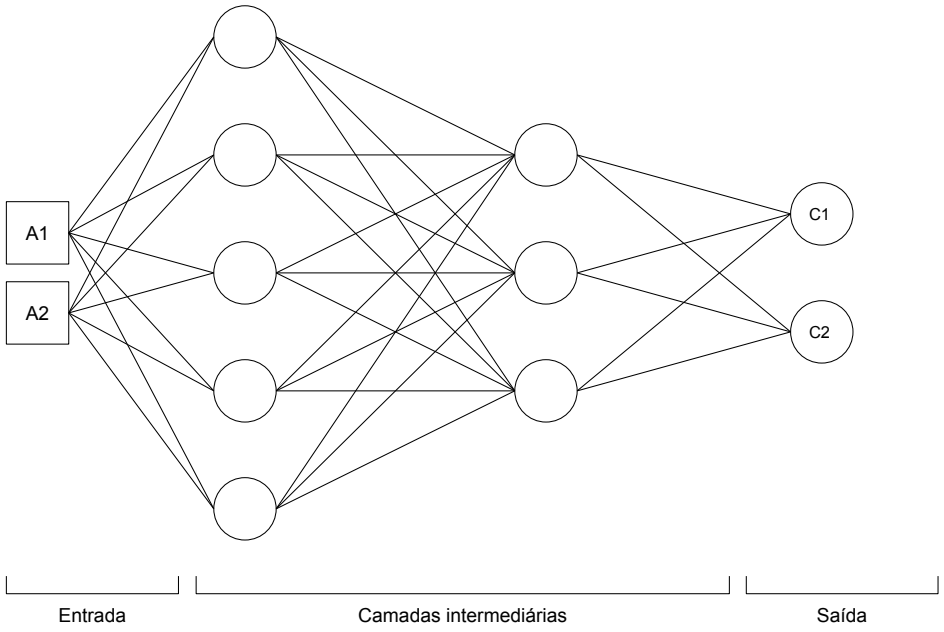
A Figura 11 apresenta os componentes básicos de um neurônio. Os estímulos são recebidos nos dendritos e repassados para o corpo celular. Este, por sua vez, combina e processa os estímulos e, conforme for a intensidade e frequência deles, gera um novo estímulo que é transmitido pelo axônio. O destino deste sinal tipicamente é outro neurônio, que trabalhará da mesma forma, gerando o aprendizado no cérebro humano quanto ao tratamento que deve ser dado para diferentes estímulos recebidos. As Redes Neurais Artificiais aproveitaram esse funcionamento biológico para criar uma reprodução computacional do neurônio, um neurônio artificial, no qual os dendritos ficam ligados aos atributos do conjunto de dados gerando como resposta a classificação do dado.

Segundo Faceli et al. (2011), a primeira RNA a ser implementada foi a rede *perceptron*, por Frank Rosenblatt em 1958. Essa rede era composta por um único neurônio, porém já possuía um processo de treinamento que era responsável por adequar os pesos atribuídos para cada conexão de entrada da rede. O criador deste tipo de RNA ainda foi capaz de provar o teorema de convergência da rede perceptron, que afirma que, se

é possível classificar um conjunto de entradas linearmente, uma rede *perceptron* fará a classificação. No entanto, esse teorema é também a maior limitação das redes neurais com apenas uma camada de neurônios, pois através delas não é possível classificar conjuntos de dados cuja distribuição não é linear.

Como forma de ampliar a capacidade das RNAs passou-se a criar redes com várias camadas, pois assim era possível resolver inclusive problemas que não eram linearmente separáveis. O modelo MLP trata-se de uma rede *perceptron* com várias camadas, recebendo portanto o nome *multilayer perceptron*.

Figura 12: Rede de aprendizado MLP



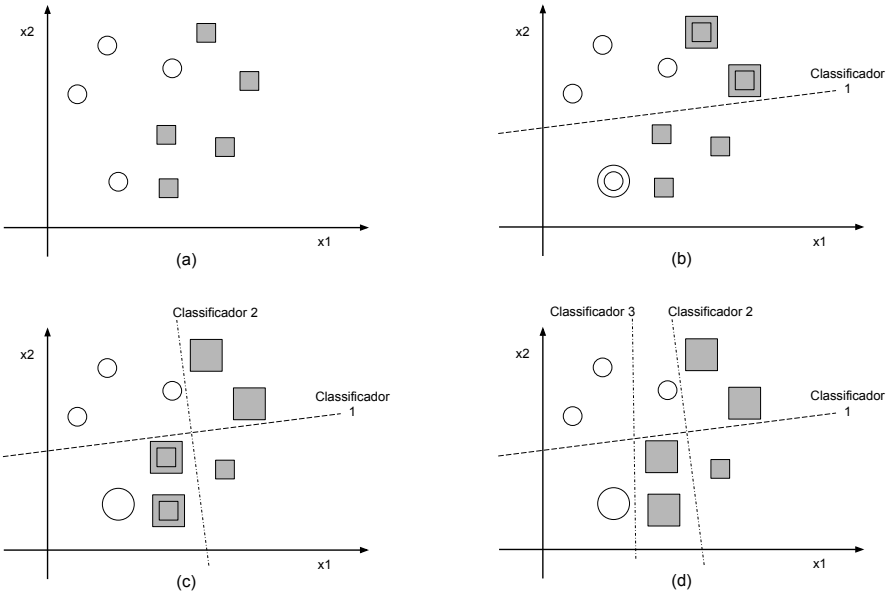
Fonte: Elaborado pelo autor segundo Faceli et al. (2011)

A Figura 12 exemplifica uma rede MLP que é composta por um conjunto de entrada, camadas intermediárias de neurônios e a classificação de saída. A1 e A2 representam os atributos do conjunto de dados, aos quais tipicamente todos os neurônios da primeira camada se conectam. As camadas intermediárias podem ter quantidades diferentes de neurônios sendo que cada neurônio possui uma função de ativação própria que trabalha os dados recebidos das camadas anteriores e gera um resultado para a camada seguinte. Segundo Cybenko (1989), a utilização de uma camada intermediária permite gerar classificações contínuas. Já a utilização de duas camadas intermediárias permite a aproximação de qualquer função. Por fim, C1 e C2 representam as classes possíveis para classificação. Após os atributos A1 e A2 passarem por todas as funções de ativação da rede serão gerados resultados nos nodos C1 e C2, sendo o nodo que possuir o valor mais elevado o resultado da classificação.

3.2.5 Gradient Boosting Classifier

O classificador *Gradient Boosting* é baseado no uso de uma técnica de processamento chamada *Boosting*. O método consiste na utilização um grupo de classificadores "fracos", que possuem regras simples de classificação, porém que em conjunto são capazes de gerar resultados mais confiáveis. Esse conceito é muito próximo ao aprendizado de conjunto descrito na subseção 3.2.3, porém um grande diferencial está na forma como os classificadores que compõem esse conjunto se relacionam no método de *Boosting*. Conforme (ALPPAYDIN, 2010), cada modelo é gerado priorizando a classificação correta de instâncias que receberam rótulos incorretos pelo modelo anterior, buscando portanto aprimorar o desempenho do conjunto de classificadores.

Figura 13: Construção de classificador com *Boosting*



Fonte: Elaborado pelo autor

A Figura 13 apresenta um processo ilustrativo para construção de um classificador com *Boosting*. Na Figura 13(a) está representado o conjunto de instâncias que precisa ser classificado, sendo uma classe ilustrada por um círculo branco e outra por um quadrado escuro. A técnica inicia com a geração do primeiro classificador fraco, que tem a regra de classificação sinalizada pela literal "Classificador 1" na Figura 13(b). Como pode ser observado, existem três instâncias que foram classificadas de forma incorreta, sendo elas o círculo branco com borda e os dois quadrados com borda na Figura 13(b). Essas instâncias então ganham um peso maior para a próxima geração de modelo preditivo, de forma que o modelo priorize a classificação correta destas instâncias. Na Figura 13(c) tem-se a regra de classificação do segundo modelo preditivo sinalizada por "Classificador 2", sendo que esta regra também apresenta falhas na classificação final das instâncias. Os quadrados

com borda destacados na Figura 13(c) ganham um peso maior para a geração do próximo modelo preditivo, o "Classificador 3" sinalizado na Figura 13(d). Dessa forma, mesmo utilizando classificadores mais simples é possível obter resultados consistentes, pois cada novo classificador procura cobrir as carências do classificador criado anteriormente.

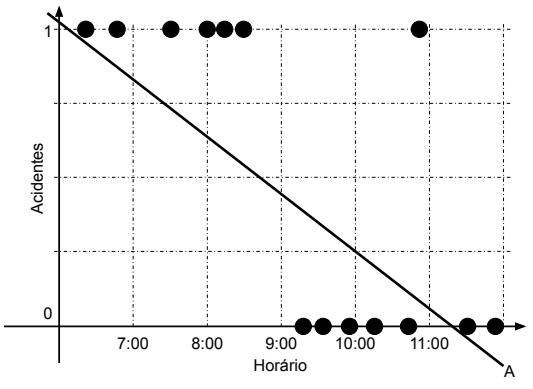
Em uma abordagem de aprendizado de conjunto como a *Random Forest*, a efetividade do classificador é obtida baseada apenas na diversidade das árvores geradas. Segundo Natekin e Knoll (2013), a técnica *Gradient Boosting* por outro lado funciona a partir da formação de conjuntos de forma estratégica, buscando aperfeiçoar o aprendizado obtido com os modelos anteriores evitando a ocorrência de classificações incorretas. Os classificadores gerados a cada interação são chamados de *base-learners*. Conforme Pedregosa et al. (2019b) indica, a implementação do classificador *Gradient Boosting* na biblioteca *scikit-learn* do Python é feita através de árvores de decisão.

3.2.6 Regressão logística

A regressão logística, ou no termo em inglês, *Logist Regression*, é uma técnica de aprendizado de máquina voltada para problemas de classificação, que é o objetivo deste estudo. Este método pode ser considerado um aperfeiçoamento da técnica de regressão linear para problemas em que o resultado deve ser discreto, ou seja, quando pretende-se categorizar a saída conforme forem os dados da entrada.

Para compreender o funcionamento da regressão logística é necessário primeiramente entender a técnica de regressão linear, que pode ser explicada supondo um cenário semelhante ao que foi analisado no estudo de Al-Ghamdi (2002), o qual propôs a regressão logística. O objetivo neste cenário hipotético é gerar a probabilidade de ocorrer um acidente de trânsito em um cruzamento dado o horário do dia. Tem-se como base para a geração do modelo preditivo as ocorrências passadas de acidentes com o respectivo horário associado.

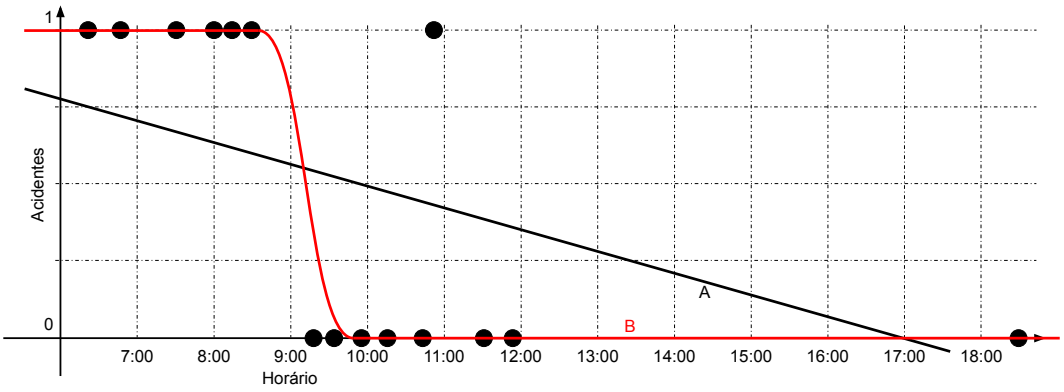
Figura 14: Modelo de regressão linear



Fonte: Elaborado pelo autor

As instâncias fictícias de acidentes ocorridos conforme o horário do dia, assim como o modelo de regressão linear gerado estão representados na Figura 14. Nesta imagem o eixo y indica a ocorrência do acidente, sendo o valor "0" o indicativo de que não houve acidente e "1" o indicativo de que houve um acidente. Neste exemplo existem 14 instâncias e é possível observar já visualmente que, conforme o horário do dia avança, a probabilidade de ocorrer um acidente diminui. A técnica de regressão linear se baseia em encontrar uma função linear que melhor descreva esse decaimento da probabilidade. Na Figura 14 a função encontrada está representada pela linha A.

Figura 15: Comparativo regressão linear e logística



Fonte: Elaborado pelo autor

A partir da função "A" identificada na Figura 14 é possível definir um valor resultante que serve de limiar para a classificação, permitindo rotular instâncias com valor inferior à esse limiar com uma classe e instâncias com valor superior com outra classe. Para esse conjunto de dados a regressão linear é capaz de classificar corretamente as instâncias, porém ao adicionar um *outlier* esse método apresenta problemas. A Figura 15 contém o mesmo conjunto de dados inicial, porém com uma instância extra que é um *outlier*, um dado que se destaca dos demais. Essa instância é o registro sem acidente presente após às 18:00 e em função desse registro a função de regressão linear ficaria diferente, como pode ser observado na linha "A". Como a função linear agora se alonga mais para a direita, a probabilidade de ocorrer um acidente indicada pela função para cada horário agora é maior, fazendo com que horários que anteriormente eram considerados com pouca probabilidade de ocorrer um acidente passassem a ser classificados de forma contrária.

Para solucionar esses problemas de classificação foi elaborado o conceito de regressão logística, que funciona de forma semelhante à regressão linear porém substituindo o uso de funções lineares, que seriam retas, por funções não lineares, capazes de gerar curvas que se adéquam melhor aos dados fornecidos. Isso faz com que a presença de *outliers* não afete a classificação geral devolvida pelo modelo. Além disso, como indica Al-Ghamdi (2002), uma das características que difere o modelo de regressão linear do modelo de regressão logística é o resultado fornecido pelo modelo. Enquanto a regressão linear pode

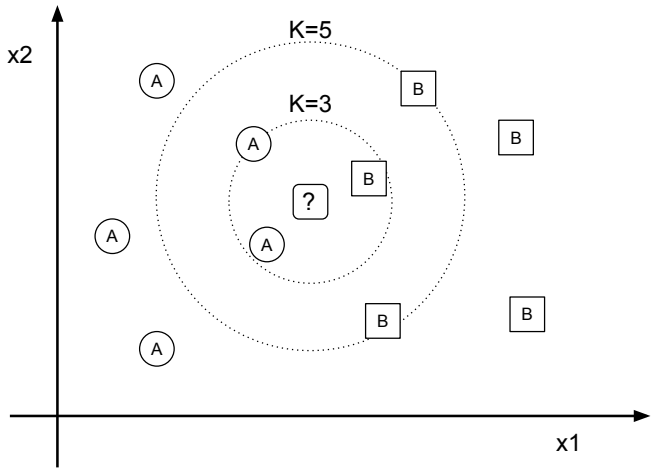
fornecer probabilidades que variam de 0% até 100%, o resultado da regressão logística é sempre binário, podendo ser utilizada para classificações.

Uma possível função de regressão logística para o cenário de exemplo está representada na Figura 15 através da linha "B". É possível observar que, apesar da presença de um *outlier*, a função não foi afetada, mantendo a região de transição próxima ao horário 9:00, que é o momento em que ocorre a maior variação entre as classes das instâncias.

3.2.7 K-Nearest Neighbors

Também referenciada por sua abreviação, KNN, a técnica *K-Nearest Neighbors* pode ser traduzida para K-vizinhos mais próximos. Esse método de classificação consiste em localizar as instâncias geometricamente mais próximas dada uma nova amostra que precisa ser classificada conforme o conjunto de dados fornecido. O parâmetro mais relevante deste algorítimo é o K, que indica quantos vizinhos serão considerados para a classificação.

Figura 16: Classificação com método K-Nearest Neighbors



Fonte: Elaborado pelo autor

Segundo Harrington (2012), o método inicia buscando as instâncias que estão mais próximas da nova instância que está sendo analisada. São procuradas K instâncias do conjunto de dados, sendo o parâmetro K uma referência para o termo em inglês *known*, que significa "conhecido". Na Figura 16 é possível observar um conjunto de dados de exemplo composto por instâncias de duas classes distintas: A e B. O objetivo neste exemplo é classificar a nova instância sinalizada pelo caractere "?". Assumindo o valor do parâmetro K como 3, seriam encontradas duas instâncias A e uma instância B. A classificação final da instância "?" é dada pela classe predominante entre as classes localizadas e, neste caso, a classificação seria para o rótulo A. É importante destacar também que o parâmetro K está fortemente vinculado com a classificação final que as instâncias novas analisadas

podem receber. Ainda no exemplo da Figura 16 ao utilizar-se um valor diferente para K, como o valor 5 ilustrado, a classificação final da instância seria diferente. Isso ocorre pois neste novo cenário seriam localizadas três instâncias da classe B e apenas duas instâncias da classe A, fazendo com que a classificação final da instância "?" passasse a ser para a classe B.

O algoritmo *K-Nearest Neighbors* possui um funcionamento simples, porém exige poder computacional elevado e demanda muita memória (HARRINGTON, 2012), pois para cada instância analisada é necessário avaliar as distâncias geométricas para todas as outras instâncias do *dataset*. Além disso, para problemas de classificação que envolvem apenas duas classes, é importante selecionar valores ímpares para o parâmetro K, pois isso evita situações de empate na contagem do número total de cada classes presente entre os vizinhos mais próximos. Apesar destes pontos negativos, esta técnica de predição ainda apresenta a vantagem de não ser afetada negativamente pela presença de *outliers*.

3.2.8 Naive Bayes

Diferentemente de outros algoritmos de classificação, a técnica *Naive Bayes* é baseada em teoremas de probabilidade, de forma que a classe resultado da classificação é a que tem a maior probabilidade de estar correta. Esse classificador utiliza o Teorema de Bayes, criado por Thomas Bayes. O termo "Naive", que em inglês significa "ingênuo", se refere a uma das principais características deste classificador, pois ele não leva em consideração nenhuma correlação entre os atributos do problema analisado, fazendo a suposição ingênua de que não existem correlações (HARRINGTON, 2012). Isso significa que, se é sabido que para uma fruta ser considerada uma maça ela deve se vermelha, redonda e com aproximadamente 10 cm de tamanho, o método de *Naive Bayes* irá trabalhar cada um desses parâmetros de forma independente.

Considerando um cenário em que existem 1000 instâncias de duas classes distribuídas de forma semelhante a Figura 16, em que uma classe se concentra mais para a direita e outra classe mais para a esquerda, pode ser comparada a utilização de um classificador probabilístico frente aos demais. Neste cenário, uma possibilidade seria utilizar um classificador *K-Nearest Neighbors*, porém isso demandaria realizar o cálculo de 1000 distâncias para cada nova instância analisada. Por outro lado, ao utilizar o método *Naive Bayes* seria necessário apenas calcular a probabilidade de cada classe para a nova instância, e comparar as mesmas, o que se apresenta como um cálculo mais simples que a abordagem por KNN (HARRINGTON, 2012).

$$P(c|x,y) = \frac{P(x|c) * P(y|c) * P(c)}{P(x) * P(y)} \tag{3.1}$$

A Equação 3.1 apresenta o modelo de cálculo de probabilidade de Bayes segundo

Harrington (2012). O termo *P(c|x,y)* refere-se à probabilidade da instância analisada pertencer a classe c dados os parâmetros x e y. O termo *P(x|c)* é o número de ocorrências de x dentre as instâncias que pertencem à classe c. Esta mesma lógica se aplica ao termo *P(y|c)*, porém este se refere às ocorrências do atributo y. A probabilidade *P(c)* expressa o número de ocorrências da classe c sobre o total de amostras da base de dados analisada. Por fim, os termos *P(x)* e *P(y)* apresentam a probabilidade de ocorrência de x e y, respectivamente, dentre todas as amostras do conjunto, independentemente da classe resultado.

Tabela 1: Conjunto de dados de dias e partidas de tênis

Dia	Aspecto	Temperatura	Umidade	Vento	Decisão
1	Sol	Quente	Alta	Fraco	Não
2	Sol	Quente	Alta	Forte	Não
3	Nublado	Quente	Alta	Fraco	Sim
4	Chuva	Agradável	Alta	Fraco	Sim
5	Chuva	Fria	Normal	Fraco	Sim
6	Chuva	Fria	Normal	Forte	Não
7	Nublado	Fria	Normal	Forte	Sim
8	Sol	Agradável	Alta	Fraco	Não
9	Sol	Fria	Normal	Fraco	Sim
10	Chuva	Agradável	Normal	Fraco	Sim
11	Sol	Agradável	Normal	Forte	Sim
12	Nublado	Agradável	Alta	Forte	Sim
13	Nublado	Quente	Normal	Fraco	Sim
14	Chuva	Agradável	Alta	Forte	Não

Fonte: Elaborado pelo autor

Para exemplificar a utilização das funções de Bayes pode ser considerada a Tabela 1 que contém atributos das condições climáticas de 14 dias e a respectiva decisão tomada por uma pessoa fictícia para jogar tênis ou não. O objetivo é a predição da coluna decisão de um dia com as seguintes características: Aspecto=Sol, Temperatura=Fria, Umidade=Alta, Vento=Forte.

$$P(Sim) = \frac{P(Sol|Sim) * P(Fria|Sim) * P(Alta|Sim) * P(Forte|Sim) * P(Sim)}{P(Sol) * P(Fria) * P(Alta) * P(Forte)} \tag{3.2}$$

$$P(Sim) = \frac{\frac{2}{9} * \frac{3}{9} * \frac{3}{9} * \frac{3}{9} * \frac{9}{14}}{\frac{5}{14} * \frac{4}{14} * \frac{7}{14} * \frac{6}{14}} \tag{3.3}$$

$$P(Sim) = 0,242 = 24,2\% \tag{3.4}$$

$$P(Não) = \frac{P(Sol|Não) * P(Fria|Não) * P(Alta|Não) * P(Forte|Não) * P(Não)}{P(Sol) * P(Fria) * P(Alta) * P(Forte)} \tag{3.5}$$

$$P(Não) = \frac{\frac{3}{5} * \frac{1}{5} * \frac{4}{5} * \frac{3}{5} * \frac{5}{14}}{\frac{5}{14} * \frac{4}{14} * \frac{7}{14} * \frac{6}{14}} \tag{3.6}$$

$$P(Não) = 0,941 = 94,1\% \tag{3.7}$$

A Equação 3.2 organiza os atributos que serão considerados em relação a classe resultado “Sim”. Na Equação 3.3 estão expostos os valores para cara atributo utilizado anteriormente. Nesta equação pode ser visto, por exemplo, que o termo *P(Sol|Sim)* assumiu o valor $\frac{2}{9}$ pois existem 2 dias com aspecto “Sol” dentre os 9 dias em que a decisão foi “Sim”. As resolver-se essa equação tem-se o resultado expresso na Equação 3.4, que indica que para os valores definidos para os atributos de classe existe 24,2% de probabilidade de ocorrer a decisão “Sim". A Equação 3.5, Equação 3.6 e Equação 3.7 apresentam o mesmo processo para a classe “Não”, dados os mesmos valores para os atributos. No caso da classe “Não” a probabilidade é de 94,1%. Neste ponto, o classificador *Naive Bayes* então compararia a probabilidade de cada uma das duas classes e geraria como resultado a classe mais provável, que neste exemplo é a classe “Não”.

3.3 LINGUAGEM DE PROGRAMAÇÃO PYTHON

Python é uma linguagem de programação de alto nível criada em 1991 que atualmente possui um modelo de desenvolvimento comunitário, porém ainda gerenciado pela *Python Software Foundation*, uma organização sem fins lucrativos. A linguagem prioriza a legibilidade de código, portanto possui uma sintaxe clara e por vezes simplificada que permite o desenvolvimento ágil de programas simples (Python Software Foundation, 2017). Apesar da simplicidade da linguagem, ela ainda dispõe de várias bibliotecas padrões para executar tarefas comuns à diversas implementações, como conectar em *Web Servers*, buscar textos com expressão regular ou ainda ler e modificar arquivos. Além destas bibliotecas padrões, a comunidade ainda desenvolve e disponibiliza outras bibliotecas que internamente utilizam linguagens de programação compiladas como C ou C++.

Segundo Raschka (2017), Python é uma das linguagens de programação mais populares no que se refere à Ciência de Dados, pois conta com várias bibliotecas específicas para este fim. O Python é uma linguagem de programação interpretada, portanto ela possui um desempenho inferior em processamentos intensos que exigem bastante poder computacional. Por este motivo as bibliotecas para Ciência de Dados são muitas vezes escritas utilizando linguagens de programação de baixo nível, garantindo uma performance boa para tarefas de *machine learning*.

Um dos primeiros passos para geração do aprendizado é ler e ajustar o conjunto de dados de treinamento. Estes dados podem se apresentar na forma de um arquivo CSV, que é a sigla para *Comma-Separeted Values*, ou seja, um arquivo cujos valores estão separados por vírgula. Este tipo de arquivo permite agrupar e organizar dados que podem ser descritos na forma de uma tabela. No contexto da Ciência de Dados, cada coluna desta tabela seria referente a um atributo enquanto que as diversas linhas representariam várias instâncias estudadas em um determinado cenário.

Tabela 2: Conjunto de dados de pesos, alturas e IMC

Nome	**Peso**	**Altura**	**IMC**
Breno	75kg	1.78m	23.7
Carla	55kg	1.60m	21.5
Matias	80kg	1.70m	27.7

Fonte: Elaborado pelo autor

Figura 17: Conteúdo de arquivo CSV

```
1  "NOME","PESO","ALTURA","IMC"
2  BRENO,75KG,1.78m,23.7
3  CARLA,55KG,1.60m,21.5
4  MATIAS,80KG,1.70m,27.7
```

Fonte: Elaborado pelo autor

A Tabela 2 apresenta um exemplo de conjunto de dados fictícios organizados na forma de uma tabela. Esses mesmos dados podem ser descritos em um arquivo CSV, e neste caso ficariam semelhantes ao exemplo da Figura 17. Na linguagem Python existe a biblioteca Pandas que permite trabalhar facilmente os dados de arquivos cujos dados estão rotulados. A utilização dessa biblioteca permite fazer diversas operações com o conjunto de dados, como adicionar ou remover colunas, além de ser uma ferramenta muito rápida em suas operações (PytData, 2019).

Outra ferramenta amplamente utilizada é a biblioteca *scikit-learn*, que disponibiliza implementações no estado da arte de diversos algoritmos de aprendizado de máquina. Segundo Pedregosa et al. (2011), o *scikit-learn* se destaca entre outras bibliotecas no Python por diversas razões: é distribuída com código aberto; foi desenvolvida para ter eficiência; possui poucas dependências, o que facilita a distribuição e instalação. Através dessa biblioteca é possível implementar os algoritmos SVC, árvore de decisão e demais algoritmos discutidos no Capítulo 3.

3.4 MÉTRICAS

Problemas de classificação binários possuem apenas duas classes possíveis para classificação, a classe positiva (P) e a classe negativa (N). Considerando as classes originais, é possível separar a classificação gerada pelo modelo preditivo em quatro conjuntos: duas classificações corretas, verdadeiros positivos (VP) e verdadeiros negativos (VN), e duas classificações incorretas, falsos positivos (FP) e falsos negativos (FN). A matriz de confusão é uma tabela que apresenta essas quatro medidas.

Figura 18: Regiões da matriz de confusão

		Classe real	
		Positiva	Negativa
Classe predita	Positiva	Verdadeiros Positivos	Falsos Positivos
	Negativa	Falsos Negativos	Verdadeiros Negativos

Fonte: Elaborado pelo autor

A Figura 18 apresenta a estrutura da matriz de confusão e as respectivas classificações para os valores demonstrados nela. Segundo Saito e Rehmsmeier (2015), a partir dos dados da matriz de confusão podem ser extraídas outras métricas relevantes sobre a eficiência do modelo gerado. Uma das medidas básicas de desempenho é a acurácia, que pode ser calculada com a Equação 3.8:

$$ACC = \frac{VP + VN}{VP + VN + FP + FN} \tag{3.8}$$

A acurácia indica de forma geral o quão corretas são as predições do modelo. Supondo um caso de 21 predições para a classe correta, seja ela positiva ou negativa, de um total de 31 predições geradas, teria-se uma acurácia de aproximadamente 0.68, o que significa que o modelo está gerando predições corretas em 68% dos casos. Por outro lado tem-se a taxa de erro, que funciona de forma semelhante à acurácia, porém avalia as predições falsas. Para calcular a taxa de erro pode-se utilizar a Equação 3.9:

$$ERR = \frac{FP + FN}{VP + VN + FP + FN} \tag{3.9}$$

Existem ainda as avaliações de sensibilidade (SN) e especificidade (SP), que analisam a classe positiva e a classe negativa, respectivamente. A sensibilidade também é

chamada de taxa de verdadeiros positivos e indica quantas das instâncias positivas foram apontadas como positivas pelo modelo preditivo. Já a especificidade ou taxa de verdadeiros negativos mede quantas instâncias que eram negativas foram classificadas como negativas pelo modelo. Essas métricas são calculadas utilizando a Equação 3.10 e a Equação 3.11.

$$SN = \frac{VP}{VP + FN} \tag{3.10}$$

$$SP = \frac{VN}{VN + FP} \tag{3.11}$$

Outras medidas que podem ser aplicadas são a de precisão (PRE) e revocação (R). A precisão mede a relação entre as instâncias que eram positivas e o total de predições para a classe positiva. Essa métrica está exposta na Equação 3.12. A revocação, também conhecida como *recall*, indica o percentual de itens identificados corretamente em uma classe em relação à quantidade de itens que efetivamente pertencem àquela classe na classificação real. A revocação indica portanto a abrangência ou cobertura das predições geradas pelo modelo e está especificada na Equação 3.13. Ambas as medidas são utilizadas para calcular o *F-score*, que é uma métrica com o propósito de indicar o desempenho geral do modelo. Esta medida ainda utiliza um parâmetro α para ponderar a relação entre precisão e revocação, tendo-se tipicamente α com o valor 0,5. A forma de cálculo do *F-score* está demonstrada na Equação 3.14.

$$PRE = \frac{VP}{VP + FP} \tag{3.12}$$

$$R = \frac{VP}{VP + FN} \tag{3.13}$$

$$F = \frac{1}{\alpha \frac{1}{PRE} + (1 - \alpha) \frac{1}{R}} \tag{3.14}$$

4 ANÁLISE DE VIABILIDADE

A partir da fundamentação teórica estudada é possível perceber que existem várias etapas a serem cumpridas para a geração de modelos preditivos. Inicialmente é necessário gerar uma base de dados com informações relevantes sobre o cenário de estudo. Após isso, a base pode ser submetida aos algoritmos de *machine learning* estudados, avaliando-se os resultados obtidos por cada um deles para identificar qual obteve o melhor desempenho. Neste capítulo está apresentado o estudo exploratório realizado sobre esse processo. A seção 4.1 detalha as atividades relativas à obtenção dos dados e preparação dos mesmos. A seguir, descreve-se a execução do aprendizado de máquina utilizando a base de dados criada inicialmente.

4.1 EXTRAÇÃO DE DADOS

O cenário estudado é a implementação de alterações no sistema ERP desenvolvido pela empresa Rech Informática. Como exposto anteriormente, esses dados são registrados na ferramenta Sicla, que é desenvolvida e mantida pela própria empresa. Todas as informações deste sistema ficam registrados em um banco de dados Oracle que pôde ser acessado com a utilização da ferramenta Oracle SQL Developer. Esse programa permite a manipulação de forma simplificada de bases de dado, possibilitando extrair as informações necessárias utilizando comandos SQL (Oracle Corporation, 2019).

As informações do Sicla estão organizadas em várias tabelas relativas a cada entidade controlada dentro do sistema. Inicialmente foi analisada a tabela que armazena informações das Fichas do Sicla. Como detalhado na seção 2.3, a entidade Ficha contém dados da produção de implementações e demais alterações no sistema ERP. Avaliando-se os dados disponíveis nesta tabela foi gerada uma primeira consulta SQL utilizando as informações apresentadas na Figura 19.

Na Figura 19 o campo ERROS refere-se à indicação principal se a alteração gerou alguma inconsistência ou não. Este é o atributo de classe que indica a classe à qual a instância pertence e que será alvo da predição. O dado em USUARIO indica o código do técnico que criou a Ficha. A coluna PROGRAMA tem a informação do principal código do programa envolvido na alteração. Em PROGRAMADOR tem-se o código do colaborador que implementou as alterações, assim como a coluna RESPONSAVEL indica o código do técnico que executou a validação da alteração. CELULA refere-se à área de negócio da alteração, indicando por exemplo se é uma alteração relativa aos sistemas de faturamento ou de controle de produção por exemplo. O TIPO indica a categoria da demanda, podendo ser implementação, correção de erro, entre outros. Os dados TEMPO-

Figura 19: Primeira lista de campos extraídos da base de dados

```
1     ERROS
2     USUARIO
3     PROGRAMA
4     RESPONSAVEL
5     PROGRAMADOR
6     CELULA
7     TIPO
8     TEMPOPRODUCAO
9     TEMPOREVISAO
10    TEMPOVALIDACAO
11    TEMPOPREVISTO
12    TEMRETRABALHO
```

Fonte: Elaborado pelo autor

PRODUCAO, TEMPOREVISAO, TEMPOVALIDACAO indicam os tempos de execução das etapas de programação, revisão e validação, respectivamente. A coluna de TEMPOPREVISTO contém a estimativa inicial de tempo de programação que havia sido dada para a alteração. Por fim, a coluna TEMRETRABALHO indica se houve a ocorrência de retrabalho durante a revisão ou validação da alteração.

Com esta seleção foi possível extrair as informações das Fichas implementadas historicamente na empresa. Foi utilizada como data de referência o período posterior à agosto de 2015, pois após esse período o processo de trabalho da empresa se consolidou utilizando metodologias ágeis como o Scrum. Para não misturar informações de dois processos distintos foi optado por apenas considerar para o aprendizado de máquina as alterações realizadas após agosto de 2015. Com isso a consulta SQL foi feita resultando em um arquivo CSV com as informações selecionadas.

Quase todas as informações extraídas com as colunas selecionadas são dados numéricos, porém a coluna CELULA apresenta classes identificadas por letras, como “N” ou “P”. Para padronizar a base de dados foi realizado um pré-processamento utilizando expressão regular para substituir o conteúdo alfabético desta coluna por um número equivalente.

Os registros de alteração considerados foram as Fichas criadas até março de 2019, para considerar alterações que já foram concluídas e estão difundidas nos clientes. Considerando o período de referência inicial de agosto de 2015 até março de 2019 foram obtidas 37.884 instâncias para o processo de geração de modelos preditivos utilizando técnicas de *machine learning*.

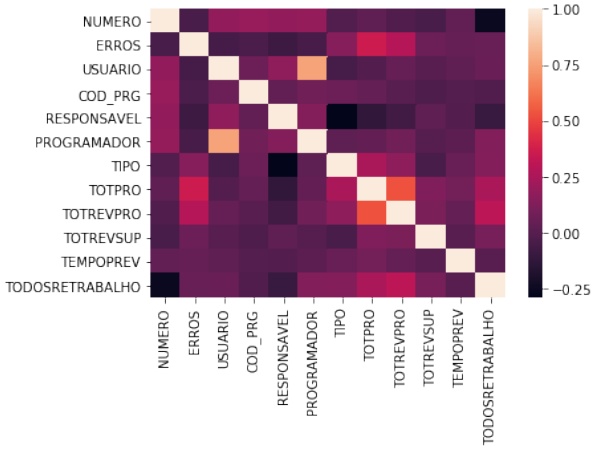
4.2 GERAÇÃO DE MODELOS

Para geração dos modelos preditivos foi escolhido utilizar a linguagem Python, pois essa linguagem conta com diversos recursos voltados para a Ciência de Dados. Alguns dos benefícios da linguagem Python são a biblioteca *scikit-learn*, que conta com diversas implementações de técnicas de *machine learning*, e a licença de código aberto, que torna a linguagem mais difundida e mais amigável para novos desenvolvedores. Uma ferramenta complementar que auxilia o desenvolvimento de estudos utilizando linguagens de programação é o *Project Jupyter*. Através dele é possível utilizar linguagens como Python, R, Julia, Scala entre outras para escrever códigos que geram saídas em texto, gráficos ou visualizações interativas (Project Jupyter, 2019).

No ambiente de desenvolvimento do *Jupyter Notebook* foi desenvolvido inicialmente um código-fonte que pudesse ler as informações contidas no arquivo CSV gerado na extração de dados. Para essa finalidade foi utilizada a biblioteca Pandas, que lê as informações do arquivo CSV e apresenta elas na forma de um *dataset*, o que permite que o código-fonte selecione os dados de uma coluna ou efetue operações sobre os dados de forma mais prática.

A avaliação inicial do conjunto de dados foi feita utilizando a biblioteca Seaborn, que é capaz de gerar diversas visualizações gráficas sobre os dados. Foi utilizado o método *pairplot* que gera uma visualização gráfica dos atributos correlatos, ou seja, as colunas do *dataset* que são redundantes, pois apresentam informações semelhantes. Esse seria o caso, por exemplo, de uma coluna que apresentasse o tempo de produção em minutos e uma outra coluna que contivesse o tempo de produção em horas, pois ambas tem a mesma informação, porém apresentada com um dado diferente.

Figura 20: Mapa de calor de atributos correlatos



Fonte: Elaborado pelo autor

Além da visualização do método *pairplot* existe a saída gerada pelo método *heatmap*, que resume os gráficos de correlatos como um mapa de calor. Esse mapa está

exposto na Figura 20 e ao analisar ele é possível perceber que se destaca uma diagonal, que são justamente os pontos em que o atributo está sendo cruzado com ele próprio. Além destes, os outros pontos que se destacam são a relação entre as colunas de USUARIO e PROGRAMADOR, que são o técnico que redigiu a implementação e o técnico que efetivamente produziu ela, respectivamente. Conforme apontado pelo mapa, em muitos casos um mesmo técnico executa as duas atividades, transformando portanto esses atributos em informações redundantes. Tendo isso em vista, a coluna de USUARIO foi eliminada do conjunto de dados antes de submeter o mesmo às técnicas de *machine learning*.

Com o conjunto de dados devidamente preparado, foram feitos testes utilizando algumas das técnicas de aprendizado preditivo expostas na seção 3.2. A base completa tem aproximadamente 38 mil registros, porém é altamente desbalanceada, possuindo apenas 10% dos registros com a classe positiva, que neste cenário de estudo são as alterações que geraram erro. Para evitar que os modelos preditivos ficassem especializados apenas na classe negativa a base de dados foi reduzida, buscando manter metade do conjunto de dados com a classe positiva e a outra metade com a classe negativa. Sendo assim o conjunto de dados utilizado no primeiro experimento contava com aproximadamente 8 mil registros. Foi aplicada a biblioteca *model_selection* para separar 10% destes dados para a validação dos modelos gerados, ficando o restante dos registros para o treinamento.

Na fundamentação teórica estão descritos oito técnicas de aprendizado de máquina, neste experimento inicial foram utilizadas as quatro primeiras técnicas estudadas: *Support Vector Machine*(SVC), árvore de decisão (*Tree*), *Random Forest* (RF) e *multilayer perceptron*(MLP). Para utilização destas técnicas de aprendizado de máquina foram criados quatro programas Python isolados para testar cada técnica individualmente porém utilizando o mesmo conjunto de dados de treinamento e validação. Os resultados obtidos nesta primeira geração dos modelos estão expostos na Tabela 3.

Tabela 3: Resultados do aprendizado de máquina

Método	**Classe Real**	**Métricas**			**Suporte**	**Matriz de confusão**	
		Precisão	**Cobertura**	**F-score**		**Predito 0**	**Predito 1**
SVC	0	0.97	0.52	0.67	727	376	351
	1	0.03	0.45	0.05	22	12	10
Tree	0	0.98	0.87	0.92	727	633	94
	1	0.06	0.27	0.1	22	16	6
RF	0	0.97	1.00	0.98	727	724	3
	1	0.25	0.05	0.08	22	21	1
MLP	0	0.97	0.95	0.96	727	690	37
	1	0.07	0.14	0.10	22	19	3

Fonte: Elaborado pelo autor

Na Tabela 3 a coluna Método especifica qual técnica de aprendizado preditivo foi utilizada para geração dos resultados. A coluna Classe Real indica o rótulo das instâncias,

sendo “0” as alterações que não geraram nenhuma inconsistência e “1” as implementações que levaram à ocorrência de inconsistências no sistema. As colunas de métricas apresentam os resultados que o modelo obteve na classificação das instâncias. A coluna suporte apresenta o total de instâncias disponíveis de cada classe no conjunto de dados de validação. Por fim, as colunas da divisão Matriz de confusão contém as quantidades de instâncias classificadas conforme o rótulo previsto e o rótulo real.

É possível observar que em geral os modelos tiveram uma precisão alta para a classe “0” porém uma precisão em torno de 5% para a classe “1”, que é a classe mais relevante para este estudo. De forma semelhante, a coluna F-score apresenta valores próximos à 1.00 para a classe negativa, o que significa que os modelos gerados previram corretamente quando uma alteração não geraria inconsistência. Em contrapartida a coluna F-score também mostra que as predições para as alterações que gerariam inconsistência foram muitas vezes equivocadas.

Buscando proporcionar mais registros de treinamento para os modelos de aprendizado foi criada uma nova base de dados específica para a validação, mantendo a base anterior de cerca de 8 mil registros completamente voltada para o treinamento. Com essa finalidade a base de validação foi criada a partir dos registros posteriores à março de 2019, que é período final da base de dados de treinamento. Sendo assim, foram utilizados registros de produção do período de abril de 2019, gerando aproximadamente 60 instâncias. A coleta destes dados ocorreu no início de maio de 2019, portanto posteriores à abril não puderam ser aproveitados.

Tabela 4: Resultados com conjunto de dados específico para validação

Método	Classe Real	Métricas			Suporte	Matriz de confusão	
		Precisão	Cobertura	F-score		Predito 0	Predito 1
SVC	0	1.00	0.32	0.49	59	19	40
	1	0.07	1.00	0.13	3	0	3
Tree	0	0.93	0.46	0.61	59	32	27
	1	0.03	0.33	0.06	3	2	1
RF	0	0.97	0.54	0.70	59	32	27
	1	0.07	0.67	0.12	3	1	2
MLP	0	1.00	0.17	0.29	59	10	49
	1	0.06	1.00	0.11	3	0	3

Fonte: Elaborado pelo autor

Submetendo novamente os modelos de aprendizado à base de treinamento e de validação foram obtidos os resultados apresentados na tabela Tabela 4. É possível observar que a precisão da classe negativa ainda é muito superior à precisão da classe positiva. Por outro lado, ao avaliar a coluna F-score, que serve como métrica para o desempenho geral das predições de uma classe, pode ser constatado que das quatro técnicas apenas uma apresentou um resultado menor nesta coluna, que foi o método de árvore de decisão

(Tree). O resultado anterior foi de 0.1 enquanto que o resultado da Tabela 4 foi de 0.06. Isso significa que os outros três modelos obtiveram um desempenho melhor com esse ajuste na base de dados, como por exemplo o método SVC, que passou do valor 0.05 na métrica F-score para 0.13 nesta mesma métrica para a classe "1" após o ajuste da base de dados.

Com este experimento inicial foi possível executar de forma simplificada todo o processo necessário para a geração de modelos preditivos. Foi confirmada a possibilidade da extração de dados através da ferramenta Oracle SQL Developer, com posterior geração de um arquivo no formado CSV como conjunto de dados. Além disso, também foi observada a viabilidade da linguagem Python para a geração de modelos preditivos a partir deste conjunto de dados. Apesar disso, os resultados obtidos neste primeiro experimento não apresentaram valores satisfatórios quanto à precisão das predições geradas.

O cenário estudado apresenta duas classes possíveis para a classificação, a classe positiva, que são alterações que geraram inconsistência e a classe negativa, que são alterações que não geraram inconsistência. Sendo assim, para ser considerado que os modelos preditivos atingiram algum nível de aprendizado sobre a base de dados é necessário que o desempenho atingido supere o nível da aleatoriedade para a predição da classe. Neste cenário de estudo, que possui duas classes possíveis para a predição, isso pode ser traduzido para o desempenho na métrica F-score, que deve apresentar um valor superior a 0.5. Nenhum dos modelos utilizados neste experimento atingiu este limiar, porém foi possível observar que ao propor alterações no conjunto de dados é possível obter resultados superiores para essa métrica. Com isso pode-se afirmar que as técnicas de mineração de dados e aprendizado de máquina são aplicáveis ao cenário de estudo e que ainda é possível obter um desempenho superior para os modelos preditivos gerados trabalhando as informações do conjunto de dados e o processo de geração destes modelos.

5 IMPLEMENTAÇÃO

No Capítulo 4 foi apresentada uma visão geral das atividades relativas à criação de modelos preditivos no cenário de estudo deste trabalho. Através dessa análise inicial foi possível observar que a geração de modelos preditivos é viável, ainda que nesta fase inicial sejam obtidos valores baixos para a precisão da classe positiva, que é de maior interesse. Nas seções do Capítulo 5 são apresentados os procedimentos práticos adotados visando obter resultados melhores na predição da ocorrência de inconsistências nas alterações. Também são destacados as medidas tomadas para a extração de dados e normalização das informações, assim como as ações feitas para garantir a confiabilidade das validações e métricas avaliadas.

5.1 PROCEDIMENTOS DE EXTRAÇÃO DE DADOS

Os principais dados de referência para a construção de modelos preditivos neste estudo estão registrados na ferramenta Sicla que, conforme citado na seção 2.3, é um sistema interno utilizado pela Rech Informática para controle das suas atividades e registro das diversas operações realizadas durante a jornada de trabalho. A consulta de forma manual dessas informações mostra-se inviável, uma vez que serão necessárias informações de centenas de alterações feitas. Em função disso, para a extração de informações deste sistema foi utilizada a ferramenta Oracle SQL Developer, que possibilita que os dados sejam extraídos de forma consolidada através de filtros.

A ferramenta Sicla contém diversas informações de clientes da empresa Rech Informática além de outras informações de uso interno, portanto para preservar a confidencialidade destes dados nenhuma imagem do sistema Sicla está exposta neste estudo. Da mesma forma, os dados utilizados para conexão no banco de dados do Sicla também são omitidos e os comandos SQL apresentados possuem nomes apenas ilustrativos para os campos e tabelas de cada seleção.

Assim pôde ser realizada a extração de dados do sistema Sicla, sendo utilizado para isso o comando SQL ilustrado na Figura 21. Na linha 2 é extraído o campo NUMERO, que é o número identificador único de cada ficha, utilizado apenas para conferência das demais informações, não sendo utilizada para o aprendizado de máquina. Nas linhas 3 a 6 é feita uma sub-consulta para a tabela RNC, que é o Registro de Não Conformidade do Sicla. Nesta coluna são filtradas todas as inconsistências que apontam para o número de ficha corrente, sendo totalizadas através do comando GROUP BY. A coluna ERROS contém, portanto, a quantidade total de inconsistências geradas pela ficha específica. Ainda nesta coluna é aplicada a função COALESCE, pois em muitos casos não existem dados na tabela

Figura 21: Comando SQL para extração da base de dados

```
1   SELECT
2       F.CMP_NUM AS NUMERO,
3       COALESCE((SELECT COUNT(*)
4           FROM TAB_RNC R
5           WHERE R.CMP_FORIGEM = F.CMP_NUM
6           GROUP BY R.CMP_FORIGEM), 0) AS ERROS,
7       F.CMP_PROG AS PROGRAMADOR,
8       COALESCE(F.CMP_REV, 0) AS REVISOR,
9       F.CMP_RESP AS RESPONSAVEL,
10      F.CMP_CEL AS CELULA,
11      F.CMP_TIP AS TIPO,
12      F.CMP_TPRO AS MIN_PRODUCAO,
13      F.CMP_TREV AS MIN_REVISAO,
14      F.CMP_TVAL AS MIN_VALIDACAO,
15      COALESCE(F.CMP_TRET, 0) AS RETRABALHO,
16      F.CMP_PRG AS COD_PROGRAMA,
17      LENGTH(F.CMP_DESC) AS DESCRICAO,
18      LENGTH(F.CMP_OBS) AS OBSERVACAO
19      (COALESCE((SELECT RN.CMP_MINTOT
20          FROM ITEMPED RN
21          WHERE RN.CMP_CODIGORN = F.CMP_CODIGORN AND RN.
                CMP_SEQRN = F.CMP_SEQRN), 0)) -
22          (F.CMP_TPRO + F.CMP_TREV) AS MIN_ANALISE
23  FROM TAB_FICHA F
24  WHERE F.CMP_DATCRIACAO > Date '2015-08-03' AND
25      F.CMP_DATCRIACAO < Date '2019-03-31' AND
26      F.STATUS NOT IN( 0, 1, 2, 3, 4, 5, 9);
```

Fonte: Elaborado pelo autor

RNC que apontam para a ficha corrente. Nestes casos a coluna apresentaria a informação "null", que indica ausência de conteúdo. A função COALESCE faz a substituição do conteúdo vazio pelo valor alternativo definido nos parâmetros da função, neste caso, o valor zero.

Ainda na Figura 21 é extraída a coluna PROGRAMADOR na linha 7. Essa informação diz respeito ao código de identificação do colaborador da Rech que implementou as alterações no código-fonte. Da mesma forma, na linha 8 é obtido o campo REVISOR, que identifica o código do colaborador responsável pela revisão técnica das alterações. Nesta coluna também foi necessário aplicar a função COALESCE, pois algumas alterações não apresentam registro de revisão. Na linha 9 é selecionado o campo RESPONSAVEL, que identifica o código do colaborador que validou a alteração, realizando o teste funcional no sistema. O campo CELULA extraído na linha 10 contém a letra de identificação interna da área de negócio responsável pela alteração. Se a alteração foi, por exemplo, feita no módulo de Vendas e Faturamento este campo trará o dado "N", que é a letra de identificação da área "Negócio", responsável pelo módulo. Na linha 11 é extraído o campo TIPO, que é um campo numérico com a classificação que a alteração recebeu quanto a

principal motivação da necessidade. Neste caso o campo possui um número que identifica necessidades como as seguintes: implementação, correção de erro, alteração de legislação, impede processo, entre outros. Os campos MIN_PRODUCAO, MIN_REVISAO e MIN_VALIDACAO, obtidos respectivamente nas linhas 12, 13 e 14 do comando SQL contém o tempo decorrido em minutos para a execução das etapas de produção, revisão e validação da alteração. Na linha 15 é extraído o campo RETRABALHO, que identifica se na alteração específica foi encontrada alguma falha nos processos de revisão ou validação que demandou algum ajuste no código-fonte que havia sido alterado. A função COALESCE também foi aplicada nesta coluna pois muitas das alterações não possuem conteúdo neste campo.

Na Figura 21, a coluna COD_PROGRAMA, obtida na linha 16, apresenta o código de identificação do principal programa ou código-fonte alterado pela Ficha. Existem programas que são mais utilizados no sistema e por consequência disto passam por mais alterações. Acredita-se que essa informação será relevante para o aprendizado de máquina pois se um programa é submetido a mais alterações ele está mais suscetível a apresentar falhas. Além destes campos, nas linhas 17 e 18 são extraídos os campos DESCRICAO e OBSERVACAO, que contém o texto descritivo da alteração necessária no sistema e o texto do que foi efetivamente feito para atender a demanda, respectivamente. Em ambos os casos a função LENGTH é aplicada para que o conteúdo textual seja representado apenas através de um valor numérico que reflete a quantidade de caracteres deste texto. Estas duas informações foram selecionadas prevendo que pode existir relação entre alterações descritas de forma muito breve e inconsistências geradas pelas mesmas.

Adicionalmente na extração dos dados das alterações buscou-se extrair também o tempo de análise da alteração. Conforme os procedimentos apresentados na seção 2.3, este tempo corresponde ao tempo que foi investido para entender a necessidade, esclarecer eventuais dúvidas e redigir uma solução através do registro de uma Ficha. Este dado não está disponível diretamente na tabela de fichas do Sicla, sendo necessário consultar a tabela de RNS para obter essa informação. Esse procedimento está sendo feito nas linhas 19, 20, 21 e 22 da Figura 21, sendo gerada a coluna MIN_ANALISE. Uma particularidade deste dado é que a RNS possui apenas a informação do tempo total realizado, sem discriminar em específico o tempo de análise. Portanto para obter esse tempo de análise foi necessário descontar do total o tempo de produção e de revisão que estão embutidos nele.

Por fim, na Figura 21 as linhas 24, 25 e 26 especificam os filtros que delimitam as fichas que serão obtidas por esse comando SQL. Está sendo avaliado o conteúdo do campo F.CMP_DATCRIACAO, que contém a data de criação da Ficha. Foram selecionadas as fichas somente criadas após a data de 03 de agosto de 2015 pois nesta data houve a mudança oficial do processo de produção das alterações, sendo adotado o Scrum. A

utilização de informações de alterações produzidas antes dessa data não seria relevante para o aprendizado de máquina, uma vez que essas informações se referem a um processo diferente do cenário atual, no qual serão geradas as predições nas próximas etapas deste estudo. Esta extração de dados ocorreu no mês de maio de 2019, sendo os registros deste mês descartados. Estas fichas, por se tratarem de alterações mais recentes, poderiam não estar totalmente concluídas ou ainda não terem sido disponibilizadas para o ambiente dos clientes, o que certamente diminui as oportunidades em que poderia ser identificada e apontada uma inconsistência na alteração. A data de criação da Ficha foi delimitada no comando SQL até o dia 31 de março de 2019 para que as Fichas implementadas após essa data, durante o mês de abril, pudessem ser utilizadas para a validação do aprendizado gerado. De forma semelhante, o campo F.STATUS, que representa a posição atual da Ficha, está sendo filtrado para excluir da seleção as alterações que estão em situações intermediárias da implementação, ou seja, que ainda não passaram por todo o processo produtivo.

O código SQL exposto foi utilizado para gerar um arquivo CSV contendo todas as informações selecionadas. A seleção do período de 03 de agosto de 2015 até 31 de março de 2019 resultou em 38.745 amostras para o aprendizado de máquina. Após extraído este conjunto de dados, foi feita uma nova seleção para o conjunto de validação, que contava com 67 amostras de fichas implementadas entre as datas de 01 de abril de 2019 até 30 de abril de 2019.

5.2 PRÉ-PROCESSAMENTO DE DADOS

Após executada a extração de dados e gerado um arquivo CSV contendo todas as informações, este conjunto foi analisando buscando otimizações que pudessem ser realizadas a fim de tornar a geração de conhecimento sobre a base mais rápida e precisa. A primeira questão observada foi se existiam dados não numéricos no conjunto de dados. Dadas as colunas extraídas na Figura 21 existe apenas uma que apresenta seu conteúdo de forma textual, que é a coluna de CELULA. Esta informação refere-se à área de negócio responsável pela alteração, representando áreas como produção, recursos humanos, livros fiscais, financeiro, entre outras. O campo CELULA representa essa informação através de um caractere único, como por exemplo “N”, “P”, “I” ou “R”, sendo cada caractere associado a uma área de negócio específica. Para este processo foi utilizado o editor de códigos VSCode, que permite utilizar recursos como expressão regular para editar o conteúdo de arquivos (Visual Studio Code, 2019).

O primeiro ajuste realizado utilizando o editor VSCode foi a busca e troca do conteúdo da coluna CELULA. Foram feitas substituições como as seguintes: “I” substituído por 1 e “N” substituído por 2. De forma semelhante, foi necessário ajustar a coluna de ERROS, pois esta é a coluna que contém o alvo da predição. Como explicado na seção 3.1,

para este estudo são utilizados modelos preditivos de classificação, portanto cada valor da coluna alvo da predição significa também uma classe possível para a predição. Os valores da coluna ERROS são crescentes de 0 até quantidades maiores como 15 ou 20 erros gerados por uma única ficha. Com estes valores para o atributo classe os modelos criados após o aprendizado de máquina gerariam predições para 20 classes ou tantas quanto fosse o maior valor da coluna ERROS, pois cada valor seria interpretado como uma classe nova. Sendo assim, foi utilizado o editor VSCode para substituir todas as quantidades de erros pelo valor padrão "1", que passou a representar que a ficha gerou alguma inconsistência, independentemente da quantidade. Tendo essa coluna apenas dois valores, "0" e "1", os modelos preditivos farão as predições para uma destas duas classes.

Outra questão observada na base de dados extraída foi a distribuição dos dados entre a classe positiva, que são as alterações que geraram alguma inconsistência, e a classe negativa, que são as alterações que não geraram inconsistência. No conjunto de dados de treino haviam 38.745 amostras de alterações, sendo que deste total 4.553 eram referentes à alterações que geraram inconsistência. Isso significa que a base de dados é composta por aproximadamente 89% de registros da classe negativa, o que tornaria os modelos preditivos mais especializados nesta classe e poderia diminuir a precisão das predições para a classe positiva, que é a classe mais relevante para esse estudo. Para tornar o aprendizado mais equilibrado e promover mais destaque para a classe positiva o conjunto de dados foi balanceado de forma que para cada amostra da classe positiva existisse uma amostra da classe negativa, mantendo a relação de 1:1 entre as duas classes. Com isso, o conjunto de dados para ao treinamento passou a contar com 9.106 registros totais, contendo o mesmo número de amostras para a classe positiva e para a classe negativa.

5.3 ALGORITMOS DE APRENDIZADO DE MÁQUINA

Com o conjunto de dados de treinamento e de validação devidamente preparados foi possível retomar a geração de modelos preditivos utilizando a linguagem de programação Python. Na seção 3.2 estão expostos oito técnicas de aprendizado de máquina, porém neste primeiro momento da implementação foram utilizados quatro destas técnicas, com o intuito de simplificar os testes e avaliação dos resultados após as modificações na base de dados. Sendo assim, os modelos gerados utilizaram as seguintes técnicas: SVC, *Decision tree*, *Random Forest* e MLP.

No Capítulo 4 haviam sido feitos experimentos com quatro das técnicas de aprendizado preditivo de forma separada, sendo utilizado para cada uma um programa Python específico que reconhecia a base de dados e gerava o modelo necessário. No entanto, este formato fazia com que fosse codificados alguns tratamentos de forma repetitiva nos quatro programas. Além disso, a execução destes programas também demandava mais tempo, pois era necessário executar quatro vezes o processo de leitura dos dados no conjunto de

treinamento e de validação. Buscando simplificar esse processo foi criado um único programa Python responsável por ler os conjuntos de dados de treinamento e de validação utilizando os mesmos através das quatro técnicas de aprendizado de máquina selecionadas.

Figura 22: Programa Python: Import e leitura de arquivo CSV

```
1   import os
2   import pandas as pd
3   import time
4   import datetime
5   from sklearn import model_selection
6   from sklearn import preprocessing
7   from sklearn.compose import ColumnTransformer,
        make_column_transformer
8   from sklearn.metrics import classification_report,
        confusion_matrix
9
10  #Lê conjunto de dados
11  os.chdir("F:/!TC/MachineLearning/")
12  dataset = pd.read_csv("1-dataset.csv")
```

Fonte: Elaborado pelo autor

A Figura 22 apresenta as primeiras instruções do programa desenvolvido. Inicialmente são realizados os *imports*, que declaram as bibliotecas que serão utilizadas ao longo da rotina. A primeira biblioteca `os` refere-se sistema operacional (*Operational System*) e é utilizada para localizar o arquivo do conjunto de dados de treinamento. A biblioteca Pandas permite abrir e manipular dados rotulados, como é o caso do arquivo CSV contendo o *dataset*. As bibliotecas `time` e `datetime` são utilizadas apenas para apurar o tempo decorrido no processo de geração dos modelos. Por fim, as demais bibliotecas referem-se às rotinas do pacote *scikit-learn*, que são utilizadas para preparação dos dados e geração dos modelos preditivos.

Figura 23: Programa Python: Preparação dos dados

```
1   #Remove colunas desnecessárias do conjunto de dados de treino
2   y = dataset['ERROS']
3   X_temp = dataset.drop('ERROS', axis=1)
4   X = X_temp.drop('NUMERO', axis=1)
5
6   #Ajusta dados de treino
7   Y_ajustado = y
8   X_temp = X
9   min_max = preprocessing.MinMaxScaler()
10  X_ajustado = min_max.fit_transform(X_temp)
11  X_train = X_ajustado
12  y_train = Y_ajustado
```

Fonte: Elaborado pelo autor

A Figura 23 contém o trecho de código utilizado para preparação dos dados lidos no campo `dataset`. O primeiro procedimento realizado é a separação da coluna ERROS em um conjunto de dados próprio. Essa informação é destaca das demais pois os algoritmos de aprendizado de máquina funcionam recebendo de forma separada a informação que deve ser predita e os dados associados à ela. Além deste ajuste, também foi eliminada a coluna NUMERO dos atributos restantes do *dataset*. Isso foi feito pois esta coluna contém o número do registro de alteração, que é um identificador único da ficha. Essa informação foi utilizada apenas para conferência do *dataset*, avaliando se as informações contidas no arquivo CSV gerado correspondem as informações que podem ser consultadas visualmente na ferramenta Sicla. Sendo assim, essa coluna foi eliminada dos atributos, pois não contém nenhuma informação relevante para o aprendizado de máquina. Além disto, foi aplicada a função `MinMaxScaler` nos dados do conjunto de treinamento. Esta função serve para ajustar os dados de cada atributo para uma escala de valores menor, tipicamente entre 0 e 1. Isso auxilia os algoritmos de aprendizado de máquina a convergirem para uma resposta mais rapidamente, pois passam a trabalhar com dados em uma grandeza menor. Por fim, neste trecho de código são obtidos os campos `X_train` e `y_train`, que são os atributos de cada instância do *dataset* e a classe real que deve ser prevista pelos modelos gerados, respectivamente. Esse mesmo procedimento também é realizado para o outro conjunto de dados de validação, através do arquivo "1-dataset_validacao.csv".

Figura 24: Programa Python: Preparação dos dados

```
1       from sklearn import svm
2
3       #Registra tempo inicial
4       start = time.time()
5
6       #Gera modelo preditivo SVC
7       svclassifier = svm.SVC(kernel='linear').fit(X_train, y_train)
8
9       #Registra tempo final
10      print("Tempo fit SVC")
11      print(time.time() - start)
12
13      #Faz teste
14      y_pred_val = svclassifier.predict(X_test)
15
16      #Gera estatísticas da classificação
17      print("Resultados SVC")
18      print(confusion_matrix(y_test,y_pred_val))
19      print(classification_report(y_test,y_pred_val))
```

Fonte: Elaborado pelo autor

O código responsável pela geração do aprendizado de máquina está exposto na Figura 24. A primeira linha de código executa a declaração do pacote `svm`, que contém as rotinas relativas à técnica de *Support Vector Machine* para classificação de dados, ou

SVC. Na linha 4 é feito o registro do horário no qual foi iniciado o treinamento do modelo, que será utilizado depois para computar o tempo total decorrido. Na linha 7 é feito treinamento utilizando o conjunto de dados específico de treino. Isso é feito através do método `fit`, que recebe os atributos das instâncias, contidos no campo `X_train`, e os rótulos atribuídos para cada instância, contidos na variável `y_train`. Após este procedimento, é realizada a aferição do tempo decorrido no treinamento, nas linhas 10 e 11. A utilização modelo gerado é feita na linha 14, na qual é utilizado o método `predict` para gerar predições para o conjunto de instâncias de validação, presentes no campo `X_test`, sendo as classificações resultantes desse teste armazenadas no campo `y_pred_val`. Por fim, são geradas estatísticas sobre as predições realizadas no conjunto de validação comparando-se as classificações originais já sabidas do conjunto de instâncias de validação, do campo `y_test`, com as classificações geradas pelo modelo preditivo, do campo `y_pred_val`.

Figura 25: Programa Python: Preparação dos dados

```
1   #Cria modelo de árvore de classificação
2   from sklearn import tree
3   treeclassifier = tree.DecisionTreeClassifier(criterion='gini')
4   treeclassifier.fit(X_train, y_train)
5   y_pred_val = treeclassifier.predict(X_test)
6
7   #Cria modelo de floresta aleatória
8   from sklearn.ensemble import RandomForestClassifier
9   rfc = RandomForestClassifier(n_estimators=200)
10  rfc.fit(X_train, y_train)
11  y_pred_val = rfc.predict(X_test)
12
13  #Cria modelo MLP
14  from sklearn.neural_network import MLPClassifier
15  mlp = MLPClassifier(hidden_layer_sizes=(12,12),max_iter=500)
16  mlp.fit(X_train,y_train)
17  y_pred_val = mlp.predict(X_test)
```

Fonte: Elaborado pelo autor

A Figura 25 apresenta os trechos de código específicos para os demais modelos de aprendizado utilizados. Da linha 2 à 5 é feito o treinamento e predição utilizando o algoritmo de árvore de decisão, acionando métodos `fit` e `predict` da mesma forma que havia sido feito para o método SVC. No trecho da linha 8 até a linha 11 é realizado o treinamento e predição com modelo de *Random Forest*. Já nas linhas de 14 até 17 é utilizado o modelo de *multilayer perceptron*. Os resultados obtidos neste experimento podem ser observados na Tabela 5.

Avaliando os resultados da tabela Tabela 5 observa-se que não houve aumento significativo nos valores obtidos para a coluna F-score, com os valores mantendo-se abaixo do nível de aleatoriedade de 0.5 que foi discutido na seção 4.2. Uma outra questão que se destaca nestes resultados é a distribuição das predições na matriz de confusão em todos os

Tabela 5: Resultados com coluna de tempo de análise

Método	Classe Real	Métricas Precisão	Cobertura	F-score	Suporte	Matriz de confusão Predito 0	Predito 1
SVC	0	0	0	0	61	0	61
	1	0.09	1.00	0.16	6	0	6
Tree	0	0.91	0.16	0.28	61	10	51
	1	0.09	0.83	0.16	6	1	5
RF	0	1.00	0.08	0.15	61	5	56
	1	0.10	1.00	0.18	6	0	6
MLP	0	0	0	0	61	0	61
	1	0.09	1.00	0.16	6	0	6

Fonte: Elaborado pelo autor

modelos. Em geral as predições geradas concentraram-se na classe positiva, existindo casos mais extremos como dos modelos SVC e MLP em que nenhuma instância foi predita para a classe negativa. Esse fenômeno ainda pode ser observado nos modelos restantes de árvore de decisão (Tree) e *Random Forest* (RF), pois em ambos os casos foram feitas poucas predições para a classe negativa "0". Isso gerou uma precisão alta para esses modelos na classe negativa, de 0.91 e 1.00 respectivamente, porém também se observa que a cobertura obtida foi baixa, com valores de 0.16 e 0.08 para os modelos Tree e RF da Tabela 5.

5.4 ALGORITMOS PARA BALANCEAMENTO

Em relação ao primeiro experimento realizado, que está descrito no Capítulo 4, houve a adição da coluna de tempo de análise ao conjunto de dados, o que fez com que fosse necessário realizar novamente os procedimentos de extração da base de dados e balanceamento das amostras para cada classe. Apenas a adição da coluna de tempo de análise não justifica a mudança dos modelos preditivos, que passaram fazer classificações que tendem muito mais para a classe negativa.

Considerando a mudança que houve nos resultados, uma questão que pode ser observada é a execução do processo de extração e balanceamento da base de dados, que foi relatado na seção 5.2. Para evitar que os modelos gerados ficassem especializados apenas na classe negativa, que possui muito mais amostras que a classe positiva, buscou-se equilibrar a base de dados mantendo uma amostra da classe negativa para cada amostra da classe positiva. Isso significa que foi necessário remover da base de dados várias amostras da classe negativa, mantendo-se apenas uma quantidade equivalente ao total de amostras da classe positiva. Estas amostras da classe negativa foram selecionadas de forma a preservar pelo menos um registro de alteração de cada programador presente na base de dados. Enquanto é possível afirmar que as amostras de alterações que geraram inconsistências são as mesmas entre os dois experimentos realizados, o mesmo não pode ser afirmado para as amostras da classe negativa, pois elas passaram por um processo ma-

nual de seleção. Como esse processo manual não garante que foram utilizadas as mesmas amostras da classe negativa, é possível que as amostras utilizadas neste novo experimento tenham menos significado para os processos de aprendizado de máquina do que as amostras utilizadas no primeiro experimento, tornando o reconhecimento de padrões para as alterações que não geram inconsistência mais difícil.

Visando automatizar o processo de balanceamento da base de dados foi então optado por criar programas Python voltados especialmente para essa tarefa. Dessa forma a execução desta etapa seria mais dinâmica e ágil. Ao adicionar novas colunas no conjunto de dados esses programas também garantiriam que seriam selecionadas as mesmas instâncias, pois ao contrário do processo manual, a execução do programa ocorrerá sempre da mesma maneira. Para a geração e avaliação dos modelos preditivos é necessário gerar duas base de dados separadas: uma para treinamento e outra para validação. Sendo assim, como as bases de dados possuem propósitos diferentes e são construídas de forma a atender esse propósito julgou-se relevante criar programas separados para a criação das duas bases de dados balanceadas.

Figura 26: Programa Python: Balanceamento da base de treinamento simplificado

```
1     #Para todo programador reconhecido no dataset
2         for row in csv_original:
3             if "1" in erro:
4                 writer.writerow(row)
5                 prog_count = prog_count + 1
6             if "0" in erro:
7                 semerro.append(row)
8         #Se possui alterações com erro e sem erro
9         while prog_count > 0 and len(semerro) != 0:
10            listarestante = []
11            ultimorevisor = "0"
12            #Percorre as fichas registradas que não tiveram erro
13            for alteracao in semerro:
14                revisor = alteracao["COD_REVISOR"]
15                if revisor != ultimorevisor and prog_count != 0:
16                    writer.writerow(alteracao)
17                    prog_count = prog_count - 1
18                else:
19                    listarestante.append(alteracao)
20                ultimorevisor = revisor
21            semerro = listarestante[:]
```

Fonte: Elaborado pelo autor

A Figura 26 apresenta de forma simplificada a lógica e respectivas instruções do programa Python criado para geração da base de dados balanceada para treinamento dos modelos preditivos. O objetivo que precisa ser alcançado por esse programa é que sejam preservadas as amostras de fichas que geraram inconsistência coletando-se também um número equivalente de amostras de fichas que não geraram inconsistência. Para isso,

o trecho inicial do programa realiza a abertura de dois arquivos CSV, sendo um o arquivo original da base de dados e outro o arquivo de saída com a base já balanceada. O arquivo original está previamente ordenado conforme o código do programador que implementou as alterações. Aproveitando-se essa ordenação o arquivo é então processado identificando-se quando ocorre a troca de um programador para avaliar os registros que foram processados para o programador. Essa lógica está sinalizada na Figura 26 através do comentário da linha 1 para simplificar e reduzir a representação do código.

A Figura 26 apresenta no trecho das linhas 2 até 7 o processamento dos registros de alterações de cada programador da base de dados. Caso a alteração tenha gerado um erro o registro é repassado diretamente para o arquivo de saída incrementando-se o contador da quantidade de erros reconhecidos para o programador atual, nas linhas 4 e 5. Já no caso oposto, no qual a alteração não gerou nenhuma inconsistência, a alteração não vai para o arquivo de saída, ficando apenas na lista auxiliar `semerro`, na linha 7. No laço iniciado na linha 9 é são processados os registros de alterações sem inconsistência da lista `semerro`. A cada iteração é selecionada uma alteração de cada revisor diferente presente nos registros de alteração até que seja atingida a mesma quantidade de alterações com erro, contida no campo `prog_count`. Esse tratamento foi feito para garantir maior diversidade de alterações nos registros selecionados, evitando que as amostras de alterações realizadas por um programador sejam também todas revisadas por um único revisor. Neste processamento a lista contida em `listarestante` é utilizada como auxiliar para armazenar as alterações de revisores repetidos ou alterações que excederiam a quantidade limite. O comando contido na linha 16 é responsável por escrever no arquivo de saída com a base de dados balanceada que será utilizada nos treinamentos dos modelos preditivos.

Da mesma forma também foi desenvolvido um segundo programa para geração de balanceamento na base de dados, porém este outro programa tem por objetivo a seleção de registros para o processo de validação dos modelos gerados. Para o conjunto de dados de treinamento a necessidade era que todas as amostras de fichas que geraram erro fossem preservadas e que a base balanceada possuísse registros diversificados. Já para o conjunto de dados de validação a principal necessidade é que a base fosse reduzida, ainda representando a base de dados original, porém com uma distribuição equivalente entre a classe positiva e a classe negativa. Com esse propósito foi criado o programa representado na Figura 27.

Nas linhas 1, 2 e 3 do programa da Figura 27 são declarados campos auxiliares para a geração da base de validação. Entre eles está o `limite`, que define a quantidade de amostras de alterações que serão coletadas de cada classe para cada programador. Na linha 5 é iniciada a varredura do arquivo CSV original que contém todas as amostras de alterações extraídas do período selecionado para validação. Na linha 6 é obtida a informação da coluna que indica se a alteração gerou alguma inconsistência. Está informação é utilizada nas

Figura 27: Programa Python: Balanceamento da base de validação simplificado

```
1       fichacomerro = 0
2       fichasemerro = 0
3       limite = 5
4       ultimoprogramador = 0
5       for row in csv_original:
6           erro = row["ERROS"]
7           programador = row["COD_PROGRAMADOR"]
8           # Se trocou de programador
9           if ultimoprogramador != 0 and ultimoprogramador !=
                programador:
10              # Atualiza último programador
11              ultimoprogramador = programador
12              fichacomerro = 0
13              fichasemerro = 0
14          if "1" in erro and fichacomerro < limite:
15              writer.writerow(row)
16              fichacomerro = fichacomerro + 1
17          if "0" in erro and fichasemerro < limite:
18              writer.writerow(row)
19              fichasemerro = fichasemerro + 1
20          ultimoprogramador = programador
```

Fonte: Elaborado pelo autor

linhas 14 e 17 para definir qual contador auxiliar deve ser incrementado: `fichacomerro`, que contabiliza as alterações que geraram inconsistência ou `fichasemerro`, que armazena a quantidade de alterações que não geraram uma inconsistência. As amostras de alterações das classes positiva e negativa são então selecionadas até que se atinja a quantidade estipulada de registros definida no campo `limite`. Na linha 7 é obtido o código do programador que implementou a alteração, para que este código possa ser avaliado na linha 9 identificando quando ocorre uma troca do programador das alterações. Quando ocorre essa troca os contadores auxiliares são inicializados nas linhas 12 e 13 permitindo que novos registros do próximo programador sejam selecionados. Nas linhas 15 e 18 os registros selecionados são gravados no arquivo CSV de saída que possui a base de dados de validação balanceada e reduzida.

Uma vez que estes programas garantem a montagem equilibrada e diversificada das bases de dados foi observada a possibilidade de utilizar mais registros para ambas as bases de treinamento e validação. Sendo assim, foi extraída uma base de dados inicial com registros de alteração decorrentes do período de 3 de agosto de 2015 até 22 de maio de 2019, utilizando o comando SQL apresentado na seção 5.1. Esta extração gerou 38745 amostras de alterações que foram submetidas aos programas Python desenvolvidos para montagem das bases de treinamento e validação. Estas duas bases foram ajustadas conforme os procedimentos descritos na seção 5.2 que trata sobre pré-processamentos da base de dados. Após passar por estes procedimentos a base de dados de treinamento ficou

com 9108 amostras de alterações. Já a base de dados de validação ficou com 937 amostras totais, das quais 502 são alterações que não geraram inconsistência e as 435 restantes são alterações que pertencem à classe positiva, das alterações que geraram algum erro. Essa diferença na quantidade de classes se deve ao fato de que o programa de geração da base de validação busca preservar dados de todos os programadores buscando pelo menos 5 alterações da classe positiva e 5 da classe negativa. Porém nem todos os programadores possuem 5 ocorrências da classe positiva, e sendo assim contribuem com mais amostras da classe negativa do que da classe positiva.

Tabela 6: Resultados com algoritmos de balanceamento

Método	Classe Real	Métricas			Suporte	Matriz de confusão	
		Precisão	Cobertura	F-score		Predito 0	Predito 1
SVC	0	0.54	1	0.70	502	502	0
	1	0	0	0	435	435	0
Tree	0	0.64	0.77	0.70	502	386	116
	1	0.65	0.50	0.57	435	216	219
RF	0	0.69	0.96	0.80	502	484	18
	1	0.92	0.49	0.64	435	220	215
MLP	0	0.59	0.99	0.74	502	497	5
	1	0.95	0.20	0.34	435	346	89

Fonte: Elaborado pelo autor

Os resultados obtidos com as bases de dados geradas através dos programas de balanceamento dos dados estão expostos na Tabela 6. Em relação aos resultados obtidos na Tabela 5, observa-se que as predições na matriz de confusão passaram a estar mais distribuída entre as duas classes possíveis. Isso confirma que a base de dados agora está proporcionando um aprendizado mais equilibrado e consistente com relação as duas classes. Pode ser observada também a coluna de precisão, que na Tabela 5 se concentrava sempre em uma das classes, ficando a classe oposta com precisão próxima à zero. Já nos resultados da Tabela 6 a precisão dos modelos gerados passou a ser mais equilibrada, com exceção do modelo SVC, que teve precisão zero para a classe positiva.

Os valores obtidos na coluna F-score da Tabela 6 resumem os resultados obtidos através da geração de bases utilizando programas específicos para essa finalidade. No experimento anterior, da seção 5.3, os valores obtidos estavam no intervalo de 0 a 0.28, que indica um nível baixo de aprendizado sobre a base de dados. Foi então eliminado o processo manual de seleção de registros, que trazia aleatoriedade para esta etapa poderia fazer com que as amostras selecionadas para o conjunto de dados tivessem pouca relevância para o aprendizado de máquina. Com a aplicação da seleção automatizada de registros para as bases de dados pôde ser observado um aumento dos valores obtidos para a métrica F-score na Tabela 6. Em geral o valores se concentraram na faixa de 0.37 a 0.80, com exceção novamente das predições para a classe positiva “1” no modelo SVC. Isso significa

que em geral os modelos obtiveram um nível de aprendizado maior para as bases de dados geradas automaticamente.

5.5 AVALIAÇÃO COM BASE DE DADOS DE ESTUDO

A utilização de programas específicos para montagem das bases de dados apresentada na seção 5.4 trouxe um desempenho maior para a geração de modelos preditivos, porém os resultados obtidos ainda não são totalmente consistentes, uma vez que o modelo SVC obteve precisão com valor zero para a classe positiva. Avaliando-se esses resultados foram levantadas duas hipóteses para melhorar o desempenho da geração de modelos preditivos: uma quanto às técnicas de aprendizado utilizadas e outra quanto ao significado presente nos dados coletados do cenário de estudo.

A primeira questão a ser tratada são as técnicas de aprendizado de máquina utilizadas, que até este momento do estudo são quatro: *Support Vector Machine* (SVC), árvore de decisão (Tree), *Random Forest* (RF) e *multilayer perceptron* (MLP). Durante as pesquisas do referencial teórico não foram encontrados estudos que tratassem da geração de modelos preditivos para identificação de alterações que gerariam alguma inconsistência. Sendo assim, não é possível afirmar que as técnicas de aprendizado preditivo selecionadas são as que melhor se adéquam ao caso de estudo. Para proporcionar uma maior variedade de técnicas a abordagem feita para este caso foi selecionar mais técnicas de aprendizado de máquina que pudessem ser aproveitadas neste estudo, fornecendo novos resultados que poderiam ser comparados com os demais. Dessa forma foram estudadas mais quatro técnicas de aprendizado que foram incorporadas ao referencial teórico: *Gradient Boosting Classifier*, Regressão logística, *K-Nearest Neighbors* e Naive Bayes. A adição destas técnicas procura trazer mais diversidade para o estudo, com a intenção de identificar um modelo preditivo que melhor interpreta as informações da base de dados de alterações em sistema de gestão.

Outra questão avaliada foi o significado dos dados percebidos pelos algoritmos de aprendizado de máquina. Considerando um cenário ruim, os dados selecionados para a base de dados não tem significado para descrever e diferenciar as classes positiva e negativa deste problema de classificação, o que faria os modelos gerados não atingirem valores altos para a métrica de F-score. Porém, outra possibilidade com relação à geração dos modelos preditivos é que os parâmetros e implementação utilizados não estejam favorecendo o processamento adequado da base de dados. Para avaliar o processamento das bases de dados optou-se por utilizar uma base de dados de estudo, que já fosse conhecidamente viável para classificação através de aprendizado de máquina. Uma vez que os algoritmos obtenham precisão elevada para a base de estudo poderia ser aplicados os mesmos algoritmos para a base de dados de inconsistências com a garantia de que as técnicas de aprendizado de máquina em si estão aplicadas corretamente.

Para buscar bases de dados de estudo foi utilizado o repositório de *dataset* da Universidade da Califórnia em Irvine, que é amplamente utilizado em estudos que envolvem *machine learning* (DUA; GRAFF, 2017a). A base de dados de inconsistências utilizada até este momento do trabalho possui um atributo de classe, com dois valores possíveis e treze atributos da instância que descrevem a alteração. Para que os resultados obtidos através da base de estudo pudesse ser comparados com os resultados da base de inconsistência procurou-se por um *dataset* que contivesse um número semelhante de atributos e de classes preditas.

Para efetuar testes dos algoritmos de aprendizado de máquina foi selecionada a base de dados *Wine Data Set* (DUA; GRAFF, 2017b). Esta base possui resultados da análise química de vinhos cultivados na Itália, em três regiões diferentes. O atributo de classe identifica qual a origem do vinho entre as três regiões que tiveram os dados coletados, portanto neste cenário de estudo existem três classificações possíveis para cada instância. Já os atributos de instância apresentam informações como intensidade da cor, nível de acidez, quantidade de magnésio, quantidade de flavonóides, entre outros dados que formam um total de doze atributos. Esta base de dados de vinhos possui portanto três classes possíveis com doze atributos descritivos e se assemelha à base de dados de inconsistências, que possui duas classes e treze atributos descritivos. O número de instâncias presentes no *dataset* de vinhos é de 178 e segundo Dua e Graff (2017b) esta base de dados é apropriada para testes de classificação, uma vez que as classes apresentam dados característicos nos atributos, o que contribui para que seja identificado o rótulo mais adequado para cada instância. Para utilizar a base *Wine Data Set* foi importada a biblioteca `datasets` do *sckitlearn*, que contém dados para acesso de várias bases de dados. Foi acionada a função `datasets.load_wine()` desta biblioteca para carregar a base de dados desejada de informações de vinhos.

Figura 28: Programa Python: Dicionário de técnicas de aprendizado

```
1       dict_classifiers = {
2           "Decision Tree":
3                   {'classifier': tree.DecisionTreeClassifier(),
4                       'params': [{'max_depth':[3,None]}]
5                   },
6           "Naive Bayes":
7                   {'classifier': GaussianNB(),
8                       'params': {}
9                   }
10      }
```

Fonte: Elaborado pelo autor

Para realizar os testes com a base de estudos procurou-se estruturar novamente o programa Python de geração de modelos preditivos utilizando técnicas de *machine learning*, de forma que todas as técnicas de aprendizado fossem aplicadas em uma execução

do programa, evitando que fosse necessário criar e executar um programa específico para cada técnica que é utilizada neste estudo. Primeiramente os modelos de aprendizado de máquina foram organizados em uma estrutura de dicionário, de forma que cada literal de identificação da técnica ficasse vinculada ao respectivo método que seria acionado e os parâmetros que seriam utilizados. Um trecho deste dicionário pode ser observado na Figura 28, que apresenta as configurações para as técnicas *Decision Tree* e Naive Bayes. Neste trecho de código, a chave `classifier` define a função que será utilizada para o aprendizado enquanto que o campo `params` especifica os parâmetros para o aprendizado. Através das configurações contidas neste dicionário o programa Python será capaz de trabalhar as oito técnicas de aprendizado utilizadas neste trabalho, que estão especificadas abaixo:

1. *Support Vector Machine* (SVC);
2. Árvore de decisão (Tree);
3. *Random Forest* (RF);
4. *Multilayer Perceptron* (MLP);
5. *Gradient Boosting Classifier*;
6. Regressão logística;
7. *K-Nearest Neighbors*;
8. Naive Bayes;

O dicionário de técnicas de aprendizado foi elaborado utilizando as quatro técnicas novas propostas e três das quatro técnicas estudadas inicialmente. A técnica de aprendizado MLP não foi utilizada neste experimento para que não fosse necessário descrever os parâmetros no dicionário neste momento. Uma vez que as configurações das técnicas de aprendizado estavam organizadas em um dicionário foi aplicada a classe `GridSearchCV` do *scikit-learn*. Essa classe permite acionar dinamicamente funções de *machine learning* como as rotinas `fit` e `predict` (PEDREGOSA et al., 2019c). Essa classe possibilita portanto que o dicionário criado seja interpretado, fazendo tanto a criação do modelo preditivo quanto o processo de predição dos dados de validação e posterior avaliação dos resultados obtidos.

A criação do modelo preditivo, que corresponde ao acionamento da função `fit`, está demonstrado na Figura 29. Neste trecho de código é feita uma repetição no dicionário `dict_classifiers` para que sejam processadas todas as técnicas selecionadas previamente. Nas linhas 2 e 3 são utilizadas as chaves `classifier` e `params` para obter o método

Figura 29: Programa Python: Acionamento da função fit a partir do dicionário

```
1       for key, classifier in dict_classifiers.items():
2           grid = GridSearchCV(classifier['classifier'],
3                               classifier['params'],
4                               refit=True, cv = 10,
5                               scoring = 'accuracy',
6                               n_jobs = -1)
7           estimator = grid.fit(X_train, Y_train)
```

Fonte: Elaborado pelo autor

principal do classificador e os parâmetros que devem ser utilizado, respectivamente. Na linha 4 é especificado o parâmetro `cv`, que se refere ao processo de *cross-validation*. Segundo Pedregosa et al. (2019a), esse processo é útil para evitar a ocorrência de *overfitting*, que é o fenômeno em que os modelos ficam especializados no *dataset* de treino porém falham ao predizer a classificação de novas instâncias. A configuração `cv=10` é referente à técnica de *Cross Validation*, que faz com que durante o treinamento dos modelos preditivos já seja executada uma pré-avaliação do desempenho do modelo em dados desconhecidos. Sendo assim, os dados de treino são separados em 10 partes, sendo utilizadas 9 para treino e 1 para avaliar se os parâmetros utilizados são aderentes para predição de novas instâncias. O parâmetro `refit=True` indica que a geração do modelo preditivo deve localizar os parâmetros mais adequados para a técnica de aprendizado e realizar novamente a função `fit` utilizando esses parâmetros. O parâmetro `scoring` define a métrica principal que será utilizada para avaliar o desempenho dos modelos gerados durante o processo de aprendizado, sendo utilizado a opção `accuracy` que se remete à métrica de precisão. Por fim, o parâmetro `n_jobs` permite configurar o número de processos paralelos que serão utilizados para a geração do modelo preditivo. Foi optado por utilizar a configuração `-1` que habilita a utilização de todos os processadores disponíveis no computador que executa o algoritmo. Após o classificador ter sido configurado com estes parâmetros, é feito o acionamento da função `fit`, que aplica essa configuração e gera um modelo preditivo correspondente.

Tabela 7: Resultados do conjunto de dados de estudo

Classificador	Precisão treino	Precisão validação
Support Vector Machine (SVC)	1.0000	1.0000
Árvore de decisão	0.9930	0.8611
Random Forest (RF)	1.0000	1.0000
Gradient Boosting Classifier	1.0000	0.9167
Regressão logística	1.0000	1.0000
K-Nearest Neighbors	1.0000	0.7777
Naive Bayes	0.9859	1.0000

Fonte: Elaborado pelo autor

Com o programa Python devidamente reestruturado foi aplicada a base de dados de estudo com informações de vinhos para avaliar os resultados gerados. A Tabela 7 apresenta a precisão gerada por esse novo algoritmo utilizando a base de dados de estudo. A primeira questão que se observa é que em geral a precisão obtida foi alta, em muitos casos chegando a precisão perfeita de 1.0000. Tem-se por exemplo as técnicas SVC, RF e Regressão Logística que obtiveram precisão máxima tanto no conjunto de dados de treino quanto no conjunto de validação. Também de forma geral a precisão no conjunto de dados de treino foi superior à precisão no conjunto de dados de validação, tendo como única exceção a técnica de Naive Bayes, que obteve precisão maior na validação. Como havia sido comentado anteriormente, esta base de dados possui dados característicos de cada classe que, embora não sejam aparentes na visão humana dos dados, tornam esta base viável para extração de regras via aprendizado de máquina. Por esse motivo pode-se afirmar que a geração de modelos preditivos utilizando o novo programa Python obteve sucesso ao gerar predições consistentes sobre essa base de dados.

Os resultados da Tabela 7 confirmam que o programa Python desenvolvido está consistente para a proposta de gerar modelos preditivos sobre uma base de dados. Com essa constatação o programa foi ajustado para que não utilizasse mais a base de dados de estudo com informações de vinhos, passando a utilizar a base de dados de alterações em sistema de gestão. O *dataset* aplicado foi o mesmo utilizado para geração dos resultados da Tabela 6, contendo 9.108 amostras de alterações para o treinamento e 937 amostras para a validação dos modelos preditivos gerados.

Tabela 8: Resultados base de dados de alterações com novo programa Python

Classificador	Precisão treino	Precisão validação
Support Vector Machine (SVC)	0.7254	0.6692
Árvore de decisão	0.7268	0.5837
Random Forest (RF)	0.9879	0.7940
Gradient Boosting Classifier	0.8175	0.6489
Regressão logística	0.7254	0.6680
K-Nearest Neighbors	0.7702	0.6947
Naive Bayes	0.6425	0.6798

Fonte: Elaborado pelo autor

A Tabela 8 apresenta os resultados obtidos com a base de dados de alterações aplicada no programa Python desenvolvido. A primeira questão que se observa é que os valores de precisão gerados são inferiores aos resultados da base de dados de vinhos, porém este é um comportamento esperado, uma vez que a base de vinhos é notoriamente viável para geração de modelos preditivos. No entanto, é possível observar que os resultados são superiores aos obtidos na Tabela 6 o que novamente comprova que o programa Python desenvolvido consegue aplicar as técnicas de aprendizado de máquina de forma mais eficaz. O modelo SVC que possui precisão de 0.54 na Tabela 6 passou a ter preci-

são de aproximadamente 0.67 na validação. O modelo de aprendizado de *Random Forest* também obteve aumento na precisão, passando de 0.74 para 0.79. No entanto, o modelo de árvore de decisão apresentou diminuição da precisão, passando de 0.65 para 0.58. O modelo de MLP não pode ser comparado pois neste experimento o dicionário de técnicas ainda não contava com as especificações desta técnica.

De forma geral houve aumento da precisão gerada para a mesma base de dados ao aplicar-se as técnicas de aprendizado de forma mais sistemática, validando a construção do programa Python através da base de estudos. Os resultados obtidos estão acima do nível de aleatoriedade, que neste caso de estudo com duas classes é de 0.50. Porém, como pode ser observado pelos resultados da base de dados de vinhos, ainda é possível obter precisões mais elevadas, contanto que os dados presentes no *dataset* tenham mais significado e proporcionem um aprendizado mais claro sobre o cenário estudado.

5.6 SEPARAÇÃO DE DADOS POR PROGRAMADOR

Na seção 5.5 foi possível observar que a aplicação sistemática das técnicas de aprendizado de máquina gerou resultados melhores para o conjunto de dados de alterações. Ainda assim, é possível buscar resultados superiores através de ajustes nas informações presentes na base de dados. Comparando-se a base de dados de alterações com a base de dados de vinhos, que obteve valores mais elevados para a precisão, pode ser observado que o *dataset* de vinhos possui uma quantidade reduzida de registros. Enquanto a base de alterações em sistema ERP contém 9.108 instâncias a base de informações de vinhos possui apenas 178 instâncias. Analisando-se esses dados levantou-se a hipótese de que a variedade maior de registros no *dataset* de alterações esteja dificultando a identificação de regras que sejam aderentes para toda a base de dados.

Para experimentar a influência da variedade de registros no *dataset* foi conduzido um experimento no qual a base de dados original com 9.108 instâncias foi diminuída. Essa base de dados contém informações sobre aproximadamente 60 programadores e como critério para redução da base de dados foi definido que seriam preservadas os dados de 3 programadores que contivessem o maior volume de registros dentre os demais. Sendo assim, a base de dados reduzida foi montada contendo 404 instâncias para o treinamento e 30 instâncias para a validação dos modelos preditivos.

Os resultados obtidos no experimento com a base de dados reduzida para apenas 3 programadores estão documentados na Tabela 9. Nesta tabela estão apresentados os valores para a precisão obtida no conjunto de dados de treino e no conjunto de dados da validação. Além dessas informações, também está visível nesta tabela a variação do valor da precisão da validação em relação aos resultados da Tabela 8, na coluna “Variação validação”. A precisão da validação continua com valores elevados, porém neste experimento a técnica de aprendizado que obteve a melhor precisão foi o classificador da técnica SVC,

Tabela 9: Resultados base de dados com 3 programadores

Classificador	Precisão treino	Precisão validação	Variação validação
Support Vector Machine (SVC)	0.7500	0.8333	+0.1641
Árvore de decisão	0.8243	0.7667	+0.1830
Random Forest (RF)	0.9876	0.7333	-0.0600
Gradient Boosting Classifier	0.9356	0.7000	+0.0511
Regressão logística	0.7649	0.8333	+0.1653
K-Nearest Neighbors	0.8771	0.6667	-0.0280
Naive Bayes	0.6856	0.6667	-0.0131

Fonte: Elaborado pelo autor

com precisão de 0.8333.

Uma questão que pode ser observa é que em geral os classificadores obtiveram uma variação positiva no resultado da precisão da validação na Tabela 9. Os classificadores SVC, Árvore de decisão, *Gradient Boosting Classifier* e Regressão Logística tiveram aumento da precisão de 0.0511 no pior caso e de 0.1830 no melhor caso. Por outro lado, os classificadores *Random Forest*, *K-Nearest Neighbors* e Naive Bayes obtiveram precisão menor, com redução variou entre -0.0131 no melhor cenário e -0.0600 no pior cenário. Ainda assim, deve ser observado que dentre os sete modelos aplicados, quatro apresentaram aumento da precisão, sendo a variação positiva da precisão muito mais expressiva do que a variação negativa. Pode ser considerado, portanto, que ao utilizar uma base de dados reduzida com informações de uma quantidade menor de programadores a geração de modelos preditivos utilizando aprendizado de máquina conseguiu identificar regras mais viáveis para a classificação correta de novas instâncias.

5.7 CONJUNTO DE MODELOS PREDITIVOS

Os resultado expostos na seção 5.6 demonstram que as técnicas de *machine learning* obtém resultados melhores ao atuarem sobre um conjunto de dados mais restrito, contendo informações de poucos programadores. Com base nessa constatação, criou-se a hipótese de que a precisão das predições deste estudo poderia ser elevada ao utilizar um *dataset* específico para cada programador existente na base de dados. Essa abordagem, no entanto, possui alguns pontos negativos que foram considerados.

A primeira questão relevante diz respeito à quantidade de amostras de alterações de cada programador, que apresenta grande variação no *dataset*. Enquanto alguns programadores possuem 200 registros de alterações outros possuem apenas 12 amostras no conjunto de dados de treino. Isso ocorre pois existem programadores recém inseridos na equipe de desenvolvimento e, sendo assim, possuem um histórico reduzido de alterações implementadas. Com poucas amostras no conjunto de dados de treinamento não é possível garantir que o cenário de estudo está expresso de forma completa no *dataset*, pois podem

faltar exemplos de situações que tendem a ocorrer pouco para membros recém inseridos na equipe. Este é o caso, por exemplo, das alterações para correções de erros, que tipicamente são executadas por integrantes mais experientes na área de negócio e, sendo assim, o aprendizado gerado para esses programadores menos experientes não contemplaria esses casos.

A rotatividade dos colaboradores é outro fator que pode ser levado em consideração para a geração do aprendizado. Se os modelos preditivos fossem construídos de forma individualizada, a cada novo colaborador inserido na equipe seria necessário fazer a geração de um modelo preditivo específico para ele. Para isso, também seria necessário aguardar que houvessem registros suficientes de alterações para utilizar no treinamento. Além disso, considerando o cenário contrário, em que um programador deixa de fazer parte da empresa, todo o conhecimento vinculado à ele nos modelos preditivos seria perdido, sem poder ser aproveitado para os demais colaboradores.

Com essas reflexões concluiu-se que utilizar um *dataset* específico para cada programador da base de dados não seria uma boa abordagem, pois geraria vários transtornos futuros e deixaria o aprendizado gerado menos genérico. Para evitar isso, foi optado por aplicar o conceito de *ensemble learning* discutido na subseção 3.2.3, que explicava como a técnica da aprendizado de *Random Forest* gerava predições baseada em um conjunto de árvores de decisão que avaliavam o problema de classificação exposto. O *ensemble learning* neste caso pode ser aplicado fazendo com que um grupo pré-determinado de modelos preditivos trabalhem em conjunto para gerar a predição para qualquer programador da base de dados, independentemente do número de alterações de exemplo que ele tenha implementado.

O primeiro passo para implementar esse conjunto de modelos preditivos foi separar a base de dados em conjuntos que seriam processados individualmente. Para que esses conjuntos contassem com uma quantidade grande de registros foram utilizados os dados dos dez programadores que geraram mais inconsistências no período extraído para o *dataset*. Esses programadores foram identificados executando uma consulta SQL no banco de dados do Sicla que totalizava a quantidade de inconsistências geradas por cada programador. Foram criados então 20 conjuntos de dados, sendo 10 destinados para treinamento e os 10 restantes para verificação dos resultados individuais, contendo dados dos mesmos programadores do treinamento. Além disso, aproveitou-se este novo experimento para inserir um atributo extra no conjunto de dados. Essa coluna contém a informação da quantidade de programas alterados durante a implementação da Ficha. Se, por exemplo, a alteração fez modificações no programa emissor de notas e no programa de manutenção do pedido do ERP, essa coluna apresentará o conteúdo “2”. Essa informação foi extraída da base de dados consultando-se a quantidade de registros na tabela de alterações que pode ser observada na Figura 4. Acredita-se que essa informação pode auxiliar na identificação de

alterações que geram inconsistência pois alterações mais abrangentes, que alteram mais pontos do sistema, estão mais suscetíveis a falhas de implementação.

O programa Python também foi adequado para que pudesse trabalhar com dados de 10 programadores simultaneamente, fazendo com que ao invés de gerar apenas um modelo preditivo fossem gerados 10 modelos preditivos, sendo cada um correspondente aos dados de um programador. Isso foi feito aproveitando principalmente o recurso de listas do Python, através do qual foi possível armazenar e tratar listas que continham os modelos preditivos gerados para cada programador, por exemplo. Ainda foi preservado o comportamento do programa de aplicar cada técnica de aprendizado de máquina, adicionando-se ainda a técnica MLP que não estava aplicada no programa até então. Sendo assim, o programa Python passou a gerar modelos preditivos utilizando as oito técnicas selecionadas neste estudo: SVC, Árvore de decisão, *Gradient Boosting Classifier*, Regressão Logística, *Random Forest*, *K-Nearest Neighbors*, Naive Bayes e MLP.

Os resultados deste programa Python que gera conjuntos de modelos preditivos foram avaliados utilizando as bases de dados específicas para validação. Isso foi feito para evitar que a validação utilizasse dados que já foram processados durante o treinamento dos modelos. Esses *datasets* foram montados a partir dos 10 programadores que geraram mais inconsistências posteriores aos 10 primeiros que foram utilizados durante o treinamento, sendo portanto do 11º ao 20º programador na lista de quantidade de erros gerados. Para cada amostra deste *dataset* de validação é gerada uma predição, sendo feita posteriormente a verificação de qual foi a classe predominante dentre as dez predições obtidas. A classe que obter o maior número de predições é selecionada como a predição final. Esse procedimento é repetido para cada uma das oito técnicas de aprendizado de máquina selecionadas neste estudo.

Na Tabela 10 estão demonstrados os resultados obtidos para a técnica de Naive Bayes. A coluna “Programador” apresenta o rótulo de identificação atribuído para cada base de dados específica com registros de apenas um programador. As linhas cujo identificador varia de 1 até 10 demonstram o resultado individual do modelo preditivo de um programador atuando sobre a base de dados de validação, que contém dados de vários programadores. A linha cuja identificação é “Conjunto” apresenta o resultado obtido quando os modelos atuaram em conjunto para gerar as predições finais. A coluna “Classe” nesta tabela identifica a classe negativa “0”, que são as alterações que não geraram inconsistência e a classe positiva “1”, que são as alterações que levaram a ocorrência de um erro. As colunas “Precisão”, “Cobertura” e “F-score” contém os valores para as métricas de validação analisadas. As duas últimas colunas contém a distribuição das predições na matriz de confusão.

Analisando-se os resultados da Tabela 10 tem-se que a precisão do conjunto atingiu o patamar de 0.80 para a classe negativa e 0.78 para a classe positiva. Entre os resultados

Tabela 10: Resultados Naive Bayes

Programador	Classe	Precisão	Cobertura	F-score	Matriz de confusão	
					Predito 0	Predito 1
1	0	0.80	0.66	0.73	33	17
	1	0.71	0.84	0.77	8	42
2	0	0.51	0.98	0.67	49	1
	1	0.67	0.04	0.08	48	2
3	0	0.84	0.42	0.56	21	29
	1	0.61	0.92	0.74	4	46
4	0	0.51	0.80	0.62	40	10
	1	0.52	0.22	0.31	39	11
5	0	0.69	0.54	0.61	27	23
	1	0.62	0.76	0.68	12	38
6	0	0.67	0.72	0.69	36	14
	1	0.70	0.64	0.67	18	32
7	0	0.76	0.68	0.72	34	16
	1	0.71	0.78	0.74	11	39
8	0	0.51	0.98	0.67	49	1
	1	0.67	0.04	0.08	48	2
9	0	0.62	0.70	0.66	35	15
	1	0.66	0.58	0.62	21	29
10	0	0.75	0.78	0.76	39	11
	1	0.77	0.74	0.76	13	37
Conjunto	0	0.80	0.78	0.79	39	11
	1	0.78	0.80	0.79	10	40

Fonte: Elaborado pelo autor

individuais houveram precisões próximas ou até superiores, como por exemplo o programador 3 que obteve o valor 0.84 para a classe negativa. No entanto, pode ser destacado que este valor alto para a precisão é acompanhado de uma cobertura de apenas 0.42, enquanto que a cobertura no identificador "Conjunto" foi de 0.78 para a classe negativa.

Existe ainda outra situação que podem ser observada ao comparar os resultados individuais com o resultado do conjunto. Tanto o programador 2 quanto o programador 8 obtiveram um índice elevado de cobertura para a classe negativa, chegando a 0.98, que significa que praticamente todas as amostras negativas da base de dados foram preditas para a classe correta. Porém, esse resultado só foi alcançado pois ambos os modelos preditivos apresentam a tendência de classificar as amostras para a classe negativa. Isso leva a cobertura desta classe à um índice elevado, em contrapartida reduz a precisão desta classe para 0.51 e ainda leva a cobertura da classe oposta positiva à um índice praticamente nulo, de 0.04. Essa característica dos modelos preditivos dos programadores 2 e 8 pode ser observada também na matriz de confusão, que concentra quase a totalidade

das predições na classe “0”.

Por essas situações é possível afirmar que o desempenho da predição do conjunto se mostrou mais equilibrado que o desempenho individual de cada modelo. Isso se reflete em índices de 0.80 e 0.78 para a precisão do conjunto, sem o comprometimento da cobertura, que ficou em 0.78 e 0.80 para as classes negativa e positiva respectivamente. A coluna F-score expressa esse equilíbrio entre precisão e cobertura, uma vez que apresenta o valor mais elevado entre os resultados obtidos para a técnica de aprendizado Naive Bayes.

Para simplificação da discussão e análise dos resultados os mesmos serão expostos consolidando a linha com o identificador “Conjunto” das demais técnicas de aprendizado, sem detalhar os resultados individuais dos modelos específicos de cada programador. Estes resultados detalhados podem ser consultados no Apêndice A, que contém as tabelas completas de resultados para cada uma das seguintes técnicas de aprendizado: Regressão logística, *K-Nearest Neighbors*, SVM, *Gradient Boosting*, Árvore de decisão, *Random Forest* e MLP.

Tabela 11: Resultados consolidados

Técnica	Classe	Precisão	Cobertura	F-score	Matriz de confusão	
					Predito 0	Predito 1
Naive Bayes	0	0.80	0.78	**0.79**	39	11
	1	0.78	0.80	**0.79**	10	40
Regressão logística	0	0.82	0.62	0.70	31	19
	1	0.69	0.86	0.77	7	43
K-Nearest Neighbors	0	0.76	0.62	0.68	31	19
	1	0.68	0.80	0.73	10	40
SVM	0	0.81	0.70	0.75	35	15
	1	0.74	0.84	**0.79**	8	42
Gradient Boosting	0	0.77	0.60	0.67	30	20
	1	0.67	0.82	0.74	9	41
Árvore de decisão	0	0.71	**0.54**	**0.61**	27	23
	1	**0.63**	0.78	0.70	11	39
Random Forest	0	0.76	0.68	0.72	34	16
	1	0.71	0.78	0.74	11	39
MLP	0	**0.88**	0.60	0.71	30	20
	1	0.70	**0.92**	**0.79**	4	46

Fonte: Elaborado pelo autor

A Tabela 11 apresenta os resultados consolidados das oito técnicas de aprendizado utilizando o programa Python que gera conjuntos de modelos preditivos. Os resultados máximos e mínimos de cada métrica estão sinalizados em negrito para as métricas de precisão, cobertura e F-score. Analisando a coluna F-score, que resume o desempenho das predições, nota-se que a técnica Naive Bayes teve a pontuação mais elevada, de 0.79 para ambas as classes, enquanto que o modelo com Árvore de decisão obteve o pior

desempenho, de 0.70 para a classe positiva e 0.61 para a classe negativa. Ainda avaliando a coluna F-score é possível observar que a técnica de aprendizado de conjunto realmente leva à geração de resultados mais consistentes, pois a técnica de *Random Forest*, que aplica este conceito, obteve um desempenho melhor que a técnica mais simples, de Árvore de decisão.

A separação da métrica F-score entre a classe positiva e negativa, que neste estudo representam as alterações que geraram inconsistência e as que não geraram, evidencia que em geral os modelos não obtém o mesmo desempenho para a predição de cada classe. Com exceção da técnica Naive Bayes, que obteve índice igual para as duas classes na métrica F-score, todas as demais técnicas de aprendizado de máquina tiveram um desempenho superior nas predições para a classe positiva "1". Essa diferença entre o desempenho das duas classes não se justifica pela distribuição das amostras na base de dados, uma vez que o treinamento dos modelos ocorreu com uma base de dados que possui quantidades equilibradas de amostras das duas classes.

A diferença de comportamento entre as predições das duas classes é mais evidente nas métricas de precisão e de cobertura. Em todos os resultados apresentados na Tabela 11 a precisão da classe negativa foi maior que a precisão da classe positiva. O resultado nessa métrica variou de 71% para a técnica de Árvore de decisão até 88% para a técnica MLP, considerando a classe negativa. Isso significa que, no caso da técnica MLP, 88% das vezes em que uma instância foi rotulada como "alteração que gera inconsistência" esta classificação estava correta. Essa diferença de desempenho pode ser observada na matriz de confusão avaliando a quantidade de instâncias classificadas com o rótulo incorreto pois, de forma geral, essa quantidade é maior nas predições da classe positiva. Na técnica KNN tem-se por exemplo apenas 10 instâncias preditas incorretamente para a classe "0", enquanto que existem 19 instâncias rotuladas de forma incorreta para a classe positiva, o que torna a precisão atingida nesta classe menor.

Em contrapartida, a classe negativa também apresentou menos cobertura que a classe positiva, o que significa que os modelos preditivos possuem uma abrangência menor ao identificar as alterações que não geram inconsistência. Isso é um reflexo da predições feitas incorretamente para a classe positiva, pois uma vez que o modelo preditivo opta mais vezes por rotular utilizando a classe positiva ele também deixa de apontar a classe negativa em mais casos. A diferença da precisão entre a classe positiva e a classe negativa não se apresentou de forma tão expressiva quanto a variação da cobertura. A variação da precisão entre as duas classes foi de 18% no modelo MLP até 2% no melhor caso, na técnica Naive Bayes. Já a variação da cobertura entre as duas classes foi maior, de 2% no melhor caso, do modelo Naive Bayes, até 32% no pior cenário, da classe MLP.

Por esses resultados da variação de desempenho entre as duas classes para as métricas de precisão e cobertura pode-se afirmar que o modelo preditivo utilizando a técnica

Naive Bayes obteve um funcionamento mais equilibrado neste cenário de estudo. A coluna F-score da Tabela 11 resume esse resultado, pois os índices mais elevados desta métrica são encontrados na técnica Naive Bayes. Isso significa que a abordagem probabilística da técnica Naive Bayes, explicada na subseção 3.2.8, foi mais aderente para solucionar o problema de classificação deste estudo.

Entretanto, o índice 0.79 para a métrica F-score, apesar de ser um valor elevado, não é um resultado excelente. Na seção 5.5 havia sido conduzido um experimento no qual foi utilizada uma base de dados com informações de vinhos aplicando-se as mesmas oito técnicas de aprendizado utilizadas na base de dados de alterações. Mesmo antes de criar qualquer processamento extra usando aprendizado de conjunto a Tabela 7 já apresentava índices altíssimos para a precisão, em todos os modelos. Isso deixa evidente que o resultado da métrica F-score limitado ao valor de 0.79 não é consequência da aplicação incorreta das técnicas de aprendizado ou da eventual incapacidade das mesmas de gerar predições apuradas no processo seguido neste trabalho, utilizando a linguagem de programação Python. O desafio na geração de predições neste cenário de estudo está justamente nas informações contidas na base de dados fornecida para os processamentos de aprendizado de máquina.

O que se observa através dos resultados obtidos é que as informações contidas na base de dados não são totalmente capazes de trazer significado para as instâncias, de forma que as técnicas de *machine learning* consigam aprender o que diferencia uma alteração que gera inconsistência de uma alteração que não gera inconsistência. Para que ocorra o aprendizado é necessário que os dados fornecidos contenham informações relevantes para a classificação, o que não está ocorrendo plenamente conforme os resultados obtidos demonstram. Uma possibilidade é que os atributos selecionados ainda não estejam expressando o cenário de estudo de forma que as técnicas de aprendizado de máquina consigam extrair regras de classificação totalmente eficientes. A inserção de mais atributos no conjunto de dados pode auxiliar nesta questão, de forma que os algoritmos de aprendizado tenha mais dados para trabalhar e portanto mais possibilidades para identificação de padrões. Outra questão que pode estar influenciando negativamente os resultados obtidos é o tamanho da base de dados de cada programador. As bases possuem cerca de 100 a 200 registros de alterações, por um lado essa redução auxilia o aprendizado por restringir a quantidade de pessoas envolvidas nas alterações, mas ao mesmo tempo diminui a variedade de situações expostas durante o treinamento, fazendo com que as regras de classificação geradas estejam mais suscetíveis à falhas.

6 CONCLUSÃO

Através do levantamento bibliográfico realizado neste estudo foi possível constatar que as aplicações dos conceitos de aprendizado de máquina estão em expansão, atingindo vários cenários distintos. Este trabalho propunha a aplicação de técnicas de aprendizado de máquina para a criação de modelos preditivos capazes de identificar se determinada alteração no sistema ERP iria gerar alguma inconsistência futuramente. O referencial teórico pesquisado apontou que eram necessárias várias etapas para alcançar esse objetivo.

Primeiramente foram obtidos os dados que seriam alvo do aprendizado. Esses dados foram extraídos via consultas SQL no banco de dados do Sicla, gerando um arquivo CSV com todas as informações relevantes para o aprendizado. Estes dados foram pré-processados de forma a facilitar a indução de modelos preditivos. Após isso foram aplicados diversos algoritmos de aprendizado de máquina, avaliando os resultados obtidos por cada um deles a partir das métricas estudadas. Além disso, os experimentos realizados indicaram que a base de dados é desbalanceada, o que dificulta a geração de modelos preditivos. Foi necessário aplicar pré-processamentos nesta base para obter resultados mais consistentes. Também foi observado que, ao utilizar um conjunto de dados específico para a validação, também houve melhora nos resultados para a predição da classes positiva e negativa.

O objetivo geral de geração de modelos preditivos com base nos dados de alterações em sistema ERP foi atingido. As métricas analisadas indicaram que os modelos gerados foram capazes de gerar predições corretas, embora o desempenho obtido por estes modelos não tenha alcançado índices elevados. Através de uma base de estudo auxiliar com dados distintos foi possível constatar que as técnicas de aprendizado preditivo utilizadas são capazes de gerar classificações próximas à perfeição. Isso indica que os dados de alterações em sistema ERP selecionados não são suficientemente expressivos para que as técnicas de aprendizado preditivo gerem regras de classificação consistentes.

Os resultados obtidos neste estudo indicam que existe um aumento considerável do desempenho dos modelos preditivos quando estes atuam em forma de conjunto para gerar predições no cenário de alterações em sistema de gestão. Além de gerar resultados mais consistentes, a aplicação da técnica de *ensemble learning* também trouxe um equilíbrio entre os resultados das métricas de precisão e cobertura, sem que uma métrica fosse prejudicada ao obter índices maiores para a outra.

Após a aplicação do aprendizado de conjunto foi possível observar que a técnica de aprendizado que obteve o pior desempenho foi a Árvore de decisão, com a métrica F-score resultando em 0.61 para a classe negativa e 0.70 para a classe positiva. Esse

resultado justifica-se pela simplicidade da técnica frente à complexidade apresentada pelo cenário de estudo. Por outro lado, a técnica com melhor desempenho foi Naive Bayes. Essa técnica apresentou o índice 0.79 na métrica F-score na predição de ambas as classes positiva e negativa. Isso indica que a abordagem probabilística utilizada por essa técnica de classificação foi mais aderente aos dados, possibilitando predições mais corretas.

No decorrer do trabalho também foi possível observar que, como já indicava o referencial teórico, a linguagem de programação Python se apresenta como uma excelente opção para desenvolvimento voltado para a Ciência de Dados. Com pouco conhecimento prévio sobre esta linguagem foi possível implementar diversas tarefas complexas de geração de modelos preditivos, mantendo ainda a simplicidade e clareza do código-fonte.

6.1 TRABALHOS FUTUROS

Os modelos preditivos gerados neste estudo foram capazes de efetuar predições com desempenho bom, acima da aleatoriedade. No entanto, existem outras abordagens que podem ser conduzidas visando obter resultados mais próximos à classificação perfeita de todas as amostras de alterações no sistema ERP. Sugere-se como trabalho futuro a busca por dados da alteração que possam diferenciar melhor uma alteração que gera inconsistência de uma alteração que não gera nenhum erro. Pode-se, por exemplo, utilizar dados mais específicos do código-fonte alterado, como a quantidade de linhas modificadas ou ainda indicadores para a modificação de código já existente ou apenas criação de código-fonte novo.

Além disso, ainda pode ser estudada a integração dos modelos preditivos gerados com a ferramenta Sicla, que controla os registros de alteração no sistema. Dessa forma, uma modificação rotulada para a classe das alterações que geram inconsistência poderia gerar alertas internos para a empresa. Os dados utilizados para realizar uma predição neste estudo são todos dados anteriores à disponibilização da alteração para o cliente final. Isso possibilita, portanto, que essa avaliação seja feita em ambiente interno, sem que o cliente receba a alteração. Isso tornaria o resultado entregue para o cliente um produto de maior qualidade.

REFERÊNCIAS

AL-GHAMDI, A. S. Using logistic regression to estimate the influence of accident factors on accident severity. *Accident Analysis and Prevention*, v. 34, p. 729–741, 2002. Citado 2 vezes nas páginas 32 e 33.

ALPPAYDIN, E. *Introduction to Machine Learning*. 2. ed. Cambridge: The MIT Press, 2010. ISBN 978-0-262-01243-0. Citado 5 vezes nas páginas 21, 23, 24, 26 e 31.

BEAL, A. *Gestão estratégica da informação: Como transformar a Informação e a Técnologia da informação em fatores de Crescimento e de alto desempenho nas organizações*. São Paulo, SP: Atlas, 2004. Citado 2 vezes nas páginas 14 e 15.

CRUZ, J. I. B.; RUIZ, D. Uma experiência em mineração de processos de manutenção de software. *In Companion Proceedings of the XIV Brazilian Symposium on Multimedia and the Web (WebMedia '08)*, Vila Velha, ES, p. 247–253, 2008. Citado na página 12.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control Signals and Systems*, v. 2, p. 303–314, 1989. Citado na página 30.

DAVENPORT, T. H. *Missão Crítica: obtendo vantagens competitivas com os sistemas de gestão empresarial*. Porto Alegre, RS: Bookman, 2002. Citado na página 11.

DIETTERICH, T. G. *Ensemble learning*. 2. ed. Cambridge, Massachusetts: The MIT Press, 2002. 405—408 p. Citado na página 28.

DUA, D.; GRAFF, C. *UCI Machine Learning Repository*. 2017. Disponível em: <http://archive.ics.uci.edu/ml>. Acesso em: 29 jun. 2019. Citado na página 61.

DUA, D.; GRAFF, C. *Wine Data Set*. 2017. Disponível em: <https://archive-.ics.uci.edu/ml/datasets/Wine>. Acesso em: 29 jun. 2019. Citado na página 61.

FACELI, K. et al. *Inteligência Artificial: Uma abordagem de Aprendizado de Máquina*. 1. ed. Rio de Janeiro: LTC, 2011. ISBN 978-85-216-1880-5. Citado 6 vezes nas páginas 20, 21, 22, 26, 29 e 30.

FAYYAD, U. M.; PIATETSKY-SHAPIRO, G.; SMYTH, P. *From Data Mining to Knowledge Discovery: An Overview. Knowledge Discovery and Data Mining*. Menlo Park: AAAI Press, 1996. Citado na página 11.

GOLDSCHMIDT, R.; PASSOS, E. *Data Mining: Um Guia Prático*. 4. ed. Rio de Janeiro: Elsevier Editora Ltda, 2005. Citado 3 vezes nas páginas 11, 12 e 26.

HABERKORN, E. *Teoria do ERP*. São Paulo, SP: Makron Books, 1999. Citado na página 11.

HARRINGTON, P. *Machine learning in action*. Shelter Island, NY: Manning, 2012. ISBN 978-1-617-29018-3. Citado 3 vezes nas páginas 34, 35 e 36.

LEON, A. *Enterprise Resource Planning*. 3. ed. New Delhi, India: McGraw Hill Education, 2014. Citado na página 16.

LUCAS, L. d. C. d. S. Árvores, florestas e sua função como preditores: Uma aplicação na avaliação do grau de maturidade de empresas. *Revista Brasileira de Pesquisas de Marketing, Opinião e Mídia*, v. 6, p. 6–11, 2011. Citado na página 26.

MITCHELL, T. *The Discipline of Machine Learning*. 2006. Disponível em: <http://www.cs.cmu.edu/~tom/pubs/MachineLearning.pdf>. Acesso em: 17 mar. 2019. Citado na página 12.

NATEKIN, A.; KNOLL, A. Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, p. 7–21, 10 2013. Citado na página 32.

Oracle Corporation. *Oracle SQL Developer: overview*. 2019. Disponível em: <https://www.oracle.com/technetwork/pt/developer-tools/sql-developer/overview/index.html>. Acesso em: 07 mai. 2019. Citado na página 41.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011. Citado na página 38.

PEDREGOSA, F. et al. *Cross-validation: Evaluating estimator performance*. 2019. Disponível em: <https://scikit-learn.org/stable/modules/cross_validation.html>. Acesso em: 30 jun. 2019. Citado na página 63.

PEDREGOSA, F. et al. *Gradient Boosting Classifier*. 2019. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble-.GradientBoostingClassifier.html>. Acesso em: 28 ago. 2019. Citado na página 32.

PEDREGOSA, F. et al. *GridSearchCV*. 2019. Disponível em: <https://scikit-learn.org-/stable/modules/generated/sklearn.model_selection.GridSearchCV.html>. Acesso em: 30 jun. 2019. Citado na página 62.

Project Jupyter. *The Jupyter Notebook*. 2019. Disponível em: <https://jupyter.org/>. Acesso em: 09 mai. 2019. Citado na página 43.

PytData. *Pandas Documentation: Package overview*. 2019. Disponível em: <http://pandas.pydata.org/pandas-docs/stable/getting_started/overview.html>. Acesso em: 02 mai. 2019. Citado na página 38.

Python Software Foundation. *BeginnersGuide/Overview*. 2017. Disponível em: <https://wiki.python.org/moin/BeginnersGuide/Overview>. Acesso em: 02 mai. 2019. Citado na página 37.

RASCHKA, S. *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and Tensor Flow*. 2. ed. Birmingham, United Kingdom: Packt Publishing, 2017. 13 p. Citado na página 37.

Rech Informática. *Nossos clientes*. 2019. Disponível em: <https://www.rech.com.br-/clientes>. Acesso em: 17 mar. 2019. Citado na página 11.

Rech Informática. *Quem somos*. 2019. Disponível em: <https://www.rech.com.br-/sobre>. Acesso em: 11 mai. 2019. Citado na página 17.

SAITO, T.; REHMSMEIER, M. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, Public Library of Science, v. 10, n. 3, p. 1–21, 03 2015. Disponível em: <https://doi.org/10.1371/journal.pone.0118432>. Citado na página 39.

TURBAN, E.; JR, R. K. R.; POTTER, R. E. *Introdução a Sistemas de Informação: uma Abordagem Gerencial*. Rio de Janeiro, RJ: Campus, 2003. Citado na página 15.

Visual Studio Code. *Basic Editing*. 2019. Disponível em: <https://code.visualstudio-.com/docs/editor/codebasics>. Acesso em: 13 set. 2019. Citado na página 50.

Apêndices

A RESULTADOS DE MODELOS ATUANDO EM CONJUNTO

Tabela 12: Resultados Regressão logística

Programador	Classe	Precisão	Cobertura	F-score	Matriz de confusão	
					Predito 0	Predito 1
1	0	0.80	0.74	0.77	37	13
	1	0.76	0.82	0.79	9	41
2	0	0.79	0.30	0.43	15	35
	1	0.57	0.92	0.70	4	46
3	0	0.92	0.68	0.78	34	16
	1	0.75	0.94	0.83	3	47
4	0	0.66	0.82	0.73	41	9
	1	0.76	0.58	0.66	21	29
5	0	0.81	0.50	0.62	25	25
	1	0.64	0.88	0.74	6	44
6	0	0.76	0.62	0.68	31	19
	1	0.68	0.80	0.73	10	40
7	0	0.82	0.62	0.70	31	19
	1	0.69	0.86	0.77	7	43
8	0	0.80	0.56	0.66	28	22
	1	0.66	0.86	0.75	7	43
9	0	0.73	0.70	0.71	35	15
	1	0.71	0.74	0.73	13	37
10	0	0.73	0.80	0.76	40	10
	1	0.78	0.70	0.74	15	35
Conjunto	0	0.82	0.62	0.70	31	19
	1	0.69	0.86	0.77	7	43

Tabela 13: Resultados *K-Nearest Neighbors*

Programador	Classe	Precisão	Cobertura	F-score	Matriz de confusão	
					Predito 0	Predito 1
1	0	0.67	0.68	0.67	34	16
	1	0.67	0.66	0.67	17	33
2	0	0.62	0.68	0.65	34	16
	1	0.64	0.58	0.61	21	29
3	0	0.76	0.82	0.79	41	9
	1	0.80	0.74	0.77	13	37
4	0	0.65	0.64	0.65	32	18
	1	0.65	0.66	0.65	17	33
5	0	0.65	0.48	0.55	24	26
	1	0.59	0.74	0.65	13	37
6	0	0.67	0.60	0.63	30	20
	1	0.64	0.70	0.67	15	35
7	0	0.65	0.72	0.69	33	17
	1	0.69	0.62	0.65	19	31
8	0	0.63	0.66	0.65	36	14
	1	0.65	0.62	0.63	19	31
9	0	0.62	0.52	0.57	26	24
	1	0.59	0.68	0.63	16	34
10	0	0.71	0.74	0.73	37	13
	1	0.73	0.70	0.71	15	35
Conjunto	0	0.76	0.62	0.68	31	19
	1	0.68	0.80	0.73	10	40

Tabela 14: Resultados SVM

Programador	Classe	Precisão	Cobertura	F-score	Matriz de confusão	
					Predito 0	Predito 1
1	0	0.87	0.68	0.76	34	16
	1	0.74	0.90	0.81	5	45
2	0	0.67	0.76	0.71	38	12
	1	0.72	0.62	0.67	19	31
3	0	0.91	0.78	0.84	39	11
	1	0.81	0.92	0.86	4	46
4	0	0.68	0.78	0.73	39	11
	1	0.74	0.64	0.69	18	32
5	0	0.85	0.56	0.67	28	22
	1	0.67	0.90	0.77	4	46
6	0	0.74	0.56	0.64	28	22
	1	0.65	0.80	0.71	10	40
7	0	0.84	0.74	0.79	37	13
	1	0.77	0.86	0.81	7	43
8	0	0.65	0.60	0.63	30	20
	1	0.63	0.68	0.65	16	34
9	0	0.74	0.68	0.71	34	16
	1	0.70	0.76	0.73	12	38
10	0	0.76	0.82	0.79	41	9
	1	0.80	0.74	0.77	13	37
Conjunto	0	0.81	0.70	0.75	35	15
	1	0.74	0.84	0.79	8	42

Tabela 15: Resultados *Gradient Boosting*

Programador	Classe	Precisão	Cobertura	F-score	Matriz de confusão	
					Predito 0	Predito 1
1	0	0.76	0.70	0.73	35	15
	1	0.72	0.78	0.75	11	39
2	0	0.61	0.70	0.65	35	15
	1	0.65	0.56	0.60	22	28
3	0	0.72	0.56	0.63	28	22
	1	0.64	0.78	0.70	11	39
4	0	0.66	0.54	0.59	27	23
	1	0.61	0.72	0.66	14	36
5	0	0.76	0.58	0.66	29	21
	1	0.66	0.82	0.73	9	41
6	0	0.62	0.52	0.57	26	24
	1	0.59	0.68	0.63	16	34
7	0	0.51	0.54	0.52	27	23
	1	0.51	0.48	0.49	26	24
8	0	0.55	0.58	0.57	29	21
	1	0.55	0.52	0.54	24	26
9	0	0.66	0.74	0.70	37	13
	1	0.70	0.62	0.66	19	31
10	0	0.58	0.80	0.67	40	10
	1	0.68	0.42	0.52	29	21
Conjunto	0	0.77	0.60	0.67	30	20
	1	0.67	0.82	0.74	9	41

Tabela 16: Resultados Árvore de decisão

Programador	Classe	Precisão	Cobertura	F-score	Matriz de confusão	
					Predito 0	Predito 1
1	0	0.71	0.60	0.65	30	20
	1	0.66	0.76	0.70	12	38
2	0	0.45	0.58	0.50	29	21
	1	0.40	0.28	0.33	36	14
3	0	0.65	0.52	0.58	26	24
	1	0.60	0.72	0.65	14	36
4	0	0.62	0.42	0.50	21	29
	1	0.56	0.74	0.64	13	37
5	0	0.72	0.72	0.72	36	14
	1	0.72	0.72	0.72	14	36
6	0	0.62	0.56	0.59	28	22
	1	0.60	0.66	0.63	17	33
7	0	0.46	0.52	0.49	26	24
	1	0.45	0.40	0.43	30	20
8	0	0.50	0.66	0.57	33	17
	1	0.50	0.34	0.40	33	17
9	0	0.56	0.58	0.57	29	21
	1	0.56	0.54	0.55	23	27
10	0	0.69	0.58	0.63	29	21
	1	0.64	0.74	0.69	13	37
Conjunto	0	0.71	0.54	0.61	27	23
	1	0.63	0.78	0.70	11	39

Tabela 17: Resultados Random Forest

Programador	Classe	Precisão	Cobertura	F-score	Matriz de confusão	
					Predito 0	Predito 1
1	0	0.71	0.72	0.71	36	14
	1	0.71	0.70	0.71	15	35
2	0	0.61	0.62	0.61	31	19
	1	0.61	0.60	0.61	20	30
3	0	0.70	0.62	0.66	31	19
	1	0.66	0.74	0.70	13	37
4	0	0.62	0.80	0.70	40	10
	1	0.71	0.50	0.59	25	25
5	0	0.71	0.60	0.65	30	20
	1	0.66	0.76	0.70	12	38
6	0	0.79	0.62	0.70	31	19
	1	0.69	0.84	0.76	8	42
7	0	0.63	0.64	0.63	32	18
	1	0.63	0.62	0.63	19	31
8	0	0.47	0.50	0.49	25	25
	1	0.47	0.44	0.45	28	22
9	0	0.69	0.86	0.77	43	7
	1	0.82	0.62	0.70	19	31
10	0	0.67	0.84	0.74	42	8
	1	0.78	0.58	0.67	21	29
Conjunto	0	0.76	0.68	0.72	34	16
	1	0.71	0.78	0.74	11	39

Tabela 18: Resultados MLP

Programador	Classe	Precisão	Cobertura	F-score	Matriz de confusão	
					Predito 0	Predito 1
1	0	0.87	0.66	0.75	33	17
	1	0.73	0.90	0.80	5	45
2	0	0.77	0.54	0.64	27	23
	1	0.65	0.84	0.73	8	42
3	0	0.91	0.64	0.75	32	18
	1	0.72	0.94	0.82	3	47
4	0	0.72	0.66	0.69	33	17
	1	0.69	0.74	0.71	13	37
5	0	0.92	0.44	0.59	22	28
	1	0.63	0.96	0.76	2	48
6	0	0.83	0.60	0.70	30	20
	1	0.69	0.88	0.77	6	44
7	0	0.91	0.64	0.75	32	18
	1	0.72	0.94	0.82	3	47
8	0	0.74	0.56	0.64	28	22
	1	0.65	0.80	0.71	10	40
9	0	0.67	0.66	0.67	33	17
	1	0.67	0.68	0.67	16	34
10	0	0.85	0.80	0.82	40	10
	1	0.81	0.86	0.83	7	43
Conjunto	0	0.88	0.60	0.71	30	20
	1	0.70	0.92	0.79	4	46