NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

Desenvolvimento de um Sistema de Previsão de Insolvência das Sociedades Seguradoras no Brasil

Bruno de Lima Vieira

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Métodos Analíticos Avançados, Especialização em Data Science

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

DESENVOLVIMENTO DE UM SISTEMA DE PREVISÃO DE INSOLVÊNCIA DAS SOCIEDADES SEGURADORAS NO BRASIL

por

Bruno de Lima Vieira

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Métodos Analíticos Avançados, Especialização em Data Science

Orientador: Prof. Doutor Mauro Castelli

Novembro 2022

DEDICATÓRIA

À minha esposa e filha, por embarcarem nessa jornada e atravessarem o oceano para a realização deste sonho. Sem vocês, nada disso seria possível.

AGRADECIMENTOS

Agradeço a minha esposa e filhas, Cíntia, Manuela e Lara, razões da minha vida e parceiras pela eternidade.

Agradeço a minha querida mãe, pela dedicação, pelo exemplo e, principalmente, pelo amor com que me criou.

Agradeço a minha família, pelo suporte e colaboração durante estes 2 anos pandêmicos.

Agradeço a todos os amigos, que compreenderam a distância, sem nunca deixar de apoiar.

Agradeço ao meu orientador, Mauro Castelli, por estar disponível sempre que necessário, pelas rápidas respostas e por acreditar na minha proposta desde o princípio. Além disso, na condição de professor de duas disciplinas durante o mestrado, por todo o conhecimento ofertado.

Agradeço à Superintendência de Seguros Privados (Susep), cujo programa de capacitação me permitiu estar afastado por 2 anos.

Agradeço ao Maurício e ao Rodrigo, da Susep, pelas contribuições na formulação da proposta.

Agradeço especialmente ao chefe, afilhado de casamento e irmão Thiago Barata, cuja imensa compreensão me permitiu finalizar esta tese.

RESUMO

O mercado segurador vem adquirindo participação cada vez mais relevante no cenário econômico do país, com a participação no PIB crescendo significativamente nos últimos anos, e a Superintendência de Seguros Privados (Susep), órgão responsável pelo controle e fiscalização de todo o mercado, tem por objetivo zelar pela solvência das companhias seguradoras e garantir o interesse dos segurados. Neste contexto, prever com antecedência a ocorrência de problemas financeiros é fundamental para evitar a quebra de uma companhia e permitir que os consumidores de seguro tenham preservado seu direito a receber as indenizações ou a poupança acumulada por anos. O presente trabalho propõe a utilização de modelos preditivos, mais especificamente a classe de algoritmos baseados em Machine Learning (ML), para sinalização antecipada de situações de insuficiência de capital em sociedades seguradoras e resseguradoras. O caso foi transformado em um problema de classificação binária, cujas variáveis explicativas foram indicadores financeiros e macroeconômicos e outros indicadores que refletem o porte da empresa, pertencimento a conglomerados financeiros, atuação em determinados ramos de seguro e problemas relacionados a controles internos. Na modelagem, foram utilizados diversos algoritmos de aprendizagem supervisionada, desde mais simples, como Naive Bayes, a mais complexos, como Gradient Boosting. Os classificadores foram treinados e avaliados, sendo conduzida uma comparação das performances em diferentes abordagens, para a Recall, Precision e F1-Measure. O modelo de melhor performance foi capaz de atingir uma Recall de 92%, conseguindo prever 11 dos 12 casos de insuficiência no test set.

PALAVRAS-CHAVE

Ciência de Dados; Inteligência Artificial; Aprendizagem de Máquina; Aprendizagem Supervisionada; Classificação Binária; Sistema de Sinalização Antecipada; Classificação Desequilibrada; Seguros; Insolvência

ABSTRACT

The Insurance market in Brazil has been taking an even more relevant in the economic outlook, with its GDP participation growing significantly in the last years, and Susep, the entity responsible for monitoring and overseeing the whole market, has the objective of protecting the solvency of insurance companies e ensure the policyholders' concerns. In this context, it is crucial to predict in advance the occurrence of financial difficulties in order to avoid the bankruptcy of a company and to permit the consumers to receive their claims or their savings accumulated over the years. This thesis aims to study the use of predictive models, more specifically the class of algorithms based on Machine Learning (ML), to create an early warning system for situations of violation of capital requirements in insurance and reinsurance companies. The case was designed to be a binary classification problem, whose independent variables were financial and macroeconomic indicators as well as further indicators that reflect the size of the company, participation in financial conglomerates, insurance lines of action, and internal control issues. In the modeling, a range of supervised learning algorithms was used, from the simplistic ones, like Naive Bayes, to the more complex ones, like Gradient Boosting. The classifiers were implemented and evaluated, by conducting a performance comparison for different approaches, using Recall, Precision, and F1-Measure metrics. The best model was able to reach a Recall of 92%, managing to predict 11 out of 12 instances of the positive class on the test set.

KEYWORDS

Data Science; Artificial Intelligence; Machine Learning; Supervised Learning; Binary Classification; Early Warning System; Imbalanced Classification; Insurance; Insolvency

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

BCB	Banco Central do Brasil
CGMOP	Coordenação Geral de Monitoramento Prudencial
CMR	Capital Mínimo Requerido
CRISP-DM	Cross-Industry Standard Process for Data Mining
FIPSUSEP	Formulário de Informações Periódicas da SUSEP
FN	False Negatives
FP	False Positives
IAIS	International Association of Insurance Supervisors
IBGE	Instituto Brasileiro de Geografia e Estatística
ICA	Índice Combinado Ampliado
ILC	Índice de Liquidez Corrente
IPCA	Índice Nacional de Preços ao Consumidor Amplo
ISIN	Índice de Sinistralidade
KDD	Knowledge Discovery in Database
KNN	K-Nearest Neighbors
ML	Machine Learning
PCA	Principal Component Analysis
PIB	Produto Interno Bruto
PLA	Patrimônio Líquido Ajustado
PRS	Plano de Regularização de Solvência
RFE	Recursive Feature Elimination
ROE	Return on Equity
SAPIEMS	Sistema de Acompanhamento e Processamento de Informações e Estatísticas do Mercado Segurador
SEMMA	Sample, Explore, Modify, Model, Assess
SMOTE	Synthetic Minority Oversampling TEchnique

SSMS SQL Server Management Studio

SUSEP Superintendência de Seguros Privados

SVM Support Vector Machines

TN True Negatives

TP True Positives

TPOT Tree-based Pipeline Optimization Tool

1. INTRODUÇÃO

1.1. O ECOSSISTEMA DE SEGUROS NO BRASIL EM NÚMEROS

Os diplomas legais que regulam o mercado de seguros no Brasil dispõem essencialmente acerca de 4 grandes segmentos: as atividades de seguros, previdência complementar aberta, capitalização e resseguro.

Em linhas gerais, o mercado segurador vem adquirindo participação cada vez mais relevante no cenário econômico do país, tendo a participação no PIB brasileiro subido de 2,6% em 2003 para 3,8% em 2019, cenário este que merece especial destaque, tendo em vista que o PIB também obteve uma tendência crescente durante o período de análise (SUSEP, 2020).

De acordo com o 8º Relatório de Análise e Acompanhamento dos Mercados Supervisionados (SUSEP, 2020), elaborado pela Superintendência de Seguros Privados (Susep), há potencial para atingir valores da ordem de 6% a 10% - valores observados em países com mercado segurador maduro.

No que se refere ao faturamento alcançado pelo setor, o volume de receitas arrecadadas em 2019 é da ordem de R$ 272,6 bilhões, o que representa um aumento de 6,9% relativamente ao ano anterior, já descontada a inflação acumulada no período (SUSEP, 2020).

Sob o prisma das provisões técnicas[1], o mercado abrange sob sua guarda um volume de R$ 1,1 trilhão, o que reforça sua representatividade e vem proporcionando expressiva contribuição para a construção da poupança nacional e para o desenvolvimento do país (SUSEP, 2020).

1.2. A SUSEP E A FUNÇÃO DE MONITORAMENTO DO MERCADO SEGURADOR BRASILEIRO

A Superintendência de Seguros Privados (Susep) é uma autarquia integrante da estrutura do Ministério da Economia e tem por objetivo o controle e fiscalização de todas as operações componentes do ecossistema de seguros acima mencionado, tendo sido criada pelo Decreto-Lei Nº 73/66, que dispõe sobre o Sistema Nacional de Seguros Privados.

O referido decreto delineia as políticas que devem ser perseguidas por todos os agentes integrantes do sistema nacional de seguros, dentre as quais se destaca a necessidade de se preservar a liquidez e a solvência das sociedades seguradoras. Tal objetivo se coaduna com um dos primeiros dispositivos da norma, que deixa cristalino que toda e qualquer diretriz por ele estabelecida tem, em última instância, por finalidade garantir o interesse dos segurados e beneficiários dos contratos de seguro.

Com estes conceitos em mente (preservação do interesse dos segurados e da solvência das empresas), a Susep possui em sua estrutura 4 diretorias técnicas, às quais compete a execução das diversas atribuições regimentais, conforme podemos ver na figura 1.

Dentro da estrutura da Diretoria Técnica 4, encontra-se a Coordenação Geral de Monitoramento Prudencial (CGMOP) que tem como uma de suas competências regimentais o monitoramento das

[1] Provisões técnicas são montantes que as seguradoras devem constituir e manter, a qualquer momento, para fazer face às obrigações decorrentes dos contratos de seguro.

operações e do funcionamento das sociedades e entidades supervisionadas sob o ponto de vista prudencial.

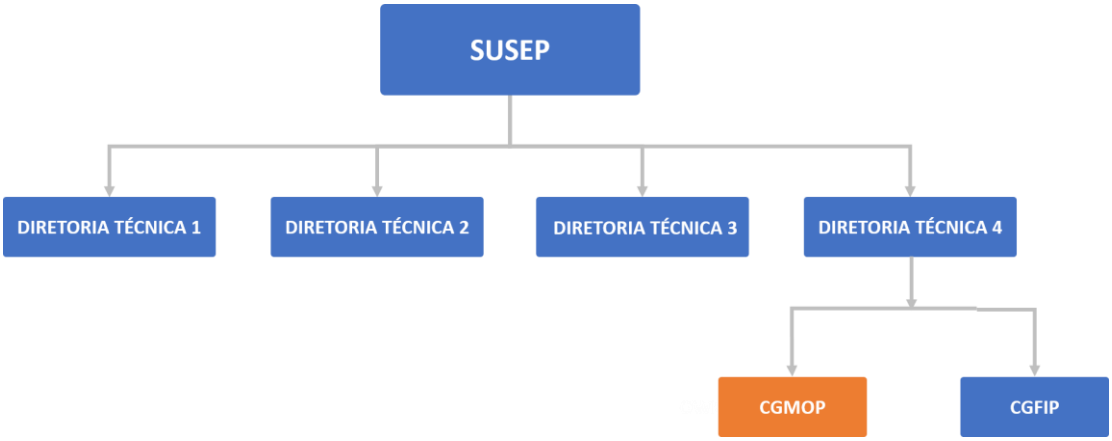

Figura 1. Organograma da Susep

Com o intuito de fazer cumprir todos os objetivos traçados no Decreto-Lei Nº 73/66, o próprio decreto obriga, em seu artigo 88, as sociedades supervisionadas ao fornecimento de dados e informações atinentes a quaisquer aspectos de suas atividades. Desta forma, as companhias seguradoras encaminham mensalmente uma vasta gama de informações, que vão desde dados contábeis de nível mais geral a outros de menor granularidade (seja por ramo de seguro ou até mesmo por apólice).

1.3. A Supervisão Prudencial de Seguros no Brasil

Em consonância com os princípios que norteiam as recomendações da International Association of Insurance Supervisors (IAIS), organização da qual o Brasil é membro, e o projeto Solvência II da União Europeia, a Susep utiliza uma abordagem para controle e redução dos riscos a que estão expostas as sociedades seguradoras baseada em três pilares (Melo & Neves, 2012).

Em síntese, o primeiro pilar trata sobre requisitos quantitativos de capital; o segundo trata sobre requisitos qualitativos de supervisão e gerenciamento de riscos, enquanto o terceiro, sobre harmonização das informações para fins de supervisão e divulgação pública (Melo & Neves, 2012).

Especificamente no que tange à adequação de capital (Pilar 1), a regulação dos capitais adicionais baseados em risco busca aumentar o nível de solvência das empresas, tornando possível honrar o compromisso de pagar indenizações aos segurados, o que, por sua vez, é fundamental para a credibilidade e estabilidade do mercado nacional em um ambiente globalizado (Melo & Neves, 2012).

Neste contexto, a Susep regulamentou a exigência de Capital Mínimo Requerido (CMR), que é dado pela seguinte fórmula:

$$CMR = \max\left(CB;\ CR_{oper} + \sqrt{\sum\sum \rho_{ij}\ x\ CR_i\ x\ CR_j}\right) \quad (1)$$

Onde:

CB: Capital base

CR_{oper}: Parcela do capital de risco operacional

ρ_{ij}: Correlação entre as parcelas de capital "i" e "j"

CR_i e CR_j: Parcelas do capital de risco de subscrição (CR_{subs}), de crédito (CR_{cred}) e de mercado (CR_{merc})

Para efeito de comparação com o CMR, utilizamos o Patrimônio Líquido Ajustado (PLA), que é calculado com base no patrimônio líquido contábil da empresa, após serem processados ajustes de ordem contábil ou aqueles associados à variação dos valores econômicos.

O valor do PLA acima do CMR configura uma situação de suficiência de capital ou, em outros termos, que a companhia possui ativos para cumprir suas obrigações com determinado nível de confiança. Contrariamente, a situação em que o valor do PLA é menor do que o CMR é denominada insuficiência de capital.

Situações de insuficiência de capital requerem cuidado por parte do órgão supervisor e, dependendo do nível desta insuficiência, disparam diferentes formas de intervenção, como o Plano de Regularização de Solvência (PRS), Direção-fiscal ou Liquidação Extrajudicial. Detalhes acerca de cada uma destas formas de intervenção fogem ao escopo deste trabalho.

1.4. A IMPORTÂNCIA DA UTILIZAÇÃO DE MODELOS PREDITIVOS

As seções anteriores demonstraram a importância do mercado de seguros e de que as empresas que o compõem se encontrem saudáveis do ponto de vista econômico-financeiro. A abordagem de supervisão baseada em riscos implementada pela Susep, utilizando como referência o regime Solvência II e as diretrizes da IAIS, abarca requisitos quantitativos de capital, calculados por meio de modelagem estatística.

Ainda, conforme já citado, situações de insuficiência de capital podem ensejar a aplicação de mecanismos de intervenção por parte do órgão regulador nas companhias supervisionadas. Muito embora tais mecanismos desempenhem um papel importante no arcabouço regulatório, é desejável que essas situações possam ser previstas com razoável antecedência de forma a evitar que se concretizem situações de insuficiência de capital.

Neste sentido, este trabalho propõe a utilização de modelos preditivos, mais especificamente a classe de algoritmos baseados em Machine Learning (ML), para sinalização antecipada de situações de insuficiência de capital em sociedades seguradoras que operem com seguros ou previdência complementar, bem como para sociedades resseguradoras. Foram excluídas da análise as sociedades de capitalização, pela característica extremamente diferenciada de seus produtos relativamente às demais.

Machine Learning pode ser definido como o processo automatizado de extrair padrões a partir dos dados (Kelleher et al., 2015). Para a construção dos modelos usados em análises preditivas, utilizam-se algoritmos de aprendizagem supervisionada, assunto que será visto em maiores detalhes no decorrer desta dissertação.

A utilização de algoritmos de Machine Learning tem se mostrado bem-sucedida a partir de dados reais por empresas atuantes em diversos campos do conhecimento: finanças, marketing, processamento de imagem, reconhecimento de voz, classificação documental, dentre outros.

1.5. DEFINIÇÃO DO PROBLEMA

No problema a ser estudado nesta dissertação, utilizaremos o que a literatura denomina aprendizagem supervisionada, ou seja, o conjunto de dados utilizados para alimentar o modelo inclui tanto as características descritivas do modelo (variáveis independentes) quanto a solução desejada (variável dependente ou *label*).

1.5.1. Variável Dependente

Após breve explanação na seção 1.3 sobre a supervisão prudencial, caracterizamos a condição de insuficiência de capital para uma companhia seguradora quando o seu PLA assume um valor calculado inferior ao CMR. Neste caso, a nossa variável dependente (*label*), chamada INSUFICIÊNCIA, tomará, por conseguinte, o valor 1.

Por sua vez, a situação inversa, em que o PLA da companhia assume um valor acima do CMR, configura uma situação de suficiência de capital, motivo pelo qual a variável INSUFICIÊNCIA assumirá o valor 0.

Trata-se, pois, de um problema de classificação binária. Apenas para reforçar o conceito, os algoritmos serão treinados com diversos exemplos de empresas em diferentes períodos de tempo juntamente com a sua classe (suficiente ou insuficiente) e deverão ser capazes de classificar como suficientes ou insuficientes as novas empresas em um período de tempo futuro.

Por fim, cabe uma importante observação: há no mercado segurador brasileiro empresas que compõem grandes conglomerados financeiros ou seguradores, em que a oferta de capital para suprir situações adversas decorrentes da operação é muito superior a algumas companhias independentes. Nestes casos, é comum que, no período imediatamente anterior à configuração de uma situação de insolvência, o grupo controlador realize o que chamamos de aporte de capital.

Os aportes de capital permitem qua a empresa receba dinheiro novo e resolvam a insuficiência ocasional, aumentando o PLA na exata quantia do aporte, mas acabam por confundir o classificador durante o período de treinamento, mascarando a real situação operacional. Para mitigar esse problema, fizemos a opção por deduzir o aporte de capital do PLA no cálculo da suficiência de capital.

1.5.2. Variáveis Explicativas

No presente projeto, serão testadas diversas variáveis, cujos campos temáticos principais estão definidos nos subitens a seguir. Aproveitamos para informar que os indicadores abaixo relacionados constituem uma lista meramente exemplificativa, não esgotando em sua totalidade as variáveis testadas, tendo apenas como objetivo de ilustrar o teor geral do trabalho.

1.5.2.1. Indicadores Macroeconômicos

Os indicadores macroeconômicos são grandezas de caráter econômico, que permitem aferir o andamento da economia de um país. O comportamento destes não está diretamente associado ao comportamento de cada empresa ou apenas do setor de seguros, tendo em vista que buscam capturar movimentos de ordem mais geral da economia de um país.

Especificamente no que se refere a situações de insolvência, as variáveis macroeconômicas podem assumir grande importância, tendo em vista que ciclos econômicos recessivos conduzem ao aumento do número de empresas em dificuldades financeiras, enquanto em períodos expansionistas este número diminui (Pacheco, 2018). Temos como exemplos utilizados no trabalho: crescimento do PIB, taxa de juros, desemprego, inflação e renda mensal média.

1.5.2.2. Indicadores Financeiros

Indicadores financeiros são indicadores utilizados para realizar uma avaliação individual das empresas com base na evolução de suas principais contas integrantes do Balanço Patrimonial e da Demonstração de Resultado do Exercício. Na atividade de monitoramento do mercado, as áreas competentes da Susep já fazem uso regular destes indicadores para detectar movimentos discrepantes em relação aos pares de mercado (análise vertical) ou, ainda, a diferentes janelas temporais para a mesma companhia (análise horizontal). São exemplos de indicadores financeiros utilizados no trabalho: Índice Combinado Ampliado (ICA), Índice de Sinistralidade (ISIN), Índice de Liquidez Corrente (ILC), Cobertura de Provisões Técnicas por Ativos Garantidores e *Return on Equity* (ROE).

1.5.2.3. Demais Indicadores

No que concerne às demais variáveis no nível individual da companhia, utilizamos diversos indicadores relacionados a características gerais da empresa. Neste caso, podem ser:

- variáveis que reflitam a atuação da empresa em segmentos de mercado: seguros, previdência complementar ou resseguro;
- variáveis que reflitam a atuação da empresa em determinados ramos de seguro: automóveis, transportes, responsabilidade, dentre outros;
- variáveis que reflitam características gerais: porte da companhia (número de funcionários), pertencimento a algum conglomerado financeiro ou segurador e a estrutura societária da empresa (número de acionistas, número de níveis da árvore de acionistas); e
- variáveis que indiquem problemas relacionados aos controles internos da companhia, que podem ocasionalmente refletir em sua solvência: número de solicitações de recargas realizadas pela empresa, ou seja, quantas vezes a companhia solicitou reenvio de dados motivada por inconsistência na remessa de dados anterior.

1.5.3. Defasagem Temporal

Conforme já referido anteriormente, o objetivo geral do trabalho é a predição da situação de insuficiência de capital com base em indicadores macroenonômicos, econômico-financeiros e outros indicadores gerais da companhia.

De forma a tornar mais clara a modelagem, é importante ressaltar que, em nossos testes, cada variável explicativa foi defasada de 1 ou 2 anos ($x_{i,t-1}$ e $x_{i,t-2}$) em relação à variável dependente (y_t). Quando pertinente, foi calculada também a variação percentual entre os períodos "t-2" e "t-1". Ou seja, ao prevermos a insuficiência de capital de uma empresa no ano de 2017, utilizamos, por exemplo, os valores do PIB, do Índice Combinado Ampliado e do Índice de Sinistralidade nos anos de 2016 e 2015 para explicá-la, bem como a variação destes indicadores entre os anos de 2015 e 2016.

1.6. RESTRIÇÕES E LIMITAÇÕES

Os dados utilizados para treinamento, validação e teste dos modelos aplicados nesta dissertação foram fornecidos pela SUSEP, tendo em vista a condição do autor de analista técnico integrante do quadro de pessoal da Autarquia.

Devido a questões de confidencialidade, informo que todo e qualquer dado aqui utilizado passou por um processo de anonimização, motivo pelo qual não serão descritos ou citados quaisquer nomes de sociedades supervisionadas, nem a sua condição de suficiência de capital no período de análise.

1.7. ESTRUTURA DO TRABALHO

O restante desta dissertação se encontra organizado da forma que se segue.

O Capítulo 2 fornece uma visão geral acerca da literatura relacionada com a elaboração de sistemas de sinalização antecipada para monitoramento da solvência de sociedades seguradoras.

O Capítulo 3 discute sobre o referencial teórico, abarcando a metodologia CRISP-DM e suas etapas, as ferramentas utilizadas neste trabalho, os algoritmos utilizados para treinamento e validação dos modelos e as técnicas usadas para redução da dimensionalidade e para equilíbrio do número de instâncias entre as 2 classes principais (suficiente e insuficiente).

Já no Capítulo 4, são apresentadas as etapas de coleta, preparação e processamento dos dados.

Mais à frente, no Capítulo 5, apresentamos as métricas utilizadas para avaliação dos modelos e discutimos os resultados encontrados, bem como fazemos uma comparação entre as diferentes abordagens.

Por fim, no Capítulo 6, concluímos o trabalho, apresentando apontamentos que se enquadram como limitações ao trabalho atual e possibilidades de melhorias para futuros projetos que visem à continuidade do que foi ora apresentado.

2. REVISÃO DA LITERATURA

No presente capítulo, trataremos da literatura especializada no assunto, notadamente acerca dos artigos científicos que servem de referência para a construção de modelos de previsão de insolvência em seguradoras, quer no Brasil, quer em âmbito internacional.

As referências são vastas, mas, nos primórdios, tendiam para a utilização de técnicas estatísticas consagradas, como os modelos de regressão logística e análise discriminante. Recentemente, em trabalhos internacionais, passou-se a fazer uso de todo o arcabouço teórico de Machine Learning, mas este ainda se mostra muito incipiente no Brasil, havendo farto campo para estudo.

Brocket et al. (1994) apresentaram um trabalho que objetiva a construção de um sistema de sinalização antecipada de previsão de insolvência para uso dos reguladores. A abordagem utilizada é baseada em redes neurais (arquiteturas bem simples), por meio do uso de dados financeiros e outras operações de seguros disponíveis, e considera uma amostra de seguradoras americanas que operam em seguros de danos e responsabilidades e se tornaram insolventes nos anos de 1991 e 1992. Os resultados obtidos também foram comparados com aqueles obtidos via análise discriminante. Ademais, uma das características ressaltadas relativamente ao modelo é a possibilidade de atualizá-lo sem ser inteiramente retreinado.

Brocket et al. (2006) ampliaram os estudos anteriormente existentes, trazendo algumas inovações aplicadas a empresas que operam com seguro de vida: utilizaram dois tipos diferentes de redes neurais (ainda modelos mais simples do que que os existentes atualmente) e realizaram comparações com análise discriminante multivariada e regressão logística; tentaram identificar o conjunto de variáveis que mais colaboravam na previsão de insolvência; não usaram somente empresas publicamente declaradas como insolventes, mas agregaram informação de empresas em dificuldades financeiras identificadas pela Unidade de Análise Financeira do Departamento de Seguros do Texas; implementação de um modelo que tenta prever a ruína financeira no ano corrente e outro modelo para prever com um ano de antecedência; e, por fim, a implementação de teste fora da amostra.

Bezerra & Corrar (2006), em uma abordagem diferente e focada no mercado segurador brasileiro, propuseram uma pesquisa com vistas à determinação das variáveis financeiras mais relevantes, por meio da utilização de análise fatorial, que devem pautar a atuação dos stakeholders no acompanhamento dos resultados das empresas seguradoras. Embora não esteja no escopo da pesquisa a construção de um modelo de previsão, o artigo colabora com o levantamento de indicadores financeiros importantes para o mercado em análise.

Ainda focado no Brasil, Okamura & Lima (2013) elaboraram um trabalho para desenvolvimento de um modelo de regressão logística para estudo da capacidade de dados contábeis preverem a insolvência de seguradoras brasileiras atuantes nos ramos de seguros de pessoas. O estudo abrangeu 138 entidades no período de janeiro de 1999 a dezembro de 2010. A variável dependente foi construída a partir da classificação das empresas em regime especial, que é um regime de intervenção mais severo, diferentemente do proposto nesta tese, em que empresas com dificuldade financeira em fase inicial já são capazes de sensibilizar o modelo e integrar a classe positiva.

Ademais, um trabalho mais recente e bem completo foi descrito por Constantino (2020), em que, novamente, recorreu-se a modelos de regressão logística na tentativa de tentar predizer, a partir de

indicadores financeiros, uma situação de insolvência no mercado segurador brasileiro. Uma diferença fundamental relativamente ao trabalho desenvolvido nesta tese diz respeito à forma como a variável dependente foi construída, tendo sido consideradas como insolventes não apenas as empresas que apresentaram insuficiência de capital, como aquelas cujo valor da suficiência de capital estiverem abaixo do primeiro quartil em relação a todas as empresas naquele período. Ou seja, se, em um dado ano, todas as empresas estiverem com uma situação confortável, ainda assim as que se situarem abaixo do primeiro quartil, constarão como amostras pertencentes à classe insolvente.

É importante destacar, ainda, dois trabalhos recentes que fazem uso de modelos de machine learning para previsão de insolvência de seguradoras. O primeiro foi desenvolvido em Rustam & Saragih (2021), com o uso de Random Forest, de forma a comparar com resultados de um estudo anterior, em que foram utilizados os mesmos dados e aplicados os modelos SVM e Fuzzy Kernel C-Means. As variáveis que serviram de entrada ao modelo seguem em linha com outros trabalhos publicados – indicadores financeiros calculados a partir de dados contábeis. O conjunto de dados, por sua vez, é bem reduzido, com 72 empresas, e não houve um acompanhamento temporal das empresas (ou seja, não representa um estudo longitudinal).

O outro estudo, apresentado em Tian et al. (2019), inovou ao aplicar uma técnica de SVM denominada non-kernel fuzzy quadratic surface support vector machine (FQSSVM), que evita a busca inglória pelo kernel apropriado e seus respectivos parâmetros, para previsão de insolvência de seguradoras que operam em ramos não-vida nos Estados Unidos da América. Além disso, houve uma investigação, por meio de seleção de variáveis, do subconjunto que gerava os melhores resultados e a incorporação de variáveis explanatórias relacionadas a mudanças nas condições macroeconômicas, o que colabora para evitar a volatilidade introduzida pelas tendências temporais durante um período longo. O modelo implementado foi capaz de desbancar outros modelos tradicionais, como redes neurais e o SVM padrão.

Percebemos, pelos trabalhos descritos ao longo deste capítulo que há uma escassez de projetos que utilizem modelos de machine learning e todo o framework disponível nesta área no Brasil, para realizar previsão de insolvência. Além disso, o presente trabalho agrega tanto pelo uso de variáveis macroeconômicas, quanto pela inclusão de variáveis de capital baseado em risco e outras variáveis qualitativas, que dizem respeito aos controles internos das companhias.

3. REFERENCIAL TEÓRICO

No presente capítulo, apresentaremos a metodologia utilizada para abordar o problema, com a descrição de todos os passos, bem como os principais algoritmos e parâmetros utilizados no processo de modelagem.

3.1. A METODOLOGIA CRISP-DM

CRISP-DM, acrônimo para Cross-Industry Standard Process for Data Mining, é uma metodologia amplamente utilizada pela indústria na implementação de projetos de Data Mining/Machine Learning. Sua concepção se deu no ano de 1996 e tem por característica ser flexível o suficiente para ser aplicado independentemente do tipo de negócio, ferramenta ou aplicação. É uma abordagem baseada inteiramente no aprendizado prático de seus colaboradores – e não de forma acadêmica sustentada apenas em princípios técnicos –, tendo sido construída a partir de experiências realizadas no mundo real, após diversas tentativas e ajustes.

De acordo com Piatestky (2014), trata-se da metodologia mais utilizada em projetos desta natureza, abrangendo 43% do total de respostas à pesquisa. Em relação à pesquisa anterior realizada pelo mesmo autor em 2007, notou-se uma manutenção tanto no que se refere à liderança do ranking, quanto ao nível percentual.

Há, ainda, na literatura 2 outras metodologias bem documentadas (o que não exclui a utilização de abordagens próprias por pessoas ou organizações): SEMMA (Sample, Explore, Modify, Model, and Assess) e KDD Process (Knowledge Discovery in Database). Na mesma pesquisa, pode-se notar que a metodologia SEMMA apresentou uma forte queda de utilização, de 13% para 8,5%, enquanto a utilização de KDD manteve-se estável, embora em um patamar inferior ao SEMMA.

A abordagem CRISP-DM, considerada, portanto, o padrão da indústria, é composta por 6 etapas básicas e busca compreender o ciclo de vida de um projeto de Data Mining. É importante notar que, embora estejam encadeadas de maneira lógica, a sequência das etapas não é rígida, sendo comum e até necessário no decurso do projeto mover-se para etapas anteriores e posteriores do ciclo a depender dos resultados da fase atual. As setas da figura 2 indicam as mais importantes e frequentes dependências entre as diversas fases (Chapman et al., 2000).

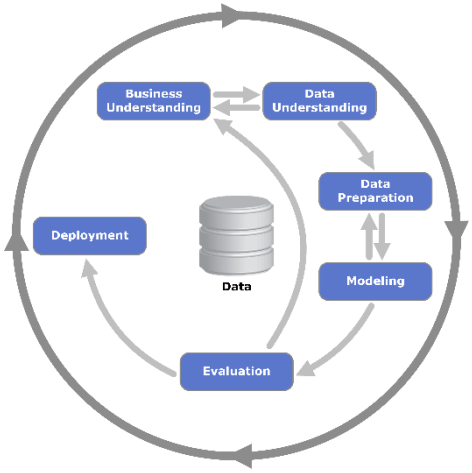

Figura 2. Etapas da metodologia CRISP-DM

Nos subitens a seguir, apresentamos uma breve descrição de cada uma das fases que compõem a metodologia.

3.1.1. Business Understanding

Esta é a primeira etapa do ciclo, momento em que se deve definir os requerimentos e objetivos do projeto. Nesta etapa, é importante entender que tipo de problema o cliente deseja que seja resolvido, eliminando objetivos concorrentes e compreendendo eventuais limitações. O correto desenvolvimento desta etapa é fundamental para o sucesso do projeto final.

3.1.2. Data Understanding

Esta etapa compreende a coleta dos dados, sua descrição geral (formato dos dados, número de registros e colunas em cada tabela), exploração (primeiros achados e hipóteses iniciais, que podem ser identificados por meio de gráficos) e verificação da qualidade (abrangência de todos os casos que o problema requer, existência de erros e frequência, identificação de valores nulos e frequência). De forma resumida, é a etapa para extração das primeiras impressões do conjunto de dados com o qual trabalharemos e nos permite adquirir uma noção sobre a confiabilidade do resultado final.

3.1.3. Data Preparation

Esta etapa contempla basicamente 4 subetapas, a saber: seleção dos dados (quais dados permanecerão e quais serão excluídos), limpeza dos dados (solucionar os problemas reportados durante a fase 3.1.2. Data Understanding), construção de dados (criar novos atributos a partir de um ou mais atributos existentes), integração dos dados (fundir diferentes tabelas, realizar agregações) e formatação dos dados.

Ou seja, esta etapa requer o processamento dos dados de forma a torná-los passíveis de serem utilizados como entrada pelos algoritmos de modelagem. Muitos projetos acabam por consumir a maior parte do tempo neste passo.

3.1.4. Modelling

A próxima etapa deve ser entendida como chave em projetos de Machine Learning. Compreende, resumidamente o seguinte: seleção dos algoritmos que serão testados (Logistic Regression, Random Forest, dentre outros), separação do conjunto de dados em subgrupos para treinamento, validação e teste dos algoritmos, definição dos hiperparâmetros e valores para cada algoritmo (incluindo a utilização de algoritmos para calibragem), e resumo dos resultados para definição do modelo considerado apropriado.

Esta etapa pode levar à 3.1.5. Evaluation ou, a depender da qualidade dos resultados, retornar para a fase de preparação.

3.1.5. Evaluation

A etapa de avaliação de resultados pode ser brevemente descrita pelos seguintes passos: avaliação dos resultados com respeito aos objetivos definidos na fase inicial de entendimento do negócio, compreensão a respeito de atividades que possam não ter sido adequadamente abordadas ou devem ser repetidas e, por fim, a determinação dos próximos passos.

3.1.6. Deployment

Neste momento, são desenvolvidas as tarefas de implementação do projeto (por exemplo, por meio do armazenamento dos resultados em uma base de dados para posteriormente alimentar um dashboard que auxilie o corpo diretivo de uma empresa no processo de tomada de decisões), a elaboração de um plano de manutenção e monitoramento, produção do relatório final e uma revisão geral.

A etapa de entendimento do negócio foi cumprida e apropriadamente abordada no decorrer da explanação realizada no Capítulo 1 desta dissertação. As etapas de compreensão e preparação dos dados e seus correspondentes resultados e *insights* serão apresentadas no Capítulo 4. A etapa de modelagem será descrita na seção 3.3., em que faremos uma descrição pormenorizada de todos os algoritmos que serão testados e os respectivos hiperparâmetros a calibrar. A etapa de avaliação se encontrará descrita no Capítulo 5. Por fim, a etapa de deployment depende da avaliação da pertinência de implementação por parte do corpo diretivo da Autarquia.

3.2. FERRAMENTAS

No processo de coleta dos dados utilizados nesta tese e, posteriormente, tratamento e modelagem com a utilização dos algoritmos de Machine Learning, compreendendo todas as etapas referenciadas pela metodologia CRISP-DM (descrita na seção 3.1.), utilizamos basicamente 2 ferramentas: Python e SQL Server Management Studio.

3.2.1. SQL Server Management Studio (SSMS)

SQL Server Management Studio é um ambiente de desenvolvimento integrado (IDE) que nos fornece uma interface gráfica para conectar e trabalhar com MS SQL Server. Este último, por sua vez, é um sistema de gerenciamento de banco de dados relacional desenvolvido pela Microsoft, sendo de larga utilização e um dos mais importantes sistemas do mercado. Utilizando-se de consultas, realizadas através da linguagem Transact-SQL, é possível realizar a recuperação e transformação dos dados armazenados.

A totalidade dos dados encaminhados pelas seguradoras por meio do Formulário de Informações Periódicas (FIPSUSEP) são armazenados em diferentes bancos de dados internos. Assim, o processo de

coleta dos dados brutos e, ainda, a construção de algumas variáveis se deu por meio da utilização deste software.

3.2.2. Python

Python é uma linguagem criada em 1991 e, desde então, tem-se tornado uma das mais utilizadas no universo da programação, especialmente quando falamos de Data Science e Machine Learning. Trata-se de uma linguagem extremamente flexível e grande parte de sua popularidade se deve às centenas de bibliotecas (pedaços de código para reutilização que podem de forma simples serem incluídos em outros projetos) disponíveis, capazes de colaborar em problemas de qualquer natureza. Em outras palavras, horas de programação deixam de ser desperdiçadas com a utilização dessas bibliotecas contendo funções predefinidas.

O trabalho como cientista de dados se vê imensamente facilitado pela utilização de bibliotecas como pandas, NumPy e scikit-learn. NumPy mostra-se como uma excelente escolha para trabalhar com dados numéricos em Python principalmente pela sua performance ao lidar com vetores multidimensionais. Pandas aproveita-se da excelente performance de NumPy, conjugada com a grande flexibilidade para manipulação de dados em formato tabular em estruturas conhecidas por *dataframes*, para se estabelecer como um poderoso ambiente para manipulação e análise de dados. Scikit-learn, por sua vez, tornou-se a mais importante ferramenta voltada para Machine Learning desde sua concepção. Conforme veremos mais adiante, scikit-learn inclui implementações para algoritmos de classificação (SVM, KNN, Random Forest, etc.), regressão (Lasso, Ridge Regression, etc.), clusterização (k-means, hierarchical clustering, DBSCAN, etc.), redução de dimensionalidade (PCA, seleção de variáveis, etc.), seleção de modelos (GridSearch, cross-validation, etc.), preprocessamento de dados (normalization, etc.) (McKinney, 2017).

As bibliotecas referidas não esgotam a totalidade dos recursos que a linguagem Python oferece, tampouco os que foram utilizados neste trabalho. No momento oportuno e, quando cabível, descreveremos outras bibliotecas utilizadas (por exemplo, imblearn para tratar o desbalanceamento dos dados) ou funções manualmente implementadas (por exemplo, time series cross-validation, tendo em consideração a componente temporal dos dados que utilizamos).

Ademais, há uma consideração que é válida de se fazer: a linguagem Python fornece a possibilidade de utilização de linguagem SQL diretamente em seu ambiente tanto para conexão, quanto para consulta a dados. Todavia, por questões de segurança e perfil de acesso à base de dados da Susep, houve a necessidade de que o autor realizasse o acesso diretamente pelo SSMS e as tabelas resultantes fossem salvas em formato “csv” para posterior importação em ambiente Python.

No decorrer deste trabalho, utilizamos o Python em sua versão 3.7.6.

3.3. ALGORITMOS

Nesta seção, descreveremos os principais algoritmos e técnicas utilizados nos processos de tratamento e limpeza dos dados, extração e seleção de variáveis, modelagem e as métricas definidas para avaliação dos modelos.

3.3.1. Logistic Regression

Logistic Regression, também conhecido por Logit Regression, é um algoritmo utilizado em problemas de classificação. No nosso projeto, tratamos de um problema de classificação binária, embora o algoritmo seja capaz de ser implementado em problemas envolvendo mais de 2 classes, como, por exemplo, se tivéssemos uma variação do projeto atual para contemplar a modelagem de 3 classes: insuficiente, pouco suficiente (até 20% de suficiência) e muito suficiente (acima de 20% de suficiência).

Este algoritmo se enquadra em uma classe de algoritmos conhecida por *error-based learning*. Isto significa que, partindo de um conjunto de parâmetros, realizaremos uma busca objetivando minimizar o erro total de previsão do modelo baseado nos dados utilizados durante o processo de treinamento. A ideia por trás desta classe de algoritmos é:

- Parte-se de um conjunto aleatório de parâmetros;
- Realizam-se previsões com base no modelo inicializado com os parâmetros do passo anterior;
- Comparam-se as previsões com o resultado real para todas as instâncias de treinamento;
- Utiliza-se uma função que permita resumir este erro de previsão e torná-lo comparável;
- Ajustam-se os parâmetros iterativamente com a finalidade de tornar o modelo mais acurado.

Logistic Regression funciona de maneira muito semelhante ao conhecido modelo de regressão linear, muito utilizado quando estamos lidando com problemas de regressão (em que a variável resposta é numérica e não categórica). Neste, após estimados os parâmetros, calculamos a soma ponderada dos valores de cada variável descritiva pelos parâmetros. Naquele, há um passo adicional, que é aplicar a soma ponderada como variável de entrada em uma função logística. O resultado pode ser interpretado como a probabilidade de uma determinada instância pertencer à classe positiva (ou classe 1). Matematicamente, temos que:

$$\hat{p} = h_\theta(\mathbf{x}) = \sigma(\mathbf{x}^T \boldsymbol{\theta}) \tag{2}$$

A função *logistic* é definida conforme a equação 3 e seu comportamento pode ser visualizado na figura 3.

$$\sigma(t) = \frac{1}{1+e^{-t}} \tag{3}$$

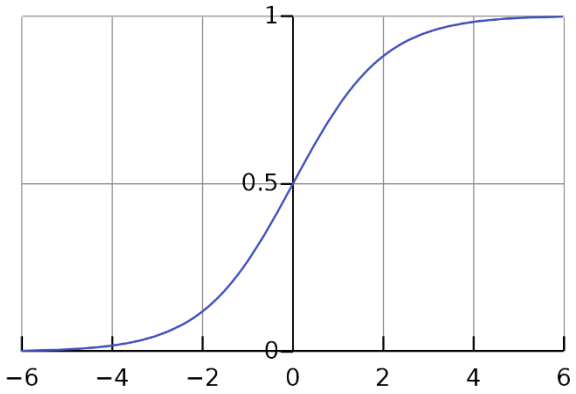

Figura 3. Gráfico da função *logistic*

A partir da observação da figura 3, fica claro que a função mostrada produz uma transformação não-linear nas variáveis introduzidas e assegura que a saída da função varia apenas no intervalo entre 0 e 1, compatível com a interpretação de que este resultado é uma probabilidade.

Uma vez estimados os parâmetros da Logistic Regression e, consequentemente, a probabilidade de uma observação pertencer à classe 1 (relembrando que em nosso problema, trata-se da classe "insuficiente"), o valor previsto final para a variável dependente é definido de acordo com o seguinte:

$$\hat{y} = \begin{cases} 0, se\ \hat{p} < 0{,}5 \\ 1, se\ \hat{p} \geq 0{,}5 \end{cases} \quad (4)$$

Traduzindo, se a probabilidade estimada for maior ou igual a 0,5, a classe prevista para esta observação é a classe 1 (insuficiente, no caso em estudo). De forma contrária, se a probabilidade estimada é menor que 0,5, a classe prevista é a classe 0 (suficiente, no caso em estudo).

Para obtermos a estimação do conjunto dos parâmetros $\boldsymbol{\theta}$, primeiramente precisamos de definir uma função custo, que, para uma única instância de treinamento, é dada pela equação 5.

$$c(\boldsymbol{\theta}) = \begin{cases} -\log(\hat{p})\,, se\ y = 1 \\ -\log(1-\hat{p})\,, se\ y = 0 \end{cases} \quad (5)$$

A ideia que envolve a função acima é a seguinte: imaginemos uma observação cujo valor real da variável dependente é 1. Um modelo apropriado deveria otimizar parâmetros que resultassem uma probabilidade estimada próxima de 1, para este caso. Caso isto se confirme, o valor de $-\log(\hat{p})$ será muito próximo de 0, o que significa que o custo é muito baixo (algo esperado). De forma diversa, caso a probabilidade estimada pelo modelo seja próxima de 0, $-\log(\hat{p})$ assumirá um valor alto, o que, mais uma vez, é desejável, pois estamos prevendo a classe 0, quando o correto seria 1.

O raciocínio análogo vale quando o valor real da variável dependente é 0. Neste caso, a função custo $-\log(1-\hat{p})$ resultará em um valor alto se a probabilidade prevista é próxima de 1 e em um valor baixo caso a probabilidade prevista seja próxima de 0.

Considerando um conjunto de treinamento de tamanho m, a função custo a ser minimizada para obtenção dos parâmetros ótimos é dada pelo custo médio de todas as instâncias de treinamento, conforme pode ser observado na equação 6.

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(\hat{p}^{(i)}\right) + (1-y^{(i)})\log\left(1-\hat{p}^{(i)}\right)\right] \quad (6)$$

Há 2 pontos que merecem ser ressaltados no que concerne à função de otimização. O primeiro é que não possui solução fechada para o cálculo dos parâmetros diferente do caso da regressão linear. O segundo é que, por ser uma função convexa, o processo de otimização, garantidas certas condições, encontrará o mínimo global para a função.

Outra diferença relativamente à regressão linear é o fato de que os coeficientes estimados não possuem interpretação direta, mas influenciam na taxa de crescimento ou decrescimento da curva.

A implementação deste algoritmo em Python utilizada no trabalho é dada pela função sklearn.linear_model.LogisticRegression(). Há uma série de hiperparâmetros passíveis de ajuste e calibração, cujos resultados serão demonstrados mais adiante.

Há 4 parâmetros comumente utilizados para calibração: *penalty*, *C*, *solver* e *class_weight*. O 2 primeiros estão relacionados com a ideia de regularização, que é um método para penalizar modelos muito complexos e que permite reduzir sua variância (Albon, 2018). Em outras palavras, quando aplicamos algum método de regularização, buscamos reduzir o valor dos coeficientes estimados. Desta forma, os valores preditos pelo modelo se tornam menos suscetíveis a alterações e ainda reduzimos a possibilidade de *overfitting*.

A forma utilizada para regularizar um modelo é inserindo uma componente extra na função de otimização apresentada pela equação 6. Na penalização do tipo L1, temos o fator adicional $\alpha \sum_{j=1}^{p} |\hat{\beta}_j|$, enquanto na penalização do tipo L2, utilizamos um fator do tipo $\alpha \sum_{j=1}^{p} \hat{\beta}_j^2$. A diferença principal, como podemos facilmente notar, é que L2 penaliza mais fortemente altos valores para os coeficientes $\hat{\beta}_j$.

O parâmetro α, que aparece em ambos os tipos de regularização apresentados, é denominado "força de regularização". A implementação disponível pela função LogisticRegression, utiliza uma versão ligeiramente modificada de α, dada pelo inverso do seu valor, que é conhecida por *C*. Assim, valores mais baixos para C implicam o aumento da penalização para valores mais altos dos parâmetros (modelos mais complexos).

O hiperparâmetro *solver* é utilizado para especificar qual técnica será utilizada para otimização dos coeficientes do modelo através da minimização da função custo apresentada anteriormente. Sem entrar em detalhes que fogem ao escopo deste trabalho e apenas para dar uma ideia geral de sua utilidade, para datasets muito grandes, Stochastic Gradient Descent é capaz de treinar modelos muito mais rapidamente do que outros *solvers* (Albon, 2018).

O último hiperparâmetro que abordaremos é chamado *class_weight* e nos permite lidar com um problema muito comum ao tratarmos com dados reais: dados desbalanceados. Muitas vezes, o número de instâncias de treinamento que compõem uma das classes em um *dataset* é muito superior ao das demais classes. Embora haja outros métodos específicos para tratamento deste caso (trataremos disto mais adiante), *class_weight* nos oferece uma opção extra, já incorporada ao modelo.

3.3.2. K-Nearest Neighbors

O modelo conhecido por K-Nearest Neighbors é um dos algoritmos usados em Machine Learning de mais fácil entendimento e interpretação, tendo em vista que seu algoritmo replica de certo modo a forma que pessoas utilizam para realizar previsões, baseando-se em ocorrências similares armazenadas na memória. É, pois, considerado um modelo de previsão baseado em similaridade (Kelleher et al., 2015).

Resumidamente, a ideia por trás deste algoritmo, em sua abordagem padrão, é:

- Armazenam-se todas as instâncias de treinamento;

- Calculam-se as distâncias entre a instância cuja classe desejamos prever e todas as instâncias pertencentes ao conjunto de treinamento;
- Selecionam-se os *k* vizinhos mais próximos;
- A classe majoritária dentre o conjunto selecionado no passo anterior é determinada como a classe prevista.

Há diversas formas de quantificação da similaridade entre 2 observações. Em geral, utilizamos a distância euclidiana ou, em menor escala, a distância de Manhattan. Ambas são casos particulares da distância de Minkowski, que é dada pela seguinte equação.

$$Minkowski(\mathbf{x}, \mathbf{y}) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p} \quad (7)$$

Onde $\mathbf{x} = (x_1, x_2, \dots, x_n)$ e $\mathbf{y} = (y_1, y_2, \dots, y_n) \in \mathbb{R}^n$, ou seja, são instâncias de um conjunto de treinamento com n variáveis descritivas.

A distância euclidiana assume sua forma para o caso em que $p = 2$. A distância de Manhattan, por sua vez, é definida para o caso em que $p = 1$. Uma consequência direta da equação de Minkowski é que, quanto maior o valor de p, maior será o efeito de uma grande diferença no valor de uma variável para 2 diferentes observações. Assim, podemos concluir que, a depender da métrica utilizada pelo algoritmo, poderemos obter previsões diferentes. A figura 4 nos oferece uma interpretação visual das distâncias euclidianas e de Manhattan para 2 pontos pertencentes ao espaço $\mathbb{R}^2$.

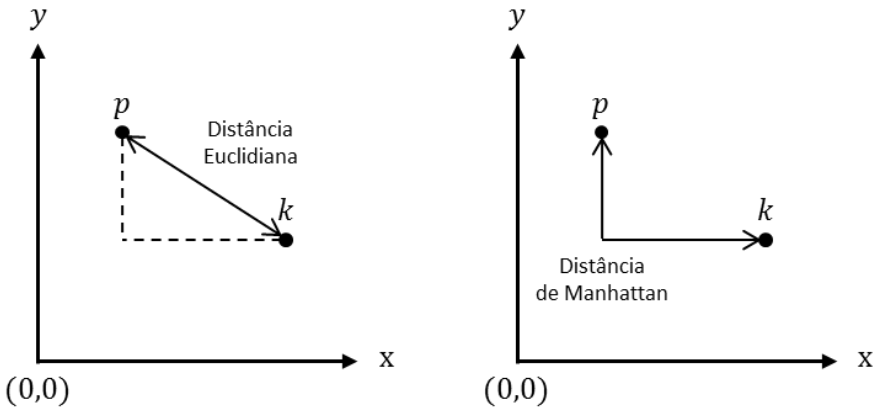

Figura 4. Diferença entre a distância euclidiana e a distância de Manhattan

O classificador K-Nearest Neighbors é um algoritmo não-paramétrico, ou seja, o número de parâmetros utilizados pelo modelo aumenta conforme o número de instâncias de treinamento também aumenta (Kelleher et al., 2015). Assim, quando novas instâncias são adicionadas ao conjunto de dados, a complexidade para representação do modelo aumenta conjuntamente. Opostamente, em um modelo paramétrico, tal qual Logistic Regression referenciado anteriormente, o número de parâmetros é independente do tamanho do dataset, variando apenas de acordo com o número de variáveis utilizadas para explicar a dinâmica de interesse.

Modelos não paramétricos são considerados mais flexíveis, no sentido de que são capazes de lidar com fronteiras de decisão mais complexas, embora possam ser computacionalmente custosos ao modelarem conjuntos de dados muito grandes.

Outra característica de *K-Nearest Neighbors* é o fato de são conhecidos por serem *lazy learners*. A diferença principal para algoritmos considerados como *eager learners* reside no momento em que o algoritmo utiliza os dados de treinamento para realizar suas previsões. No primeiro caso, algoritmos pertencentes a esta classe utilizam os dados apenas quando chamados a realizar previsão. Em face desta característica, tal algoritmo mostra-se usualmente mais lento, notadamente nos casos em que estamos lidando com *datasets* muito grandes, embora existam técnicas que visem a redução da complexidade de se comparar uma instância com todos os possíveis vizinhos.

Modelos *lazy learners* são mais adaptáveis a situações em que o relacionamento entre as variáveis descritivas e a variável alvo variam ao longo do tempo. No caso em apreço, *eager learners* podem se desatualizar, necessitando de novos treinamentos, o que, por sua vez, também é uma tarefa custosa.

No que concerne ao parâmetro *k*, a escolha do valor ideal não é trivial. Valores de k muito pequenos tornam o modelo muito sensível à existência de outliers e mais suscetível à possibilidade de ocorrência de overfitting. Além disso, apresentam uma fronteira de decisão com aspecto mais recortado. Já o caso em que os modelos possuem valores para k muito altos apresentam fronteiras de decisão mais suaves, enquanto assumem o risco de conseguir capturar o verdadeiro padrão existente nos dados, aumentando a possibilidade de ocorrência de underfitting. A figura 5 demonstra o comportamento citado para um caso simplificado com a utilização de apenas 2 variáveis descritivas.

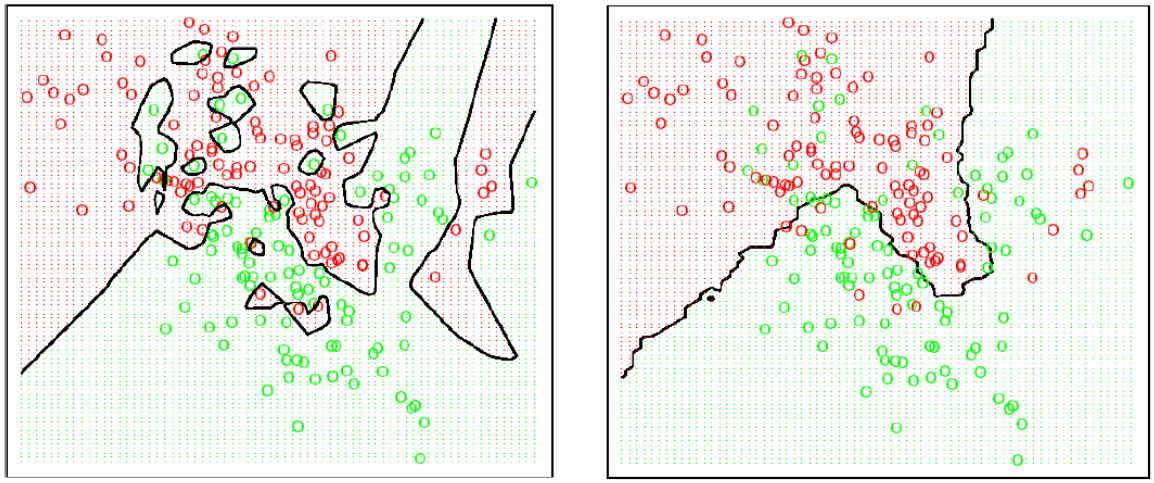

Figura 5. Fronteira de decisão para casos com k=1 e k=15

Vale ressaltar, ainda, 2 importantes características relativamente ao modelo K-Nearest Neighbors: a importância da padronização dos valores das variáveis descritivas e a necessidade de utilização de métricas adequadas para aferir a distância entre 2 observações de acordo com o tipo de variável a ser utilizada.

Relativamente à primeira questão, a padronização dos valores impede que qualquer variável que assuma valores em intervalo muito superior ao das demais acabe por dominar o cálculo do valor final da distância, evitando também que o modelo apresente viés relacionado a esta variável.

No que tange à necessidade de utilização de métricas apropriadas ao tipo da variável, há uma infinidade de opções para tratarmos variáveis binárias (que assumam valor 0 ou 1, dependendo da presença ou não daquela característica), que produzem resultados mais intuitivos para tais situações,

como, por exemplo, Jaccard, Russel-Rao ou Sokal-Michener. Cabe, por óbvio, recordar que o processo de experimentação é sempre necessário para a determinação da medida de similaridade mais efetiva no caso concreto.

A implementação deste algoritmo em Python utilizada no trabalho é dada pela função sklearn.neighbors.KNeighborsClassifier(). Por não terem sido abordados na descrição do algoritmo realizada ao longo desta seção, falaremos brevemente da utilidade de 2 hiperparâmetros: *weights* e *algorithm*.

Weights permite-nos definir pesos diferentes aos vizinhos na votação para determinação da classe prevista para uma determinada instância, ou seja, vizinhos mais próximos exerceriam uma influência maior que vizinhos mais distantes.

Algorithm garante-nos alternativas mais eficientes para lidar com datasets muito grandes, mitigando a necessidade de calcular a distância para todas as instâncias de treinamento no momento de definição dos vizinhos mais próximos.

3.3.3. Decision Trees

Decision Trees (ou Árvores de Decisão) são um dos mais importantes e versáteis algoritmos utilizados em projetos que envolvem modelagem via Machine Learning, seja em problemas de regressão, seja em problemas de classificação, como é o nosso caso.

Uma de suas grandes vantagens está relacionada ao fato de serem de fácil entendimento, motivo pelo qual são considerados modelos white box – diferentemente, por exemplo, de uma rede neural, cujos resultados podem ser difíceis de interpretar. Treinar uma Decision Tree significa aprender a sequência de questões do tipo verdadeiro/falso que nos leva à resposta correta mais rapidamente (Müller & Guido, 2016). Cada questão pertencente à árvore é conhecida por teste. Um exemplo de uma Decision Tree bem simples pode ser ilustrado na figura 6.

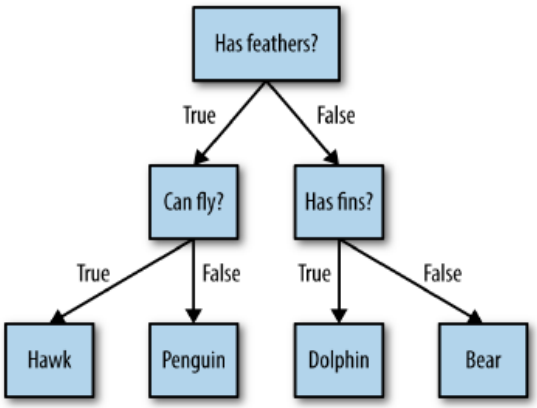

Figura 6. Exemplo de uma Decision Tree (Fonte: Müller & Guido, 2016)

Cada retângulo dentro da estrutura geral da árvore é conhecido por nó. Podemos notar, ainda, a existência de 2 tipos de nós diferentes: aqueles internos (que são compostos por questões), assim como os nós terminais da árvore, que são denominados folhas. Ao construirmos uma árvore de

decisão, devemos ter em mente que o objetivo principal é ser capaz de discriminar entre as classes para obter folhas tão puras quanto possível, o que significa que aquele nó é composto somente por instâncias de uma única classe. Trataremos sobre medidas de impureza mais adiante nesta seção.

No exemplo ilustrado, as variáveis utilizadas são do tipo binárias, porém estes algoritmos também podem ser treinados com variáveis de entrada do tipo contínuas. A forma mais simples de lidar com variáveis deste tipo é por meio da definição de um valor de corte contido no intervalo de valores que a variável assume. Definido um corte "c", o teste é definido da forma: "*O valor da variável é maior que c?*"

O algoritmo, então, busca dentre todos os possíveis testes aquele que produz o maior ganho de informação. Os 2 critérios mais utilizados para quantificação do ganho de informação são a entropia e o índice de gini. Para um melhor entendimento destes conceitos, iniciaremos na equação 8 com a definição de entropia para um conjunto de dados $\mathcal{D}$:

$$H(t, \mathcal{D}) = -\sum_{l \in levels(t)} (P(t = l) \text{ x } log_2(P(t = l))) \quad (8)$$

Onde:

$levels(t)$ representa as classes da variável dependente

$P(t = l)$ representa a probabilidade de selecionarmos uma instância pertencente à classe l.

A equação 9 define a entropia restante após particionarmos o conjunto de dados $\mathcal{D}$, considerando uma variável descritiva específica d.

$$rem(d, \mathcal{D}) = -\sum_{l \in levels(d)} \left(\frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|} \text{ x } H(t, \mathcal{D}_{d=l}) \right) \quad (9)$$

Onde:

$H(t, \mathcal{D}_{d=l})$ representa a entropia de uma partição do conjunto de dados que contém todas as instâncias com valor l para a variável independente d

$|\mathcal{D}_{d=l}|$ representa o tamanho da partição

$|\mathcal{D}|$ representa o tamanho do conjunto de dados

Definidos os conceitos anteriores, podemos determinar formalmente o ganho de informação ao dividirmos o conjunto de dados utilizando a variável descritiva d por meio da equação 10 abaixo representada.

$$IG(d, \mathcal{D}) = H(t, \mathcal{D}) - rem(d, \mathcal{D}) \quad (10)$$

Podemos definir, ainda, o índice de Gini, conforme a equação 11. O cálculo do ganho de informação, neste caso, se dá de forma análoga, apenas precisando de se promoverem as alterações pertinentes, sempre substituindo $H(t, \mathcal{D})$ por $Gini(t, \mathcal{D})$.

$$Gini(t, \mathcal{D}) = 1 - \sum_{l \in levels(t)} P(t = l)^2 \quad (11)$$

Por óbvio, a tarefa de descobrir o melhor conjunto de testes quando possuímos variáveis contínuas – o que implica possibilidades ilimitadas – pode ser realmente desafiadora. Há, no entanto, técnicas para evitar a realização de infinitos cálculos. Resumidamente, iniciamos pela ordenação das instâncias do conjunto de dados com base no valor da variável contínua. A partir da ordenação, instâncias adjacentes que possuem diferentes valores para a variável dependente são consideradas candidatas a ponto de corte. Os candidatos a ponto de corte recebem o valor médio entre as instâncias adjacentes. Assim, calculamos o ganho de informação para cada uma desses candidatos, selecionando como ponto de corte ótimo aquele que produz o maior ganho de informação.

Além da facilidade de interpretação dos motivos que embasaram uma decisão tomada por meio deste algoritmo, como já comentamos anteriormente, um dos grandes benefícios relacionados à utilização de árvores de decisão é que os algoritmos são completamente invariantes à escala dos dados. Isso ocorre, pois cada variável é processada de forma independente e o ponto de corte não depende da escala. Assim, não são requeridas quaisquer espécies de normalização de dados. Em particular, Decision Trees funcionam bem quando temos variáveis que variam em escalas completamente diferentes, ou quando possuímos um mix de variáveis contínuas e binárias (Müller & Guido, 2016).

Ademais, árvores de decisão podem ser utilizadas para seleção de variáveis, tendo em vista que as características mais relevantes acabam por ocupar a parte superior da árvore. Em especial, a implementação em Python nos oferece um atributo chamado *feature_importances_*, que se situa entre 0 e 1 para cada variável e soma no total o valor 1, ou seja, nos permite avaliar a importância de cada variável na tomada de decisão após a construção da árvore. No entanto, o fato de uma variável apresentar um valor baixo, não necessariamente quer dizer que esta é uma variável não informativa, podendo ocorrer pelo fato de que outra variável já agrega o mesmo tipo de informação.

Além disso, já abordamos na seção 3.3.2. a diferença entre modelos paramétricos e não-paramétricos. Decision Trees são modelos não-paramétricos, o que não quer dizer que não possuem nenhum parâmetro, mas sim que o número de parâmetros não é determinado *a priori*, permitindo ao modelo que se ajuste livremente aos dados.

Essa liberdade para se ajustar aos dados, quando não acompanhada de nenhuma restrição, garante uma ótima flexibilidade, mas pode resultar em overfitting. Apenas para contextualização, trazemos uma definição mais formal do conceito, extraída de (Mitchell, 1997):

> *"Dado um espaço de hipóteses $\mathcal{H}$, diz-se que uma hipótese $h \in \mathcal{H}$ produz um overfitting nos dados de treinamento se existe alguma hipótese alternativa $h' \in \mathcal{H}$, tal que h produz um erro menor do que h' sobre as instâncias de treinamento, mas h' produz um erro menor do que h quando consideramos a distribuição geral das instâncias."*

Em outras palavras, o algoritmo é capaz de reproduzir de forma excelente (ou perfeita) o comportamento das instâncias de treinamento, mas apresenta performance inferior (ou muito pobre) ao se deparar com novas instâncias para realizar previsões, generalizando mal o conjunto de dados de forma geral.

Neste sentido, de forma a mitigar possíveis problemas relacionados a overfitting, aplicamos o processo de regularização dos hiperparâmetros, que é equivalente a restringir os graus de liberdade durante o

treinamento. Essas restrições podem ser impostas de diversas formas, como, por exemplo, restringindo a profundidade máxima que uma árvore pode atingir.

Essencialmente, existem 2 tipos de abordagens para evitar overfitting: pre-pruning e post-pruning. Na primeira, interrompemos o crescimento da árvore antes da finalização do seu processo de treinamento, privilegiando o poder de generalização do modelo em detrimento de sua consistência com os dados de treinamento. Na abordagem alternativa, o processo de treinamento da árvore de decisão ótima é concluído e, posteriormente, cada ramo da árvore é examinado e, baseado em algum critério especificado, é podado, caso seja considerado provável causador de overfitting.

Outra possível desvantagem no que concerne à utilização de árvores de decisão como classificadores é o fato de que são instáveis, pois pequenas variações nos dados podem acarretar grandes mudanças na estrutura da árvore de decisão ótima. Além disso, as fronteiras de decisão resultantes do treinamento de uma Decision Tree tradicional são ortogonais. Tal fato nem sempre é desejável, tendo em vista a existência de dados cuja fronteira de decisão real não é paralela aos eixos cartesianos (há, inclusive, modelos de Decision Trees oblíquas, cuja grande desvantagem é que os algoritmos são mais custosos computacionalmente). A utilização de Análise de Componentes Principais (PCA) é uma possível forma de se contornar esse problema, proporcionando uma reorientação dos dados de treinamento.

A implementação deste algoritmo em Python utilizada no trabalho é dada pela função sklearn.tree.DecisionTreeClassifier(). Há uma vasta gama de hiperparâmetros que podem ser calibrados durante o processo de treinamento.

A maior parte deste hiperparâmetros se relaciona ao processo de regularização da árvore de decisão, que é propensa, conforme vimos, a memorizar as instâncias de treinamento e performar mal quando em contato com novos dados.

Assim, temos a opção de utilizar, dentre outras opções, *max_depth*, *min_samples_split*, *min_samples_leaf* e *max_leaf_node.* Ao ajustarmos *max_depth*, estamos estabelecendo uma profundidade máxima para a árvore. Com *min_samples_split*, um número mínimo de instâncias necessárias para que um nó interno seja dividido. Já *min_samples_leaf* estabelece o número mínimo de amostras requeridas para que um nó seja terminal. Por fim, *max_leaf_node* determina o número máximo de *leaves* que uma árvore pode possuir.

Embora sejam hiperparâmetros diferentes, todos guardam relação com a abordagem pre-pruning supramencionada. O aumento dos hiperparâmetros *min_ ** e a diminuição dos hiperparâmetros *max_ ** colaboram para a regularização do modelo e a mitigação do overfitting. Já o hiperparâmetro *ccp_alpha* nos oferece uma opção para implementar a abordagem post-prunning.

Temos, ainda, em linha com o explanado acerca dos critérios para aferição da pureza de um nó, a possibilidade de selecionarmos os critérios de entropia ou o índice de gini através do hiperparâmetro *criterion*.

O último hiperparâmetro que julgamos relevante, tendo em conta a própria característica do dataset que estamos utilizando neste trabalho, é *class_weight*, idêntico ao existente para a função KNeighborsClassifier. Assim como ocorrer em outros algoritmos, o algoritmo para treinamento de árvores de decisão cria árvores viesadas diante da existência de uma classe dominante.

Mais à frente, abordaremos modelos mais complexos, que utilizam um conjunto de diferentes Decision Trees, tentando capturar as melhores características de cada uma das árvores. A este grupo de modelos, dá-se o nome de *Ensemble*, dos quais podemos citar como exemplo as Random Forests.

3.3.4. Naive Bayes

Naive Bayes é uma família de classificadores que se baseia fortemente no Teorema de Bayes, formalizado pela equação 12.

$$P(A|B) = \frac{P(B|A) \text{ x } P(A)}{P(B)} \tag{12}$$

Especificamente no que concerne ao classificador Naive Bayes, para um conjunto de classes representado pela variável y e um conjunto de n variáveis descritivas $x_1, x_2, \ldots, x_n$, a equação assume a forma apresentada abaixo.

$$P(y|x_1, x_2, \ldots, x_n) = \frac{P(x_1, x_2, \ldots, x_n|y) \text{ x } P(y)}{P(x_1, x_2, \ldots, x_n)} \tag{13}$$

Onde:

$P(y|x_1, x_2, \ldots, x_n)$ é conhecida como *posteriori* e representa a probabilidade de que uma observação pertença a uma determinada classe y dados os valores da observação para o conjunto de variáveis $x_1, x_2, \ldots, x_n$

$P(x_1, x_2, \ldots, x_n|y)$ é conhecida como *verossimilhança* e representa a probabilidade de ocorrência dos valores $x_1, x_2, \ldots, x_n$ para as n variáveis dado uma classe particular y

$P(y)$ é conhecida com *priori* e representa a probabilidade inicial de ocorrência da classe y antes de olharmos para os dados

$P(x_1, x_2, \ldots, x_n)$ é conhecida por probabilidade marginal

No processo de otimização dos parâmetros, a classe $\hat{y}$ prevista é aquela que maximiza $P(y|x_1, x_2, \ldots, x_n)$. Matematicamente, temos que:

$$\hat{y} = \underset{y \in C}{\text{argmax}} P(y|x_1, x_2, \ldots, x_n) = \underset{y \in C}{\text{argmax}} \frac{P(x_1, x_2, \ldots, x_n|y) \text{ x } P(y)}{P(x_1, x_2, \ldots, x_n)} \tag{14}$$

Tendo em vista que o denominador $P(x_1, x_2, \ldots, x_n)$ não depende de y, temos que:

$$\hat{y} = \underset{y \in C}{\text{argmax}} P(x_1, x_2, \ldots, x_n|y) \text{ x } P(y) \tag{15}$$

Utilizando a regra da cadeia para probabilidades condicionais:

$$P(x_1, x_2, \ldots, x_n|y) = P(x_1|y) \text{ x } P(x_2|x_1, y) \text{ x } P(x_3|x_1, x_2, y) \text{ x } \ldots \text{ x } P(x_n|x_1, x_2, \ldots, x_{n-1}, y) \tag{16}$$

A determinação de todas as probabilidades condicionais existentes na equação 16 é difícil e sua complexidade aumenta conforme o número de variáveis descritivas também aumenta. Assim, tornar-se-ia necessário um aumento exponencial no tamanho do dataset para assegurar que existem instâncias suficientes que permitam a caracterização de cada uma das distribuições de probabilidade condicionais. Ocorre, neste caso, uma vertente da maldição da dimensionalidade.

Uma forma de mitigar o problema apresentado é assumir a hipótese de independência condicional, representada na equação 17.

$$P(x_1, x_2, \ldots, x_n|y) = P(x_1|y) \text{ x } P(x_2|y) \text{ x } \ldots \text{ x } P(\ x_n|y) \tag{17}$$

Trata-se de uma suposição difícil de garantir – na realidade, na grande maioria dos casos, tal hipótese está incorreta -, mas que simplifica enormemente os cálculos. Por isso, a literatura a denomina como uma suposição ingênua, o que justifica o nome do modelo.

Combinando as equações 15 e 17, resulta-se:

$$\hat{y} = \underset{y \in C}{\operatorname{argmax}} P(x_1|y) \text{ x } P(x_2|y) \text{ x } \ldots \text{ x } P(\ x_n|y) \text{ x } P(y) \tag{18}$$

O resultado desta abordagem é um algoritmo extremamente rápido e eficiente, mas que acaba por perder em poder de generalização. Ainda assim, produz bons resultados em diversos domínios de aplicação, como, por exemplo, classificação de documentos.

Como resultado, um modelo Naive Bayes é muitas vezes um bom modelo de previsão a ser usado para definir um baseline para a métrica de avaliação do problema (accuracy, precision, recall, etc.). Além disso, funciona muito bem mesmo para uma pequena quantidade de dados.

Dentre as opções de modelo Naive Bayes implementados em Python, na biblioteca *scikit-learn*, utilizaremos o algoritmo *GaussianNB*, que recebe este nome pois a distribuição condicional das variáveis descritivas dado que pertence a uma classe particular y é modelada por meio de distribuição normal, conforme exposto na seguinte equação.

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right) \tag{19}$$

Esta versão do algoritmo é a mais adequada para tratar do nosso problema, tendo em vista que as variáveis que utilizamos são majoritariamente contínuas. Os 2 únicos hiperparâmetros passíveis de serem calibrados são *priors* e var*_smoothing*.

No primeiro caso, deve ser usado quando há um palpite inicial a respeito da distribuição entre as classes. Caso não adicionemos nenhum valor, o algoritmo se encarrega de ajustá-lo baseado na distribuição do próprio conjunto de dados de treinamento.

3.3.5. Ensemble Learning

Ensemble Learning é uma técnica muito utilizada em Machine Learning que se baseia na utilização de não um, mas um grupo de algoritmos de forma a alcançar um resultado final mais acurado. A ideia por

trás desta técnica é a de que um comitê de especialistas trabalhando conjuntamente em um problema é mais provável de resolvê-lo com sucesso do que apenas um especialista trabalhando sozinho (Kelleher et al., 2015).

Cada algoritmo que compõe um *ensemble model* deve ser treinado para realizar suas previsões de forma independente dos demais componentes do grupo e, então, baseado em algum critério, os resultados são agregados para produzir uma previsão final única.

Ensemble models podem produzir resultados muito acurados, ainda que, individualmente, os algoritmos do grupo atinjam performance ligeiramente superiores a um modelo que preveja os resultados de forma aleatória.

Há diferentes abordagens para combinar modelos resultando na criação de um *ensemble model*, dentre as quais destacamos voting, bagging, pasting, boosting e stacking.

Voting

É uma das mais simples abordagens para criação de um ensemble. Sejam C1, C2, ..., Cn, classificadores que foram treinados em um conjunto de dados D. Um classificador do tipo voting é capaz de gerar um resultado melhor do que os resultados individuais por meio da agregação das previsões de cada classificador.

Caso a agregação se dê por maioria de votos, denominamos por classificador hard voting. Ou seja, a classe que recebeu mais votos individualmente passa a ser a classe prevista pelo modelo final. Conforme explanamos anteriormente, ainda que os classificadores C1, C2, ..., Cn sejam considerados weak learners, o classificador final V pode ser um strong learner (capaz de atingir alta acurácia). A condição para tal é a existência de um número suficiente grande de classificadores e que estes produzam a diversidade necessária. Este resultado é decorrente da aplicação da lei dos grandes números.

Uma observação importante acerca destas técnicas é que, quanto maior a independência entre os modelos, implicando erros não correlacionados, maior a acurácia atingida. No entanto, dado que os modelos são treinados no mesmo conjunto de dados, trata-se de uma hipótese claramente difícil de atingir. Ou seja, é muito provável que eles cometam o mesmo tipo de erro, resultando em maioria de votos para a mesma classe incorreta e, consequentemente, reduzindo a possível eficácia teórica do ensemble. Assim, uma sugestão na tentativa de aumentar a independência entre os modelos é treiná-los em algoritmos os mais diferentes possíveis.

A implementação em Python para construção de um classificador do tipo voting é dada por meio da biblioteca sklearn, que, por sua vez, possui a função VotingClassifier(). O hiperparâmetro voting permite 2 opções: hard e soft. A opção hard realiza a votação da maneira explicada anteriormente: maioria de votos. A opção soft o faz com base na classe predita com a maior probabilidade dentre todos os classificadores. Para isto, depende que o algoritmo implementado contenha o método predict_proba().

Bagging

A abordagem bagging (ou bootstrap aggregating) é um pouco diferente da anterior: nela, utilizamos o mesmo algoritmo em cada classificador individual C1, C2, ..., Cn, porém treinado em subconjuntos diversos do dataset. Uma característica importante deste modelo é que cada subconjunto deve necessariamente possuir o mesmo tamanho do dataset original e o processo de amostragem é feito com reposição.

Tendo em conta que cada subconjunto é diferente dos demais, os modelos resultantes treinados em diferentes amostras também serão diferentes, garantindo de certa forma a diversidade necessária. Um dos modelos mais conhecidos baseado em bagging é o Random Forest, que possui como algoritmo base uma Decision Tree. Esta escolha é particularmente acertada haja vista sua grande sensibilidade a mudanças no conjunto de dados (ou seja, as árvores de decisão resultantes podem não apresentar as mesmas variáveis selecionadas para dividir o dataset na raiz ou nos nós superiores da árvore).

Outra possibilidade no processo de criação de um ensemble por meio de bagging é a realização de amostragem das variáveis descritivas, utilizando apenas um subconjunto das variáveis disponíveis no dataset completo. Trata-se, aqui, de mais um recurso para gerar diversidade entre os modelos treinados, antes de que seja realizada a agregação.

Com o objetivo de chegar à classe prevista final, o classificador realiza a agregação por maioria de votos. A figura 7 apresenta um resumo visual do processo de criação de um classificador ensemble usando bagging.

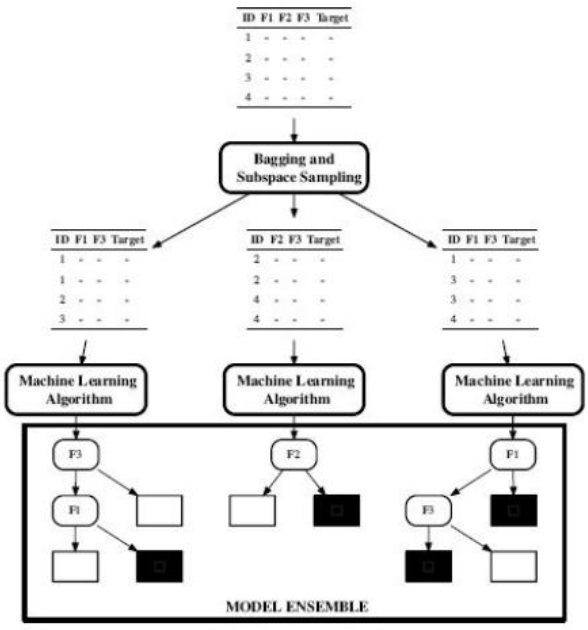

Figura 7. Processo de criação de um classificador utilizando a abordagem bagging (Fonte: Kelleher et al., 2015)

Uma abordagem muito semelhante a esta é conhecida por pasting. A diferença principal entre bagging e pasting reside no fato de que, neste último, o processo de amostragem é feito sem reposição.

No que tange ao custo computacional de se treinar uma modelo via bagging, ressalta-se que os preditores podem ser treinados em paralelo, em diferentes núcleos de uma mesma CPU ou mesmo

em diferentes servidores, o que torna uma opção interessante sob o ponto de vista da escalabilidade quando comparados a outras abordagens de ensemble que veremos mais adiante.

Por fim, embora um modelo Random Forest seja um exemplo de ensemble através do conceito de bagging, Python nos oferece o recurso de treinarmos o nosso próprio modelo com a utilização da função BaggingClassifier(), também disponível na biblioteca guarda-chuva sklearn.

Boosting

A técnica conhecida por boosting também é muito utilizada em Machine Learning e possui algumas características que a diferem da anterior. Inicialmente, é válido destacar que os modelos componentes do ensemble são treinados de maneira sequencial – e não em paralelo –, associando a cada iteração pesos diferentes às instâncias que foram classificadas incorretamente na etapa anterior, a fim de tentar corrigir os erros cometidos pelos antecessores.

A ideia geral por trás desta técnica é que, no fim, obteremos um grupo de especialistas que se complementam. A previsão final é feita através de uma votação ponderada, em que os classificadores que alcançaram uma acurácia maior, receberão pesos maiores no processo de votação.

Podemos descrever de forma bastante simplificada o procedimento de criação de um modelo via boosting através dos seguintes passos:

1. Recebem-se de entrada o conjunto de dados, o algoritmo que servirá de base para o treinamento e o número de classificadores que serão treinados.
2. Inicializam-se de forma igualitária os pesos que serão utilizados para extrair a amostra do dataset inicial, na qual o número de vezes que uma instância aparecerá é proporcional ao seu peso.
3. Treina-se um weak learner nesta amostra.
4. Calcula-se o erro do classificador.
5. Ajustam-se os pesos, aumentando-os nos casos de instâncias que foram classificadas incorretamente e reduzindo-os quando corretamente classificadas.
6. Repete-se o processo até atingir o número de classificadores definidos inicialmente.
7. Combinam-se os resultados de todos os classificadores.

Como desvantagem no uso desta metodologia em contraponto à utilização de bagging, podemos citar que o fato de serem treinados de forma sequencial e iterativa, não admitido os cálculos em paralelo, aumenta o tempo de processamento e reduz a escalabilidade desta opção. Além disto, pode-se mostrar mais sensível à existência de outliers, tendo em vista que cada classificador se dedica a corrigir problemas de seus antecessores, o que pode levar a uma atenção demasiada a este tipo de observações.

Em Python, o pacote sklearn nos oferece 2 implementações populares de algoritmos baseado na técnica de boosting: AdaBoost e Gradient Boosting. Nas seções a seguir, veremos maiores detalhes sobre tais algoritmos, como o algoritmo base, o processo de atualização dos pesos e cálculo do erro do classificador e, por fim, como se dá a combinação dos classificadores sequenciais para produção de um resultado final.

Stacking

O último método de ensemble que abordaremos nesta tese é conhecido por stacking, que se baseia fundamentalmente em treinar um novo modelo, em vez de utilizar critérios como hard voting, para realizar a agregação das previsões dos classificadores de primeiro nível. Ao algoritmo de segundo nível dá-se o nome de meta-learner, enquanto os preditores de primeiro nível são chamados de base-learners.

As previsões dos algoritmos base são utilizadas para criar um novo dataset, que será treinado pelo meta-learner para, aí sim, produzir as previsões finais. Uma importante ressalva é que, se os mesmos dados que utilizamos para treinar o primeiro nível forem utilizados para gerar o novo dataset, as chances de overfitting se tornam muito aumentadas. Portanto, são consideradas boas práticas que as instâncias usadas para gerar o novo dataset sejam excluídas das instâncias de treinamento do primeiro nível.

Uma abordagem comumente utilizada é descrita a seguir. Primeiramente, o conjunto de treinamento é dividido em 2 partes, uma das quais será utilizada para treinar os algoritmos base. Treinados os preditores de primeiro nível, utilizamos a segunda parte para fazer as previsões que comporão o novo dataset. O número de variáveis descritivas deste novo conjunto de treinamento será igual ao número de classificadores treinados na primeira etapa e a variável resposta é mantida a mesma utilizada inicialmente. Então, treina-se o meta-learner (também conhecido por blender) no novo dataset. As figuras abaixo postas lado a lado representam as etapas de treinamento dos algoritmos base (à esquerda) e do blender (à direita), considerando a divisão do dataset em 2 partes com o intuito de evitar a ocorrência de overfitting.

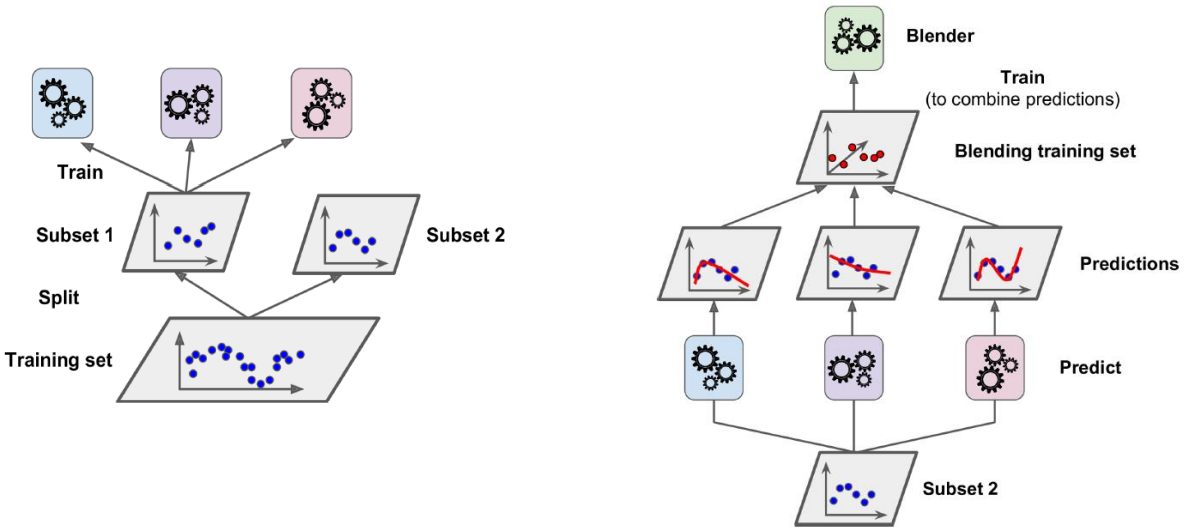

Figura 8. Treinamento dos algoritmos de primeiro nível (à esquerda) e do meta-learner (à direita) (Fonte: Géron, 2019)

A abordagem descrita pode ser estendida para prever mais de um blender, formando um nível intermediário. Neste caso, o dataset deveria idealmente ser dividido em 3 partes: a primeira para treinar o primeiro nível de classificadores; a segunda para criar o conjunto de treinamento (a partir das

previsões dos classificadores treinados na primeira etapa) que será usado pelo segundo nível; e, por fim, a terceira para gerar o conjunto de treinamento que será usado pelo terceiro nível (a partir das previsões dos classificadores treinados na segunda etapa).

A biblioteca mais utilizada em Python para projetos de Machine Learning, scikit learn (ou sklearn), não nos oferece uma função implementando stacking, mas é possível alternativamente o uso de outras bibliotecas, tais como brew ou ML-Ensemble (mlens).

3.3.6. AdaBoost

Anteriormente, pôde-se verificar na teoria uma abordagem para implementação de um ensemble na modalidade boosting. AdaBoost é, na prática, um dos mais conhecidos algoritmos que se utiliza desta técnica. Os passos definidos na seção 3.3.5. consistem apenas em uma descrição geral do procedimento, mas existem partes não especificadas que serão tratadas neste momento.

O algoritmo base geralmente utilizado quando treinamos um classificador AdaBoost é a já conhecida Decision Tree, embora o treinamento possa ser feito com algum outro modelo (preferencialmente algum weak learner).

Especificamente no que concerne a este modelo, os pesos iniciais definidos para cada instância são dados por $1/m$, onde m representa o número total de observações que compõem o conjunto de treinamento.

Após o treinamento de cada classificador, seu erro do classificador é calculado por meio da equação 20.

$$r_j = \frac{\sum^{m}_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}} \tag{20}$$

O numerador da equação acima representa a soma dos pesos de todas as instâncias que foram incorretamente classificadas. Após calculado, divide-se pela soma total do peso conferido a todas as instâncias.

Cada preditor passa, então, a possuir um peso associado α_j (não deve ser confundido com o peso associado a cada instância $w^{(i)}$), calculado da forma que se segue.

$$\alpha_j = \eta \text{ x } \log_e \frac{1-r_j}{r_j} \tag{21}$$

η é a taxa de aprendizagem e pode ser calibrada no processo de treinamento.

Assim, os pesos das instâncias que foram incorretamente classificadas podem ser atualizados para adquirirem uma importância maior, enquanto as demais permanecem idênticas. A equação 22 especifica a regra de atualização.

$$w^{(i)} \leftarrow w^{(i)} \text{ x } \exp(\alpha_j) \tag{22}$$

Repare, aqui, que a regra de atualização é consistente com o que desejamos do ponto de vista teórico: o peso α_j associado a cada preditor se torna maior quanto menor for a taxa de erro (podendo,

inclusive, ser negativo caso r_j assuma valores superiores a 0,5). Ou seja, caso o classificador possua uma acurácia muito grande, α_j terá um valor alto e as instâncias incorretamente previstas (instâncias de difícil previsão, pois, recorde-se, o classificador neste caso apresentou alto poder preditivo) adquirirão grande importância com pesos $w^{(i)}$ altos.

Por fim, os pesos $w^{(i)}$ são todos divididos pelo somatório dos pesos, em um processo conhecido por normalização. Assim, treina-se um novo preditor com os pesos atualizados e o processo já descrito se repete até que o número de classificadores previamente definido seja treinado.

A classe final prevista pelo modelo é atingida por meio da equação 23.

$$\hat{y}(\mathrm{x}) = \underset{k}{\operatorname{argmax}} \sum_{\substack{j=1 \\ \hat{y}_j(\mathrm{x})=k}}^{N} \alpha_j \tag{23}$$

De forma simplificada, para uma determinada instância, cada um dos N classificadores realiza uma previsão. Supondo um problema de classificação binária, destacam-se todos os classificadores que previram a classe 0 e realiza-se a soma dos pesos α_j associados. Analogamente, reproduzimos o mesmo processo para todos os classificadores que previram a classe 1. A classe que produz o maior somatório ponderado é considerada a classe final prevista.

A implementação deste algoritmo em Python está disponível no pacote sklearn, por meio da função AdaBoostClassifier().

3.3.7. Gradient Boosting

Gradient Boosting é uma outra opção de modelo que faz uso da técnica de boosting anteriormente referida. Deste modo, o princípio básico é o mesmo: treinar diversos classificadores sequencialmente, com o objetivo de que o sucessor seja capaz de corrigir os erros cometidos pelo antecessor.

A diferença principal do Gradient Boosting para o AdaBoost é a forma utilizada para o treinamento de cada algoritmo: em vez de aumentar os pesos direcionados às instâncias de mais difícil previsão, esta abordagem busca treinar o próximo algoritmo nos resíduos do preditor anterior.

Embora a explicação detalhada fuja ao escopo desta tese, buscaremos descrever de forma sucinta a ideia do algoritmo de Friedman (2001) nos passos a seguir:

1. Inicializa-se a função score F com valor 0 para cada uma das classes.
2. Os scores são utilizados para calcular as probabilidades de cada classe por meio da função $P_J(\mathrm{x}) = \frac{e^{F_J(\mathrm{x})}}{\sum_{c=A}^{Z} e^{F_c(\mathrm{x})}}$.
3. O valor da variável resposta para cada instância será utilizado para construir a distribuição de probabilidade Y.
4. A diferença entre a distribuição de probabilidade real Y e a prevista P é utilizada para calcular os resíduos.
5. Treina-se uma Decision Tree h utilizando os resíduos como target do modelo.
6. Atualizam-se os scores F de forma tal que $F := F + \rho \, \mathrm{x} \, h$, onde ρ é taxa de aprendizado (hiperparâmetro do modelo).

7. O processo segue de forma iterativa até que um número especificado inicialmente de iterações seja treinado.

Uma implementação deste tipo de algoritmo é fornecida pela biblioteca sklearn através da função GradientBoostingClassifier(). Por fazer uso de árvores de decisão, muitos dos hiperparâmetros já comentados na seção 3.3.3. estão também disponíveis aqui para calibragem.

Adicionalmente, cabe-nos tecer comentários mais específicos acerca dos seguintes hiperparâmetros: learning_rate e n_estimators.

O primeiro destes trata exatamente do parâmetro ρ comentado na explicação dos passos do algoritmo. O objetivo aqui é controlar a força com que cada árvore adicional corrige os erros cometidos pela predecessora, sendo que valores altos para a taxa de aprendizagem geram modelos mais complexos. O segundo parâmetro em comento – n_estimators – visa a adição de mais árvores ao ensemble, o que também resulta em aumento de complexidade.

Ambos os parâmetros estão intimamente relacionados, tendo em vista que um valor menor para a taxa de aprendizado acarreta a necessidade de um maior número de árvores para compensar o modelo na complexidade necessária. O aumento indefinido do número de estimadores pode levar o modelo a overfitting.

Em suma, gradiente boosting são modelos muito poderosos e extensamente usados em projetos que envolvam previsão. Seus principais pontos fracos são a necessidade de cuidadosa calibragem de seus parâmetros e o tempo potencialmente necessário para treinamento do modelo.

3.3.8. Support Vector Machines (SVM)

Support Vector Machine é uma ferramenta poderosa adequada para utilização em projetos de Machine Learning que envolvam problemas de classificação e regressão. Trata-se de um dos modelos mais utilizados e se ajusta bem a conjuntos de dados complexos, seja de tamanho pequeno ou médio. Há, ainda, uma característica interessante, que é a sua capacidade de performar bem na detecção de outliers.

A ideia principal em que se baseia esse algoritmo é encontrar o hiperplano que maximiza as margens entre as classes de um conjunto de treinamento. Para melhor entendimento do funcionamento, é necessária a compreensão dos termos subjacentes. Assim, hiperplano é um subespaço de dimensão n-1 contido em um espaço n-dimensional. Por exemplo, caso desejemos dividir um espaço bidimensional ($\mathbb{R}^2$), nós utilizaríamos uma reta (hiperplano unidimensional). Já no caso do espaço $\mathbb{R}^3$, um plano bidimensional seria utilizado para dividi-lo.

Em outros termos, podemos pensar um classificador SVM como aquele que ajusta a “rua” mais extensa possível entre 2 classes linearmente separáveis. A figura 9 permite uma visualização do conceito por trás deste tipo de classificador.

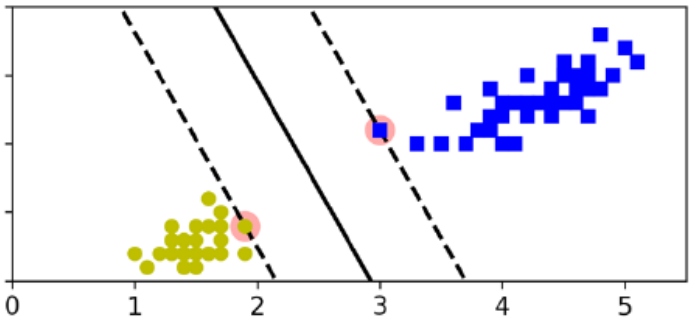

Figura 9. Ajuste gerado por um classificador SVM para um conjunto de treinamento linearmente separável (Géron, 2019)

Olhando novamente para a figura, podemos imaginar uma vasta gama de retas que seriam capazes de dividir as classes em grupos distintos. Deste modo, o que um SVM faz é trazer-nos uma fronteira de decisão, onde as retas paralelas pontilhadas representadas na figura criam uma espécie de rua e, de todas as ruas possíveis, o classificador gera a mais extensa delas. O nome dado a este processo é *large margin*. Já as instâncias de treinamento situadas nas extremidades (circuladas em rosa) são chamadas *support vectors*, que dão sentido ao nome conferido ao algoritmo.

O caso apresentado possui uma situação muito peculiar, raramente encontrada em algum dataset usado em projetos reais: as classes são linearmente separáveis. A imposição de uma restrição estrita de que todas as instâncias de uma classe devem estar "fora da rua" e de apenas um lado (hard margin classification), sem ocorrência de sobreposição, só é passível de ser atendida nestes casos. Ademais, este tipo de restrição torna os resultados do modelo extremamente sensível à presença de outliers, podendo implicar um maior risco de haver perda do poder de generalização.

Uma forma de escapar destas questões é suaviza as restrições, permitindo a existência de violações, que nada mais seriam do que erros de classificação. A este processo, damos o nome de soft margin classification.

A implementação em Python deste algoritmo permite especificar uma série de hiperparâmetros. Um deles, que se encontra relacionado com a flexibilização das restrições, é o parâmetro C, cujo valor padrão é 1. Valores pequenos para este parâmetro fazem com que o algoritmo admita um número maior de violações, enquanto valores mais maiores penalizam fortemente o classificador, o que o faz evitar erros de classificação. Ou seja, caso um SVM esteja com problemas de overfitting, deve-se tentar regularizar o modelo reduzindo os valores de C.

Alternativamente, para conjuntos de dados em que as classes não conseguem ser divididas por uma fronteira de decisão definida por um hiperplano linear, há a possibilidade de implementarmos diferentes kernels objetivando lidar com este tipo de situação.

De forma geral, conforme definido em Albon (2018), um classificador support vector pode ser representado pela seguinte equação:

$$f(\mathrm{x}) = \beta_0 + \sum_{i \epsilon S} \alpha_i K(x_i, x_{i'}) \tag{24}$$

Onde:

β_0: viés

S: conjunto de todos as observações consideradas support vectors

α: parâmetros do modelo

$(x_i, x_{i'})$: par de observações consideradas support vectors

K: função kernel, que compara a similaridade entre x_i e $x_{i'}$

A função K será responsável por definir o formato do hiperplano (fronteira de decisão) que segregará as classes existentes. Diferentes kernels são capazes de gerar diferentes fronteiras, inclusive com aspecto não-linear.

Dentre as diversas possibilidades de implementação de kernels, as equações 25, 26 e 27 apresentam, respectivamente, os denominados kernel linear, polinomial e RBF (Radial Basis Function).

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij}\, x_{i'j} \tag{25}$$

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij}\, x_{i'j}\right)^d \tag{26}$$

$$K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} x_{i'j})^2\right) \tag{27}$$

O limite superior presente em todos os somatórios, p, representa o número de variáveis descritivas existentes no conjunto de dados. O expoente d presente na equação 26 é chamado grau da função kernel polinomial. Além disso, temos ainda o hiperparâmetro γ (gamma) na equação 27, que atua como mais um parâmetro de regularização do modelo, ou seja, caso o modelo apresente sinais de overfitting, devemos reduzir o valor de gamma para aumentar o poder de generalização.

A figura 10 nos mostra um exemplo de uma fronteira decisão criada a partir de um kernel do tipo RBF, deixando clara a diferença de resultado para um kernel linear, que seria incapaz de separar as 2 classes desta mesma maneira.

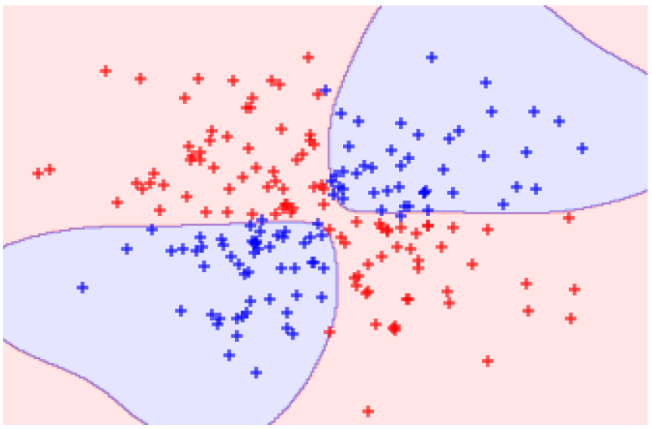

Figura 10. Hiperplano criado a partir de um kernel RBF (Fonte: Albon, 2018)

3.3.9. TPOT Classifier

Tree-based Pipeline Optimization Tool (TPOT) utiliza programação genética para desenvolver e otimizar uma série de transformações nos dados e modelos de machine learning com o objetivo de maximizar a acurácia em problemas de classificação para um dado conjunto de dados para aprendizagem supervisionada (Olson, 2016).

Como detalhado na seção 3.1., há diversas etapas que compõem o processo de modelagem via algoritmos de machine learning. A escolha do modelo que será testado como classificador é apenas um dos passos por que deve passar um cientista de dados, havendo diversos pipelines possíveis envolvendo a execução das etapas de exploração dos dados, criação de novas variáveis ou calibragem dos hiperparâmetros dos modelos testados.

Por este motivo, a literatura tem-se esmerado no sentido de desenvolver alternativas para a automatização de modelos de machine learning (AutoML), o que nos permite reduzir o tempo despendido em projetos, componente esta extremamente valiosa.

Algumas opções para automatizar pequenas partes deste pipeline já são bem conhecidas e incorporadas no cotidiano de um projeto. Por exemplo, Grid Search e Random Search são abordagens frequentes para calibragem automatizada de hiperparâmetros dos modelos. A primeira realiza uma busca exaustiva no espaço paramétrico, enquanto a segunda alternativa tem-se mostrado uma opção promissora e muitas vezes mais eficiente.

Entretanto, ao contrário das opções comentadas, TPOT pode construir pipelines complexos que englobam várias das etapas existentes no processo. A figura 11 mostra a representação de um projeto típico de modelagem utilizando algoritmos de aprendizagem supervisionada em problemas de classificação e especifica quais das etapas TPOT é capaz de tornar automáticas.

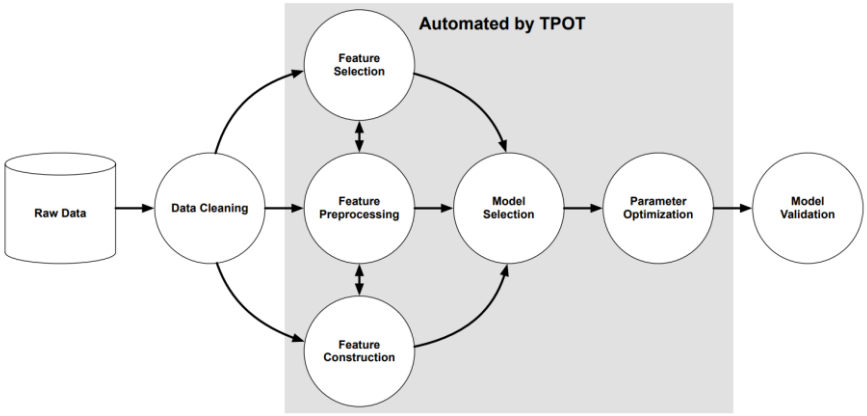

Figura 11. Etapas de um pipeline típico de machine learning automatizadas por TPOT (Olson, 2016)

Em sua versão original, o artigo que descreve a metodologia TPOT lista quatro tipos principais de operadores utilizados (todos eles fazem uso de funções previamente implementadas no *sklearn*): *preprocessors*, *decomposition*, *feature selection* e *models*.

A metodologia se baseia na combinação desses operadores para a formação dos pipelines em formato de árvores. Cada árvore é formada inicialmente por uma ou mais cópias do dataset (leaves da árvore), que são, então, inseridos como entrada em algum dos operadores (nós). Após, podemos realizar a combinação dos múltiplos conjuntos (já modificados), resultando em um novo e único dataset, que passará pelos processos de seleção de variáveis e aplicação dos modelos. A figura 12 mostra o exemplo de um possível pipeline.

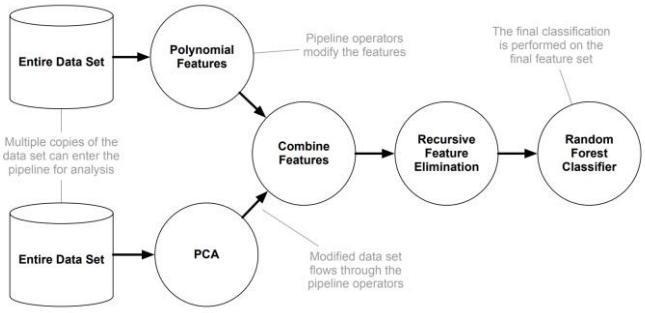

Figura 12. Exemplo de um pipeline gerado após a aplicação de TPOT (Olson, 2016)

Uma característica interessante implementada no algoritmo de otimização de TPOT é a utilização de 2 funções objetivo: uma para maximizar a acurácia final do problema e outra visando minimizar a complexidade geral do pipeline (o número de operadores total do pipeline).

A implementação em Python deste algoritmo se dá por meio da biblioteca tpot, que nos oferece a função TPOTClassifier. Dentre os principais parâmetros, podemos destacar os seguintes: *generations*, *population_size, mutation_rate, crossover_rate* e *scoring*.

O parâmetro chamado *generations* corresponde ao número de iterações que serão executadas no processo de otimização. Já o parâmetro *population_size* representa o número de indivíduos existentes na população da programação genética a cada iteração. Por sua vez, os parâmetros *mutation_rate* e *crossover_rate* representam, respectivamente, quantos pipelines devem sofrer mutações (alterações aleatórias) e crossover (devem reproduzir). Por fim, o parâmetro *scoring* tal qual em diversas outras funções implementadas no sklearn nos permite definir a métrica de avaliação do problema (*accuracy*, *precision*, *recall*, etc.).

3.3.10. Time Series Cross-Validation

O processo de construção de um modelo carrega consigo a necessidade de avaliar sua performance e capacidade de generalização. É considerada uma boa prática treinar o modelo em um conjunto de dados e avaliar os resultados em um conjunto de dados totalmente diferente.

Uma forma usual de medir o poder de generalização de um modelo é dividir o conjunto de dados em 2 partes: um subconjunto de treinamento e um subconjunto de teste (train/test split). O primeiro subconjunto, obviamente, é utilizado para construir o modelo e treiná-lo; no segundo subconjunto, aplicamos o modelo previamente treinado, realizamos as previsões para as instâncias que o compõem

e comparamos estas com o valor real, sendo possível, assim, sob o uso de alguma métrica, mensurar a performance.

Outra possibilidade é a utilização de uma metodologia estatística conhecida por cross-validation. Trata-se de uma opção mais robusta e estável, em que realizamos diversas partições do dataset e múltiplos modelos são treinados (um para cada split). A versão mais comum é a k-fold cross-validation, onde k é um número previamente especificado.

Para um melhor entendimento da utilização desta técnica, podemos supor k=5 (opção muito usual em projetos envolvendo dados reais), o que significa que o dataset será dividido em 5 partes aproximadamente de mesmo tamanho, denominadas folds.

Assim, um modelo é treinado usando o primeiro split, ou seja, o primeiro fold é utilizado como conjunto de teste, enquanto os folds de 2 a 5 servem como conjunto de treinamento. Avaliamos, então, o resultado conforme a métrica requerida para este modelo. Em sequência, um novo modelo é treinado tendo o fold 2 como conjunto de teste e os demais (folds 1, 3, 4 e 5) como conjunto de treinamento. Novamente, registramos o resultado para este modelo. O mesmo processo é aplicado tendo os folds 3, 4 e 5 como conjuntos de teste. Após o armazenamento dos resultados para cada um dos 5 modelos treinados, agregamos os resultados de todos os splits, de forma a obtermos uma medida de performance geral. A figura 13 ilustra a aplicação da técnica de 5-fold cross-validation.

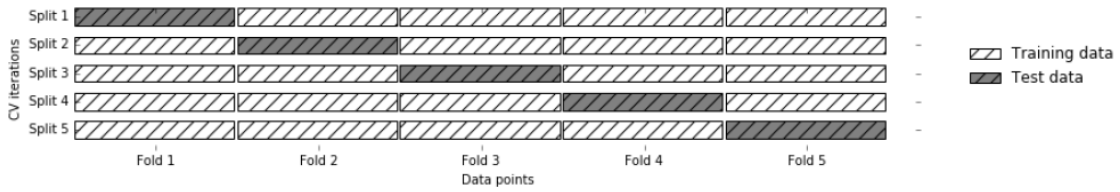

Figura 13. Divisão do dataset pela utilização de 5-fold cross-validation (Fonte: Müller & Guido, 2016)

Há vários benefícios decorrentes da abordagem aplicada em detrimento da utilização de train/test split, dentre os quais podemos citar o fato de que, naquele caso, há um uso mais eficiente dos dados, tendo em vista que todas as instâncias do dataset são utilizadas tanto para treinamento, quanto para teste do modelo. Além disso, é possível obter uma estimativa mais precisa da acurácia do modelo com dados fora da amostra. A principal desvantagem diz respeito ao aumento do custo computacional, pois é necessário treinar k modelos em vez de um único, o que tornará o processo de modelagem aproximadamente k vezes mais lento.

Não obstante todas as vantagens citadas em relação ao uso da técnica de k-fold cross-validation, os dados utilizados nesta tese para modelagem, conforme comentado anteriormente, possuem uma componente temporal. Para ser mais preciso, estamos utilizando um conjunto de dados com várias unidades amostrais (companhias seguradoras) acompanhadas durante vários anos (2013 a 2017) para tentar compreender a dinâmica da insolvência destas empresas. Tais dados são denominados dados em painel ou dados longitudinais.

Neste caso, o uso de cross-validation não é trivial. Perceba que, caso aplicássemos a referida abordagem, em dado momento, estaríamos utilizando dados de 2017 para treinar o algoritmo e

realizar previsões para o ano de 2013. Em outras palavras, estaríamos usando valores do futuro para prever valores no passado, o que não é desejável sob nenhum aspecto, considerando que há uma dependência temporal entre as observações que deve ser preservada.

Neste sentido, existem algumas alternativas para implementação de cross-validation que mantêm a estrutura temporal intacta e nos garantem os benefícios de treinar e testar o modelo diversas vezes, melhorando a estimativa dos resultados. A esse método dá-se o nome de time series cross-validation.

Neste trabalho, utilizamos uma metodologia baseada na seguinte ideia: no primeiro split, considera-se um pequeno subconjunto dos dados para treinamento do modelo e realizam-se previsões nos dados do período imediatamente subsequente, armazenando a performance obtida. Em prosseguimento, as mesmas instâncias que foram usadas para realizar a previsão passam agora a fazer parte do training set (em conjunto com o training set do primeiro split) e os dados do período imediatamente subsequente agora são utilizados para fazer previsão e avaliar os resultados. A figura 14 favorece o entendimento, proporcionando-nos uma análise visual da metodologia.

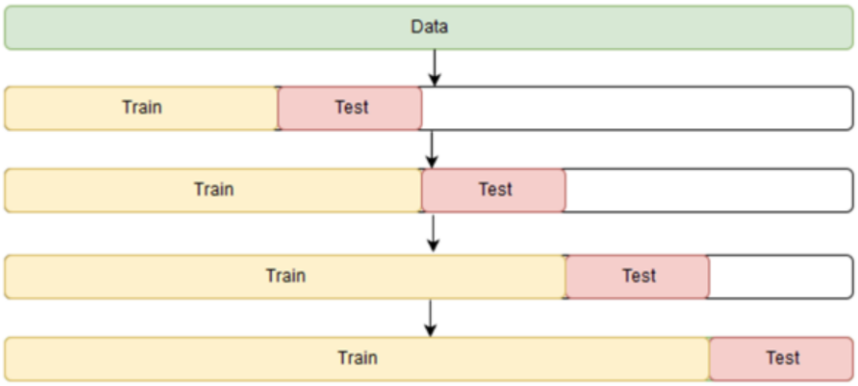

Figura 14. Time series cross validation

O conjunto de dados que separamos para treinamento e validação dos modelos abrange o período de 2013 a 2017. Assim, o primeiro split utiliza os dados de 2013 para treinamento e 2014 para previsões. Em seguida, o segundo split engloba os anos de 2013 e 2014 no training set e o ano de 2015 é destinado ao test set. A mesma lógica se aplica aos demais splits, até que no último split os anos de 2013 a 2016 pertencem ao training set, enquanto o último ano disponível (2017) é usado no test set.

A implementação do algoritmo para criação dos splits foi integralmente realizada pelo autor, haja vista não haver uma função predefinida no sklearn para tal.

3.3.11. Recursive Feature Elimination (RFE)

Muitos projetos de Machine Learning envolvem milhares ou mesmo milhões de variáveis para cada instância de treinamento (Géron, 2019). Embora a priori possa parecer uma grande vantagem possuir tantas variáveis na modelagem, há uma questão associada de tornar o treinamento extremamente lento e, mais ainda, a possibilidade de ocorrência de um problema bem-definido na literatura, conhecido como maldição da dimensionalidade (curse of dimensionality).

Conjuntos de dados com muitas dimensões possuem alto risco de serem muito esparsos, ou seja, a maioria das instâncias de treinamento situam-se muito distantes das demais, o que torna a tarefa de realizar previsões menos confiável do que em menos dimensões, tendo em vista que serão baseadas em extrapolações muito maiores. Em outras palavras, quanto maior o número de dimensões que um dataset possui, maior o risco de overfitting (Géron, 2019).

Neste sentido, há 2 soluções possíveis para mitigar o problema exposto: aumentar o número de instâncias de treinamento para conseguir uma maior densidade deste conjunto ou aplicar técnicas de redução da dimensionalidade. Ocorre que a primeira solução levantada muitas vezes mostra-se impossível na prática, pois o número de observações necessárias para este aumento de densidade cresce exponencialmente com o número de dimensões.

Neste contexto, a seleção de variáveis aparece como uma alternativa viável no sentido de evitar que o modelo sofra dos problemas causados pela maldição da dimensionalidade. A literatura classifica os métodos disponíveis para seleção de variáveis em 3 categorias principais: filter, embedded e wrapper.

Métodos do tipo filter não dependem do algoritmo selecionado para a modelagem. Na realidade, neste método são selecionados subconjuntos das variáveis explanatórias por meio do uso de técnicas estatísticas que avaliam a relação entre a variável que está sendo testada e a variável dependente. As medidas estatísticas utilizadas para aferir a relação dependem do tipo das variáveis dependente e independente. Por exemplo, se ambas forem variáveis numéricas, é comum o uso do coeficiente de correlação de Pearson, que é uma medida de correlação linear entre 2 variáveis. Em contrapartida, se ambas as variáveis são categóricas, utilizamos com frequência o teste χ^2. Uma desvantagem no uso deste método é que este desconsidera o efeito conjunto das variáveis no poder de predição do modelo e sabemos que uma variável pode não ser útil isoladamente, mas muito útil quando considerada em conjunto com outras.

Os métodos do tipo embedded realizam a seleção das variáveis durante o processo de treinamento do modelo. Neste caso, o melhor subconjunto de variáveis é selecionado como uma extensão do processo de aprendizagem. Os exemplos mais comuns deste tipo são os métodos de regularização, como, por exemplo, Ridge e Lasso. Em síntese, são introduzidas penalizações adicionais à função custo do modelo no processo de otimização para obtenção dos parâmetros. Além destes, as árvores de decisão também se enquadram nesta categoria, pois possuem um mecanismo embutido que executa o ranqueamento das variáveis.

Há, ainda, os métodos do tipo wrapper, que utilizam uma abordagem de tentativa e erro para realizar a busca do subconjunto que produz modelos com as previsões de melhor qualidade. Em outras palavras, esta abordagem equivale a realizar um experimento para cada subconjunto de variáveis. Assim, um modelo é treinado sobre este subconjunto e a performance é registrada. O subconjunto que leva ao modelo com melhor resultado é, então, selecionado. De forma geral, a principal desvantagem desta técnica é o custo computacional muito maior relativamente às demais, tendo em vista que envolve o treinamento de múltiplos modelos.

O método Recursive Feature Elimination (RFE), objeto de interesse principal desta seção, enquadra-se na categoria de wrappers. A seguir, apresentamos um passo a passo descritivo do algoritmo:

1. Inicialmente, um modelo é treinado utilizando todas as variáveis de entrada disponíveis.

2. Cada variável é ranqueada pela sua importância relativa no modelo.
3. A cada iteração, as n variáveis com menor importância relativa são descartadas.
4. O modelo é retreinado com o novo conjunto de preditores e as novas importâncias relativas de cada variável são mensuradas (a importância de cada variável pode variar conforme utilizamos diferentes datasets).
5. O algoritmo se repete até que se alcance o número mínimo de variáveis especificado previamente.

Na biblioteca sklearn, temos a implementação de RFE com cross-validation por meio da função RFECV(). Assim, a cada iteração do algoritmo, utiliza-se cross-validation para avaliar o modelo. Caso este apresente uma performance melhor após eliminarmos as variáveis menos importantes do passo anterior, o algoritmo continua para a próxima iteração. Por outro lado, se o modelo apresentar um resultado pior após a eliminação das variáveis, o algoritmo se encerra e o conjunto de variáveis do passo anterior é selecionado.

Dentre os hiperparâmetros existentes, podemos destacar os seguintes: estimator, step e scoring. O primeiro nos permite selecionar o algoritmo que vai ser utilizado como base para o treinamento, como Decision Trees ou Logistic Regression, por exemplo. Já o segundo define o número ou o percentual de variáveis que serão descartadas a cada iteração. Por fim, o parâmetro scoring especifica a métrica que será utilizada para avaliar o modelo durante o processo de cross-validation.

3.3.12. Análise de Componentes Principais (PCA)

Conforme visto na seção anterior, um conjunto de dados com muitas dimensões dificulta muito a obtenção de bons resultados, em decorrência da maldição da dimensionalidade, que pode levar os algoritmos à ocorrência de overfitting.

Neste contexto, a Análise de Componentes Principais (PCA) é a técnica de redução de dimensionalidade mais utilizada. Em suma, PCA é uma técnica não supervisionada – não necessita da variável resposta no processo de extração – que projeta as observações em um conjunto de componentes principais de forma a manter o máximo da variância original do dataset.

Em outras palavras, a ideia por trás deste método é conseguir reduzir a dimensionalidade dos dados com a menor perda de informação possível. Repare que, aqui, diferentemente das opções de seleção de variáveis levantadas na seção 3.3.11., o produto final são novas variáveis, resultantes da projeção dos dados em um espaço de dimensão inferior, enquanto lá as variáveis são as mesmas das originais, exceto pelo fato de ser um subconjunto menor.

Resumidamente, o algoritmo é processado de forma a encontrar a direção que contém a maior parte da variação dos dados. A essa direção dá-se o nome de primeira componente principal. Da mesma forma, a segunda componente principal, ortogonal à primeira, é aquela que contribui ao máximo para a variação restante. O algoritmo, ao fim, produz tantas componentes principais quanto o número de dimensões do dataset.

A figura 15 nos apresenta de forma simplificada como se daria o comportamento de uma análise de componentes principais caso aplicada a um conjunto de dados simples com apenas 2 dimensões, com

um formato semelhante ao de uma elipse. Claramente, a primeira componente se situa ao longo do eixo maior da elipse, enquanto resta à segunda componente apenas uma pequena parte da variação, na direção do eixo menor (ortogonal ao primeiro).

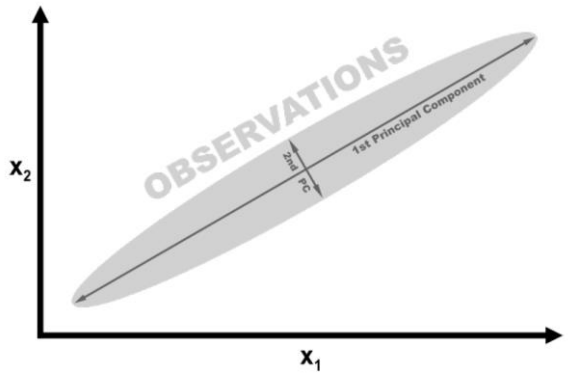

Figura 15. Exemplo da aplicação de análise de componentes principais em um dataset de 2 dimensões (Albon, 2018)

A biblioteca sklearn, mais uma vez, possui uma implementação deste algoritmo em Python. Dentre os principais parâmetros da função, podemos citar n_components. Há 2 possibilidades para este parâmetro: caso receba um valor maior do que um, representa o número de componentes que o algoritmo deve retornar; caso receba um valor decimal entre 0 e 1, representa a variância que o modelo consegue explicar. Esta última opção resolve de antemão o problema de ter que especificar o número de componentes.

Outro parâmetro interessante é o svd_solver, que se preenchido com randomized, permite uma solução via modelos estocásticos em uma quantidade de tempo significativamente menor.

Por fim, cabe atenção ao fato de que certos conjuntos de dados podem não se mostrar adequados à aplicação da técnica em estudo por se tratar de uma transformação linear. Neste cenário, é importante destacar que existem técnicas alternativas como KernelPCA, em que é possível definir diferentes tipos de kernel como rbf, polynomial ou sigmoid.

3.3.13. SMOTE e Outras Técnicas para Lidar com Datasets Desbalanceados

Uma das maiores dificuldades enfrentadas em projetos de classificação com dados reais é o forte desbalanceamento entre as classes, isto é, uma desproporção do número de exemplos existentes nas diferentes classes (Fernández et al., 2018). Especificamente neste trabalho, que trata de um problema de classificação binária, há um desequilíbrio entre as classes insuficiente e suficiente da ordem de 16% e 84%, respectivamente.

Há uma questão especial em se treinar algoritmos utilizando datasets muito desbalanceados, que está relacionada com o fato de que estes algoritmos não sabem lidar com o desequilíbrio do número de instâncias entre as classes e terminam por obter uma performance muito ruim na classe minoritária, que usualmente é a classe mais importante – lembrando que, neste trabalho, o foco é justamente a previsão da insolvência das empresas, que possui uma quantidade de observações muito inferior a outra classe, como é esperado em um mercado saudável.

Mais à frente, abordaremos as métricas utilizadas para avaliação dos resultados neste projeto, porém, apenas para elucidar a questão da maior relevância da classe minoritária, é muito mais importante identificar corretamente uma empresa que está caminhando para uma situação de insolvência, uma vez que as consequências da não detecção podem ser fatais, enquanto um falso positivo para uma companhia financeiramente saudável não seria tão prejudicial – ensejaria, no máximo, um alerta de fiscalização.

Na literatura especializada, muitos estudos têm mostrado que, para diversos tipos de classificadores, rebalancear o conjunto de dados melhora significativamente a performance geral da classificação em comparação com um conjunto de dados que não passou por pré-processamento (Fernández et al., 2018). Dentre as várias soluções propostas disponíveis para lidar com o problema em questão, nesta seção, vamos nos ater a um grupo em especial denominado técnicas de reamostragem.

Esta categoria engloba outras 3 subcategorias principais: métodos de undersampling, métodos de oversampling e métodos híbridos. Em resumo, o primeiro tipo cria um subconjunto do dataset original eliminando instâncias pertencentes à classe majoritária. No segundo caso, a ideia de reequilibrar as classes utiliza uma estratégia diferente, por meio da criação de um novo dataset, contendo integralmente o dataset original e novas instâncias, seja pela replicação de instâncias pertencentes à classe minoritária, seja pela criação de instâncias inteiramente novas a partir das existentes. Por fim, os métodos híbridos combinam de alguma forma ambas as abordagens anteriores.

Relativamente à técnica de undersampling, uma das principais desvantagens é o risco de descartar informação útil, que poderia ser importante no processo de treinamento. No que tange à técnica de oversampling, quando aplicada apenas pela mera replicação de exemplos já existentes na classe minoritária (random oversampling), diversos autores reportam que é potencialmente um fator de aumento da probabilidade de overfitting.

Assim, apresentamos nesta seção uma abordagem de oversampling mais sofisticada, conhecida por Synthetic Minority Oversampling TEchnique (SMOTE). Ao contrário da metodologia que replica aleatoriamente exemplos pertencentes à classe minoritária, SMOTE tem por fundamento a criação de instâncias sintéticas. De forma geral, o procedimento de introdução destas novas instâncias está ilustrado na figura 16.

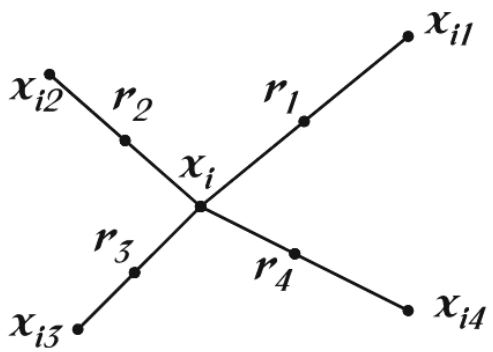

Figura 16. Criação de novas instâncias por meio do algoritmo SMOTE (Fernández et al., 2018)

A observação x_i no centro da figura, pertencente à classe minoritária, servirá de referência para sintetizar novos pontos. Definida previamente alguma métrica para cálculo da distância, selecionaremos os vizinhos mais próximos integrantes da mesma classe (x_{i1} a x_{i4}). Finalmente, é executada uma interpolação, para obtenção das novas observações (r_1 a r_4). O algoritmo completo segue na figura 17.

Algorithm 3 SMOTE algorithm

function SMOTE(T, N, k)
Input: T; N; k ▷ #minority class examples, Amount of oversampling, #NNs
Output: $(N/100) * T$ synthetic minority class samples
Variables: $Sample$[][]: array for original minority class samples;
$newindex$: keeps a count of number of synthetic samples generated, initialized to 0;
$Synthetic$[][]: array for synthetic samples
  if $N < 100$ **then**
    Randomize the T minority class samples
    $T = (N/100)*T$
    $N = 100$
  end if
  $N = (int)N/100$ ▷ The amount of SMOTE is assumed to be in integral multiples of 100.
  for $i = 1$ to T **do**
    Compute KNN for i, and save the indices in the $nnarray$
    POPULATE(N, i, $nnarray$)
  end for
end function

Algorithm 4 Function to generate synthetic samples

function POPULATE(N, i, $nnarray$)
Input: N; i; $nnarray$ ▷ #instances to create, original sample index, array of NNs
Output: N new synthetic samples in $Synthetic$ array
  while $N \neq 0$ **do**
    nn = random(1,k)
    for attr = 1 to numattrs **do** ▷ $numattrs$ = Number of attributes
      Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
      Compute: $gap = random(0, 1)$
      $Synthetic[newindex][attr] = Sample[i][attr] + gap \cdot dif$
    end for
    $newindex++$
    $N--$
  end while
end function

Figura 17. Algoritmo de implementação da técnica SMOTE (Fernández et al., 2018)

Uma das principais desvantagens do algoritmo original SMOTE é o fato de que não considera a classe majoritária no processo de criação das novas instâncias, podendo sintetizar exemplos ambíguos, caso haja forte sobreposição entre as classes. Além disso, não se mostra uma metodologia muito eficiente para conjuntos de dados com muitas dimensões. Com o objetivo de corrigir suas principais deficiências, há diversas extensões de SMOTE, como Borderline-SMOTE, SVM SMOTE, dentre outras.

Ademais, cabe aqui uma importante ressalva acerca da combinação de técnicas: o próprio artigo que descreveu inicialmente a técnica, de autoria de Chawla e outros autores, informa que a combinação de SMOTE e undersampling performa melhor do que a simples aplicação de undersampling.

A implementação em Python para todas as técnicas descritas nesta seção se dá por meio da biblioteca imblearn. Especialmente neste trabalho, fizemos uso de 3 funções, quais sejam, RandomUnderSampler, RandomOverSampler e SMOTE.

Todas as 3 funções possuem um parâmetro em comum chamado sampling_strategy, que tem por objetivo definir a proporção final desejada entre o número de instâncias que comporão a classe minoritária dividido pelo número de instâncias que comporão a classe majoritária, após a utilização da técnica desejada.

Além deste, há um outro importante parâmetro – já conhecido em diversas funções da biblioteca sklearn – denominado random_state, que estabelece uma semente para geração dos números aleatórios no algoritmo e admite, em última análise, obter os mesmos resultados nas próximas vezes que o algoritmo for rodado, garantindo a reprodutibilidade desejada em pesquisas científicas.

Por fim, destacamos a existência de um hiperparâmetro disponível na função SMOTE. Trata-se de k_neighbors, que oferece um ajuste para o número de vizinhos que serão utilizados pelo algoritmo para construção das amostras sintéticas, conforme explicado anteriormente na descrição do algoritmo SMOTE.

4. DADOS

Neste capítulo, abordaremos as principais etapas relacionadas ao processamento dos dados, desde a sua coleta e fontes, passando pela preparação e primeiros entendimentos e finalizando com o processamento e feature engineering (processo de utilização do conhecimento do assunto para selecionar e transformar as variáveis mais relevantes a partir dos dados brutos obtidos).

4.1. COLETA DOS DADOS

Os dados utilizados no desenvolvimento deste trabalho foram em sua maioria fornecidos pela Superintendência de Seguros Privados (Susep), como parte do processo de afastamento para cursar pós-graduação no exterior, e tendo em vista a condição do autor de servidor público integrante do quadro de pessoal da autarquia. Conforme citado na seção 1.6, foi retirada desta tese qualquer menção expressa às empresas componentes do mercado supervisionado, garantindo o sigilo de informações sensíveis, por meio de processo de anonimização.

4.1.1. SQL Server Management Studio

O processo de extração dos dados teve início no SQL Server Management Studio, onde temos acesso a um servidor, que é composto por dezenas de bancos de dados, notadamente o SAPIEMS, sigla para Sistema de Acompanhamento e Processamento de Informações e Estatísticas do Mercado Segurador.

O banco de dados supracitado é utilizado para armazenar dados encaminhados mensalmente pelas companhias por meio do Formulário de Informações Periódicas - FIPSUSEP. Os dados que compõem o formulário vão desde simples informações cadastrais das empresas (endereço, membros participantes dos órgãos estatutários, acionistas, dentre outros) a dados de menor nível de granularidade e que contém detalhes operacionais da empresa, como prêmios ganhos, provisões técnicas, sinistros ocorridos e ativos de resseguro segregados por ramo de seguro em que a companhia opera.

Há, também, quadros com detalhes das parcelas para cálculo de capital baseado em risco e dos ativos garantidores das provisões técnicas, o que por si só é suficiente para demonstrar a riqueza de informações existentes e a complexidade envolvida na extração dos dados. Maiores detalhes acerca das tabelas estão disponíveis neste link, que contém um arquivo descritivo de todos os atributos de cada tabela, bem como os relacionamentos entre estas.

Ademais, entendemos a necessidade de que seja fornecida uma breve descrição das tabelas que serviram como fonte para o projeto. Do universo total de tabelas armazenadas no servidor, 14 foram manipuladas para construção de variáveis explanatórias ou usadas como insumo para posterior transformação em variáveis. Abaixo, apresentamos a relação das tabelas mais relevantes para o sucesso do presente trabalho:

TABELA	DESCRIÇÃO
BIB_DEFCAMPOS	Contém todos os campos existentes na base de dados.

ENTIDADES	Armazena os dados das empresas do mercado segurador para um determinado mês de referência.
LISTAENTIDADES	Contém a lista dos códigos das entidades cadastradas no sistema.
LOGCARGA	Armazena as informações sobre os pedidos de recarga.
PARTICIPACOESACIONARIAS	Registra as participações acionárias de entidades, em determinada data de referência, seja como acionistas ou como participadas.
PESSOAS	Armazena informações adicionais, para determinada data de referência, sobre entidades que têm alguma participação acionária, seja como acionistas ou como participadas, tais como a quantidade total de ações.
QUADROS	Armazena os quadros tabulares oferecidos.
RAMOSOPERACAO	Lista os ramos de seguros com os quais a entidade operou para um determinado mês de referência.
RELACIONAMENTOACIONISTA	Armazena os organogramas de acionistas/participadas.
VALORESBALANDEMON	Registra os valores das demonstrações contábeis.
VALORESMOVRAMOS	Registra os valores de prêmios, sinistros, provisões, despesas de comercialização, variações de despesas, despesas e custos diferidos, direitos creditórios e resultados líquidos por ramos de seguros para uma entidade e data de referência.
VALORESMUTACAO	Registra os valores das reservas de capital, de reavaliação e de lucros.
VALORESRESMOVGRUPOS	Amazena as informações relativas aos quadros de mapas demonstrativos por grupos.
VIGENCIACAMPOS	Armazena a vigência dos campos.

Tabela 1. Descrição das tabelas utilizadas para construção das variáveis do projeto

Como resultado dos trabalhos de extração efetuados nesta etapa, obtivemos como produto final 2 tabelas, que foram agregadas com as demais já no ambiente Python.

4.1.2. Demais Informações

Além de todo o material de estudo disponibilizado por meio das bases de dados anteriormente descritas, houve a incorporação de informações extras. Por exemplo, os dados de Patrimônio Líquido Ajustado (PLA) e as parcelas de capital baseado em risco, que são resultados de apuração ou conferência interna, foram enviados pela Coordenação de Monitoramento de Ativos Financeiros e Macroprudencial – COMAP.

Já as séries históricas dos indicadores macroeconômicos foram manualmente obtidas a partir do site do Instituto Brasileiro de Geografia e Estatística – IBGE, que disponibiliza as séries temporais para os indicadores de renda média por trabalhador, taxa de desemprego, inflação (Índice Nacional de Preços ao Consumidor Amplo - IPCA) e Produto Interno Bruto (PIB brasileiro), exceto pelo indicador de taxa de juros (SELIC), que foi extraído do repositório do Banco Central do Brasil (BCB).

Adicionalmente, os dados relacionados à caracterização de grupos prudenciais foram disponibilizados pela Coordenação de Monitoramento de Solvência e Contabilidade (COMOC). Em essência, um grupo prudencial é representado por um conjunto de supervisionadas no qual um mesmo sócio ou grupo de sócios detém o controle ou participa em regime de controle conjunto. Tal variável carrega uma

informação importante, pois empresas pertencentes a grandes conglomerados financeiros costumam estar sujeitas a controles internos mais robustos e acesso facilitado a capital para cobrir eventuais problemas relacionados a prejuízos na operação.

4.1.3. Python

Os subprodutos gerados pelas consultas em SQL foram agregados aos da etapa anterior para formação de uma tabela única, por meio da utilização da linguagem Python. A maior dificuldade nesta etapa esteve relacionada ao ajuste das tabelas para que a variável resposta representasse o valor em t, e as variáveis explicativas representassem os valores defasados por 1 e 2 anos, ou seja, em $t-1$ e $t-2$.

Para um melhor entendimento, imagine que, inicialmente, foi extraído das etapas anteriores o valor da variável resposta nos diversos anos disponíveis. A tabela 2 mostra um exemplo fictício para uma dada companhia no período de 2013 a 2018 e os respectivos valores da variável INSUFICIÊNCIA.

ANO	INSUFICIÊNCIA
2013	0
2014	1
2015	0
2016	0
2017	1
2018	0

Tabela 2. Exemplo fictício da variável resposta para uma companhia ao longo do período de análise

Agora, por motivos de clareza, imagine de forma análoga que a tabela 3 representa os valores das variáveis explicativas no mesmo período para a mesma empresa.

ANO	IND1	IND2	IND3
2013	...	...	...
2014	...	...	...
2015	...	...	...
2016	...	...	...
2017	...	...	...
2018	...	...	...

Tabela 3. Exemplo fictício das variáveis explicativas para uma companhia ao longo do período de análise

A mera fusão das 2 tabelas, tendo como chave o ano de referência, resultaria na seguinte tabela:

ANO	INSUFICIÊNCIA	IND1	IND2	IND3
2013	0	...	...	...
2014	1	...	...	...
2015	0	...	...	...
2016	0	...	...	...
2017	1	...	...	...
2018	0	...	...	...

Tabela 4. Resultado da fusão das 2 tabelas anteriores

A aplicação de qualquer modelo de machine learning no conjunto de dados apresentado, em que todas as variáveis explicativas e a variável resposta são contemporâneas, não estaria coerente com o desenho do problema apresentado neste projeto. Intuitivamente, é fácil notar que desejamos utilizar dados do passado para realizar previsões futuras.

Para alcançarmos o resultado desejado, é necessário que os indicadores, representados na tabela por IND1, IND2 e IND3, estejam posicionados com 1 e 2 anos, respectivamente, de antecedência relativamente à variável INSUFICIÊNCIA. A tabela 5 ilustra o que acabamos de expor.

ANO	INSUFICIÊNCIA $_{t}$	IND1 $_{t-1}$	IND1 $_{t-2}$	IND2 $_{t-1}$	IND2 $_{t-2}$	IND3 $_{t-1}$	IND3 $_{t-2}$
2013	0	...	...	...	...	...	...
2014	1	...	...	...	...	...	...
2015	0	...	...	...	...	...	...
2016	0	...	...	...	...	...	...
2017	1	...	...	...	...	...	...
2018	0	...	...	...	...	...	...

Tabela 5. Desenho correto do problema, com as variáveis explicativas defasadas

Cabe recordar que, nos exemplos fictícios utilizados nesta seção, para facilitar a explicação, foi utilizada apenas uma empresa, embora o conjunto de dados completo seja composto de mais de 100 empresas para cada unidade de tempo.

4.2. ENTENDIMENTO E PREPARAÇÃO DOS DADOS

Nesta seção, daremos uma visão global sobre o conjunto de dados, tentando obter as primeiras impressões e hipóteses, bem como verificaremos a existência de erros e descreveremos as etapas de processamento para que os dados se encontrassem em condições ideais de servirem de input aos modelos que serão testados.

Inicialmente, vale mencionar que o trabalho contava com dados relativos aos anos de 2011 a 2018. Entretanto, como necessitamos de 2 anos de defasagem para a construção das variáveis explanatórias, o dataset se restringe ao período de 2013 a 2018. Para cumprimento da fase de treinamento e validação dos modelos, foram separados os dados de 2013 a 2017. Em contrapartida, o ano de 2018 foi exclusivamente destinado ao teste dos modelos selecionados no passo anterior.

Assim, o conjunto de treinamento é composto de 583 observações, enquanto o conjunto de teste possui um total de 125 observações. A tabela 6, apresentada abaixo, mostra a quantidade de empresas por ano no training set e no test set.

	ANO	TOTAL DE EMPRESAS
TRAINING SET	2013	114
	2014	110
	2015	115
	2016	120
	2017	124
TEST SET	2018	125

Tabela 6. Visão geral do dataset, contendo o número de observações por ano de referência

É possível notar, ainda, que há uma oscilação no número de observações a cada ano, o que é natural e totalmente esperado, tendo em vista que novas empresas são criadas, algumas empresas encerram sua operação de forma compulsória (por exemplo, por meio de liquidação extrajudicial) ou espontânea (decisão estratégica dos acionistas), além dos casos de reorganização societária (fusão, cisão e incorporação de empresas).

Porém, sob a ótica de entendimento dos dados e conhecimento para o processo de modelagem, mais importante do que saber a quantidade de empresas que compõem cada ano, é conhecer a distribuição destas empresas nas classes de interesse (relembrando: a classe 0 é determinada pelas empresas que apresentam suficiência de capital, enquanto a classe 1 pelas empresas que apresentam insuficiência).

A figura 18, a seguir, evidencia o fato reportado na seção 3.3.13. acerca do desequilíbrio no número de instâncias que compõem a classe positiva (insuficiência) em relação ao número das que compõem a classe negativa (suficiência). Percebe-se, portanto, que o percentual de empresas insuficientes varia de 18% em 2013, atingindo o seu pico (25%) no ano de 2014 e iniciando a sua tendência de queda em 2015 (20%), passando para 11% em 2016, até alcançar o mínimo de 8% em 2017. De forma global, durante todo o período, a proporção é de 84% para as suficientes e 16% para as insuficientes.

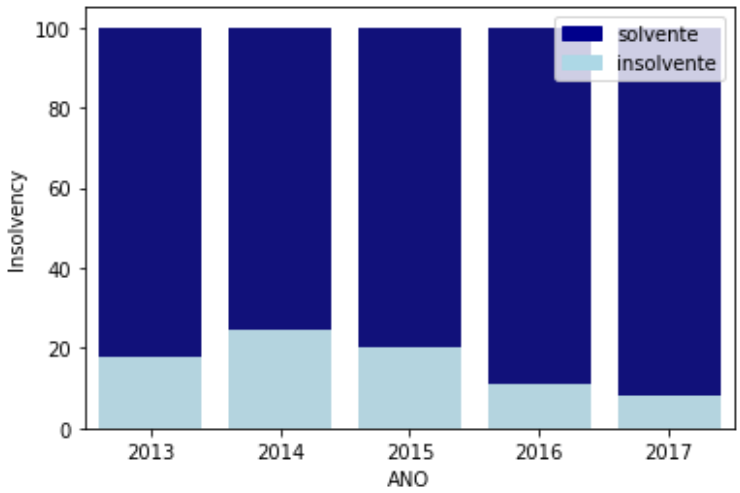

Figura 18. Percentual de instâncias que compõe cada uma das classes do problema

Desta forma, foi necessário lançar mão das técnicas descritas anteriormente (SMOTE, Random Undersampler e Random Oversampler), na tentativa de mitigar os efeitos deste desbalanceamento, que prejudica o poder de generalização dos classificadores, principalmente na classe que se encontra em desvantagem na quantidade de observações. Mais à frente, descreveremos os resultados advindos da aplicação destas técnicas.

Em seguida, no que se refere à dimensionalidade do dataset, trabalhamos com um conjunto final de 103 variáveis independentes. A respeito deste ponto, cabe informar que, além da utilização das variáveis defasadas por 1 e 2 anos, a depender do indicador, também achamos interessante utilizar a variação percentual entre os períodos $t-1$ e $t-2$.

Tomemos, por exemplo, a parcela do capital de risco de subscrição e imaginemos que, para uma dada companhia, houve um aumento expressivo de um ano para o ano seguinte. Tal fato pode significar que a empresa esteja assumindo muito risco, o que pode ser problemático caso os controles internos da empresa não se adequem à nova realidade. Assim, a variação da parcela do capital no período ($\Delta CRsubs_{\ t-1,t-2}$) pode ser uma variável muito informativa para o modelo.

A princípio, pode causar estranheza o fato de que, na fase de entendimento dos dados, já tenhamos criado novos atributos, algo que, pela metodologia CRISP-DM, deveria se realizar na fase seguinte. Isso se justifica pelo fato de que o autor possui conhecimento prévio dos dados em uso, por já ter trabalhado no setor de análise de empresas, produzindo relatórios que subsidiam as equipes de fiscalização. Além disso, a citada abordagem, embora possua uma distribuição lógica das etapas de execução do projeto, é bastante flexível, como alertado na seção 3.1., permitindo transitar entre os diferentes estágios no decorrer dos trabalhos.

No que tange aos tipos de dados, a maioria das variáveis é do tipo numérica (91 variáveis), havendo, ainda, um total de 12 variáveis do tipo binária (dummies), que são variáveis que assumem valor 1 caso possuam determinada característica ou valor 0, caso não a possuam.

Inicialmente, para obter as primeiras impressões sobre as variáveis, calculamos diversas estatísticas descritivas para os dados em estudo. Utilizamos, para tal, média, desvio-padrão, 1º, 2º e 3º quartis e optamos, ainda, por incluir alguns percentis nas caudas da distribuição (1º, 5º, 95º e 99º percentis) para facilitar a identificação de eventuais outliers.

Para um melhor entendimento do relacionamento entre as variáveis explicativas numéricas, fizemos uso de gráficos de dispersão, que também são muito úteis para identificar a existência de outliers. A figura 19 mostra dois exemplos de scatterplots, que permitem a análise do relacionamento entre as variáveis ICA_t-1 e ROE_t-1.

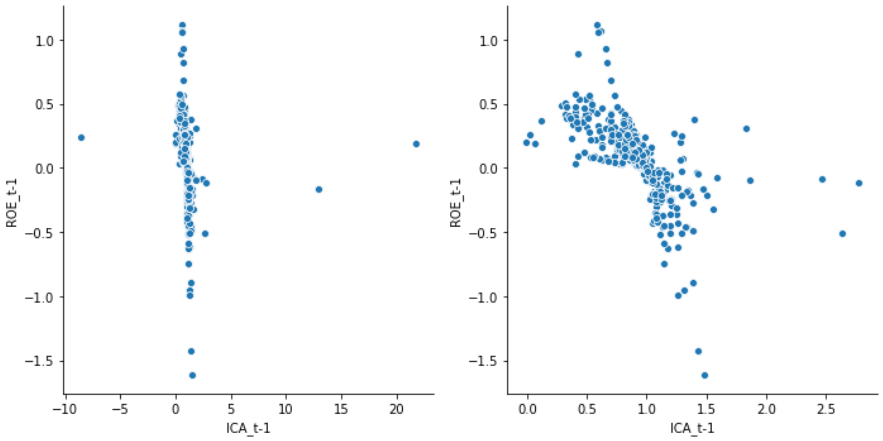

Figura 19. Scatterplot entre as variáveis ICA_t-1 e ROE_t-1, sem e com a presença de outliers, respectivamente

O primeiro exemplo é especialmente didático, pois oferece um recurso visual para identificação de outliers. É fácil ver que há 3 empresas com o valor do ICA muito discrepante do valor das demais. O ICA, em síntese, é um índice que mede o montante de despesas consumidas relativamente ao montante de receitas arrecadadas. Empresas com ICA muito alto demonstram um potencial problema, pois tiveram, dentro de um mesmo ano, um volume muito maior de despesas do que de receitas.

Em um mercado tão heterogêneo quanto o de seguros no Brasil, a existência de outliers, não por erro nos dados, é algo relativamente comum. Isto se deve ao fato de que coexistem no mercado tanto empresas gigantescas pertencentes a conglomerados econômico-financeiros, quanto empresas com estrutura enxuta que nasceram a partir de pequenas cooperativas.

O segundo gráfico, ainda na figura 19, representa o scatterplot, sem a interferência dos outliers, que acabam por influenciar na escala do eixo x e atrapalham a visualização da relação entre as variáveis. Neste caso, é possível perceber uma relação inversamente proporcional, ou seja, quanto maior o valor do ICA, menor tende a ser o valor do ROE, que relaciona o lucro com o patrimônio líquido da empresa. Mais uma vez, o gráfico apenas confirma um comportamento teórico já esperado.

Adicionalmente, também é importante entender o relacionamento entre cada variável explicativa e a variável dependente. Para este fim, fizemos uso de boxplots, que mostram a distribuição de uma variável numérica para cada classe pertencente ao eixo x, como pode ser visto na figura 20.

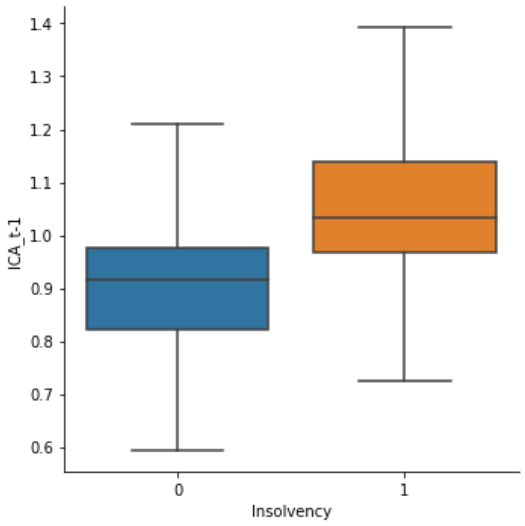

Figura 20. Boxplots para o ICA, segregados por cada classe do problema

Neste caso, estamos analisando a distribuição da variável ICA_t-1, sendo possível identificar uma diferença a maior entre os valores medianos e dos demais quartis para as empresas que apresentaram insuficiência de capital, o que está coerente com a teoria. Uma análise análoga foi realizada para os demais indicadores, sempre que cabível.

No que tange ao processo de limpeza dos dados, esta etapa comumente se inicia por meio do tratamento de valores nulos. Das 103 variáveis disponíveis, 2 possuíam mais de 10% de valores faltantes e, portanto, foram excluídas. Relativamente às demais variáveis, 11 destas apresentaram pelo menos um valor nulo, mas em percentual inferior a 3% do total. Nestes casos, optou-se por mantê-las, empregando alguma estratégia de imputação de valores, analisando caso a caso.

Para as variáveis que possuíam valores nulos gerados a partir de uma divisão de zero por zero, imputamos o próprio valor zero em substituição. Nos casos remanescentes, os valores nulos foram substituídos pela média dos valores assumidos pela mesma variável da empresa no período de treinamento. Por exemplo, em 2015, uma dada companhia apresentou valor nulo para a variável Porte_PREM_t-1. Assim, como a mesma empresa apresentava valores para os anos de 2013, 2014, 2016 e 2017, utilizamos a média destes valores em substituição ao valor nulo original.

O baixo número de casos de valores faltantes se deve ao fato de que a maior parte dos dados passa por monitoramento constante das equipes de fiscalização, o que não impede a ocorrência de pequenos erros de forma pontual.

Além da presença de valores nulos, também foi notada a existência de valores np.inf em 2 variáveis explicativas, em função da divisão de valores estritamente positivos por zero. A existência de valores infinitos advém da criação de variáveis a partir da divisão dos valores de outros 2 preditores. Em ambos os casos, não passavam de 3% do total do dataset, motivo pelo qual resolvemos manter as variáveis, substituindo os valores pelo valor máximo assumido por aquela variável.

Em seguida, uma análise do coeficiente de correlação de Pearson foi executada para avaliar o relacionamento entre as variáveis numéricas. Variáveis cujo valor absoluto da correlação esteve acima de 0,7 foram removidas do dataset. Tal procedimento nos levou a um conjunto final de dados com 71 variáveis explicativas, das quais 59 são numéricas e 12 binárias.

5. RESULTADOS E DISCUSSÃO

Neste capítulo, apresentaremos a estratégia de avaliação utilizada neste trabalho, bem como discutiremos os resultados para os algoritmos de *machine learning* aplicados, após a aplicação de todas as técnicas de preparação e processamento detalhadas no capítulo anterior. Para tal, iniciaremos com uma breve explicação das métricas selecionadas para avaliação da performance dos modelos. Após, introduziremos os resultados para os algoritmos *baseline*. Em seguida, são apresentadas as performances para cada algoritmo após a aplicação de técnicas de redução de dimensionalidade (Recursive Feature Elimination e PCA). Por fim, avaliamos a performance dos modelos com a introdução de SMOTE, RandomUnderSampler e RandomOverSampler para mitigar problemas decorrentes do elevado grau de desbalanceamento do dataset nos modelos de previsão.

5.1. MÉTRICAS

A literatura científica relacionada à aplicação de modelos de machine learning é vasta e oferece diferentes métricas para avaliação de performance. A métrica mais usual é conhecida por Accuracy e mede, em linhas gerais, o número de previsões corretas em relação ao total de previsões realizadas. De forma a nos auxiliar na compreensão das métricas disponíveis, introduziremos o conceito de matriz de confusão, que é uma ferramenta que calcula a frequência de cada possível resultado previsto por um modelo aplicado em um test set. Para um problema de classificação binária (em que, por convenção, denominamos as classes como positiva ou negativa), há somente 4 combinações possíveis:

- True Positive (TP): uma instância pertencente à classe positiva foi corretamente prevista pelo modelo, ou seja, classificada como pertencente à classe positiva;
- True Negative (TN): uma instância pertencente à classe negativa foi corretamente prevista pelo modelo, ou seja, classificada como pertencente à classe negativa;
- False Positive (FP): uma instância pertencente à classe negativa foi incorretamente prevista pelo modelo, ou seja, classificada como pertencente à classe positiva; e
- False Negative (FN): uma instância pertencente à classe positiva foi incorretamente prevista pelo modelo, ou seja, classificada como pertencente à classe negativa.

A seguir, a figura 21 fornece a estrutura de uma matriz de confusão:

		Predito	
		Negativa	Positiva
Real	Negativa	TN	FP
	Positiva	FN	TP

Figura 21. Estrutura de uma matriz de confusão.

A Accuracy é uma métrica muito difundida, mas sua utilização em problemas de classificação binária cuja quantidade de observações em cada uma das classes é muito desbalanceada mostra-se inadequada. Tomemos, por exemplo, o conjunto de dados utilizado neste trabalho, em que 84% das instâncias pertencem à classe negativa (suficiência). Neste caso, o mais simples dos classificadores, que previsse toda empresa como suficiente já seria capaz de atingir uma Accuracy equivalente a 84%. Casos mais extremos (imaginemos uma desbalanceamento da ordem 99,9%/0,1%) produziriam distorções ainda maiores.

Nesta tese, foram selecionados 3 diferentes tipos de métricas amplamente disseminadas e discutidas na literatura: Recall, Precision e F1-Measure. A seguir, apresentaremos a definição matemática para cada uma delas, bem como a interpretação e sua aplicabilidade ao trabalho.

A Recall pode ser definida pela equação 28:

$$Recall = \frac{TP}{TP+FN} \tag{28}$$

Modelos com alto valor de Recall minimizam a ocorrência de falsos negativos, ou seja, nos permitem inferir a nossa confiança de que o modelo foi capaz de capturar todas as instâncias positivas.

Para tornar a interpretação mais razoável, utilizaremos o nosso próprio problema. Sob a ótica de um regulador, há uma preocupação especial com a possibilidade de alguma seguradora quebrar e o regulador não conseguir tomar nenhuma ação prévia, pois isso significaria que potencialmente milhões de consumidores estariam descobertos (sem indenização em caso de sinistro).

Desta forma, o custo associado com um FN é muito mais alto do que o custo associado a um FP. Explique-se: se uma empresa suficiente é classificada incorretamente como insuficiente, a maior consequência é o endurecimento de ações regulatória como uma priorização de fiscalização. Já o contrário – classificar uma companhia como suficiente quando, na realidade, ela está insuficiente – significaria não proceder com as ações regulatórias que poderiam salvá-la de uma possível falência.

No caso da Precision, a definição pode ser dada pela equação 29:

$$Precision = \frac{TP}{TP+FP} \tag{29}$$

Modelos com alto valor de Precision minimizam a ocorrência de falsos positivos, ou seja, nos permitem determinar com que frequência, quando o modelo prevê uma instância como positiva, essa previsão se torna correta.

Na mesma linha interpretativa utilizada anteriormente, imagine que para aumentar a probabilidade de capturar a ocorrência das companhias insuficientes, o modelo passasse a classificar todas as instâncias do test set como positiva. Tal fato poderia implicar um aumento da ocorrência de eventos FP.

Novamente, sob a ótica do regulador, há uma capacidade limitada para um acompanhamento mais próximo das empresas supervisionadas. A utilização de modelos de classificação é fundamental para uma priorização de recursos humanos e financeiros e para definição das companhias que necessitam de ações regulatórias mais intensas. Desta forma, o custo associado com um FP não é desprezível e não podemos ignorar a necessidade de atingirmos valores razoáveis de Precision.

Por fim, a equação 30 define a representação matemática da F1-Measure:

$$F1\ Measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{30}$$

É fácil ver que a F1-Measure é uma forma de combinar em uma única medida os conceitos expressados pela Precision e pela Recall. Para ser mais preciso, a F1-Measure é a média harmônica entre as 2 métricas anteriormente referidas. Enquanto a média aritmética pondera igualmente todos os valores,

a média harmônica dá um peso muito maior a valores pequenos. Consequentemente, o classificador somente alcançará altos valores para a F1-Measure se Recall e Precision também apresentarem altos valores.

Resumidamente, todas as métricas apresentadas apresentam valores no intervalo [0,1] e quanto maior o valor apresentado, melhor a performance do classificador. Além disso, é importante mencionar que as métricas foram aplicadas tanto no training set, para seleção dos hiperparâmetros dos modelos, quanto no test set, para aferição da performance em novas observações e avaliação do poder de generalização dos modelos selecionados.

5.2. BASELINE

Esta seção apresenta os resultados da utilização dos algoritmos supramencionados, após passarem por uma sintonia fina para seleção dos hiperparâmetros por meio da utilização da função GridSearchCV, disponibilizada pela biblioteca *sklearn*.

Esta função faz uso de uma abordagem baseada em busca exaustiva usando cross-validation, por meio da qual define-se um conjunto de valores para cada hiperparâmetro que será testado. O algoritmo treina o modelo para todas as combinações possíveis de valores previamente definidos. No nosso caso específico, foi utilizada a Time Series Cross-Validation, na forma detalhada na seção 3.3.10.

Os modelos baseline definidos nesta seção servirão como referência neste projeto para contextualizar a performance posteriormente, após a aplicação de técnicas para redução de dimensionalidade. Assim, iniciamos a discussão apresentando as figuras 22, 23 e 24, que contemplam os resultados de cada modelo no training set e no test set, respectivamente, para as métricas F1-Measure, Recall e Precision.

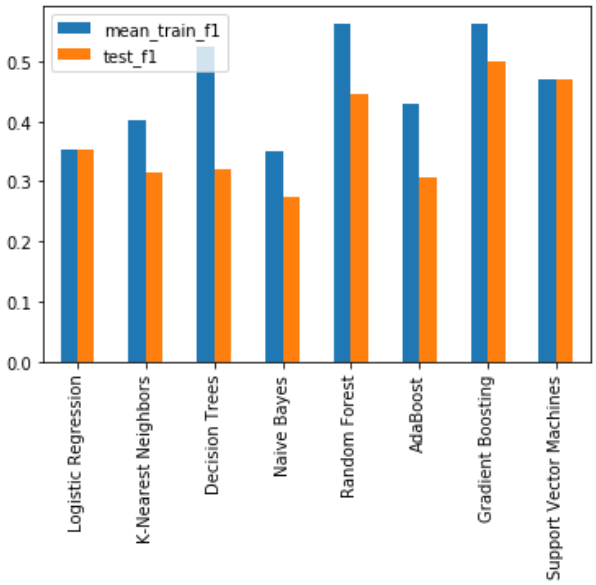

Figura 22. Gráfico contendo valores de F1-Measure no training set e no test set para cada algoritmo

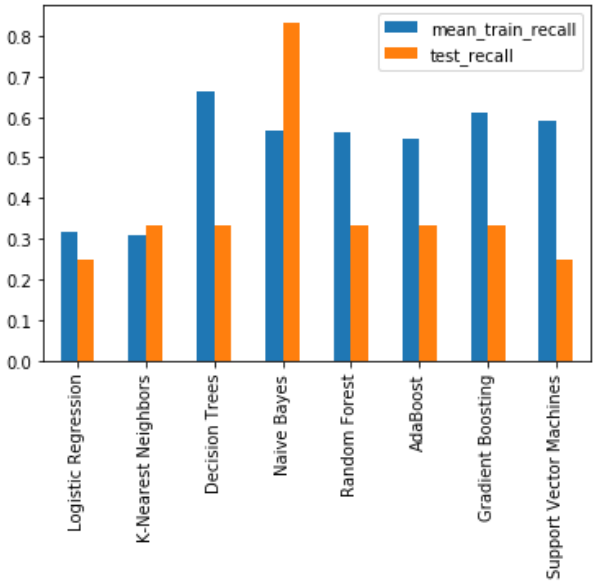

Figura 23. Gráfico contendo valores de Recall no training set e no test set para cada algoritmo

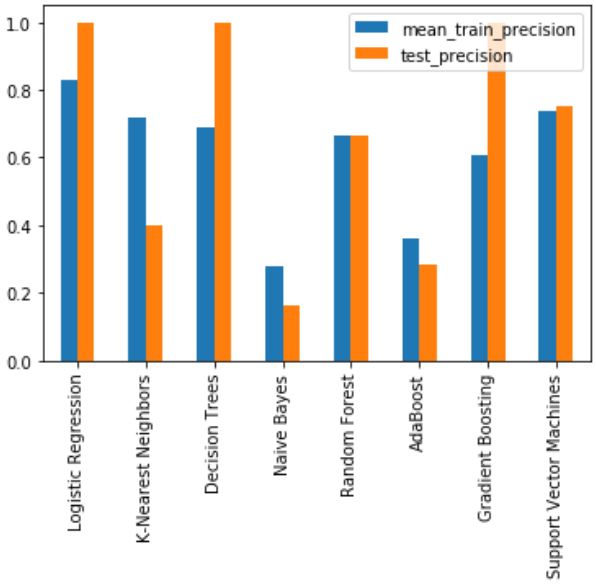

Figura 24. Gráfico contendo valores de Precision no training set e no test set para cada algoritmo

De acordo com a figura 22, para a F1-Measure, pode-se notar que, no training set, modelos mais simples como Logistic Regression, K-Nearest Neighbors e Naive Bayes apresentaram resultados inferiores aos demais modelos, que são considerados mais complexos. Random Forest e Gradient Boosting, dois tipos de ensemble models, obtiveram as melhores performances, o que é de certa forma

esperado e condizente com a teoria, embora seja importante lembrar que são modelos com alta capacidade de memorização das instâncias de treinamento.

No que se refere às instâncias novas, o modelo Gradient Boosting obteve o melhor resultado, apesar de apresentar overfitting, demonstrando um valor para F1-Measure no test set ligeiramente inferior ao do training set. Ainda sob esse aspecto, ressalte-se que o modelo que apresentou o pior overfitting foi a Decision Tree, o que novamente reforça um conceito amplamente difundido na literatura e mencionado na seção 3.3.3 – capacidade de reproduzir de forma excelente (ou perfeita) o comportamento das instâncias de treinamento, apresentando performance inferior (ou muito pobre) ao se deparar com novas instâncias para realizar previsões, generalizando mal o conjunto de dados.

Passando à análise dos resultados informados na figura 23, cabe frisar que novamente os valores mais altos para a recall, no training set, surgiram em regra da aplicação de modelos mais complexos, sendo a Decision Tree o modelo de melhor performance, mas novamente replicando o comportamento demonstrado no treinamento de modelos para aferição da F1-Measure, com baixo poder de generalização. Por outro lado, o modelo mais simples (Naive Bayes) apresentou o maior valor para a recall.

Tal resultado se explica pelo fato de que o algoritmo previu uma quantidade muito alta como instâncias representativas da classe insuficiente. Para tornar mais claro, perceba que o modelo Naive Bayes classificou 61 instâncias na classe 1, enquanto as Decision Trees classificaram apenas 10 instâncias na mesma classe. Logo, neste caso, o resultado mais alto ocorreu em consequência de um baixo valor para a precision.

Por fim, em relação aos resultados disponibilizados na figura 24 (precision), podemos ver que, para o training set, Logistic Regression apresentou a melhor performance, seguido de SVM e K-Nearest Neighbors, nesta exata ordem. Já para o test set, 3 modelos alcançaram a precision máxima (Logistic Regression, Decision Trees e Gradient Boosting). Analogamente ao ocorrido para a recall, a precision perfeita nestes 3 casos ocorre de forma concomitante com a penalização da recall para os respectivos modelos.

5.3. PCA

Após obtermos os nossos baselines, partimos para os testes aplicando a primeira técnica de redução de dimensionalidade. Neste caso, fizemos uso da Análise de Componentes Principais (PCA), descrita anteriormente na seção 3.3.12.

Como já havíamos adiantado na descrição da técnica, a implementação da Análise de Componentes Principais em Python se deu pela utilização da biblioteca sklearn. A função disponibilizada necessita da definição de um parâmetro denominado n_components, que precisa ser calibrado.

Neste contexto, há 2 formas de especificação do parâmetro. A primeira seria a utilização de um valor situado entre 0 e 1, que representa a variação explicada pelo modelo. A segunda seria o próprio número de componentes a ser retornado pelo algoritmo. Pela dificuldade de calibrarmos o número de componentes, a primeira opção se mostra mais fácil.

Neste trabalho, optamos pela manutenção de 90% da variação original dos dados, o que resultou em modelo final com 17 componentes principais. Levando-se em consideração que o dataset original contava com 71 variáveis, podemos perceber que houve uma redução significativa da quantidade de variáveis (54 variáveis a menos), mesmo com uma perda pequena da variação original (10 pontos percentuais).

Após a aplicação de PCA, foi utilizado um pipeline análogo àquele empregado no baseline para sintonia fina dos hiperparâmetros dos algoritmos testados. Para darmos sequência à análise dos resultados, apresentamos as figuras 25, 26 e 27, que materializam os resultados de cada modelo no training set e no test set, respectivamente, para as métricas F1-Measure, Recall e Precision.

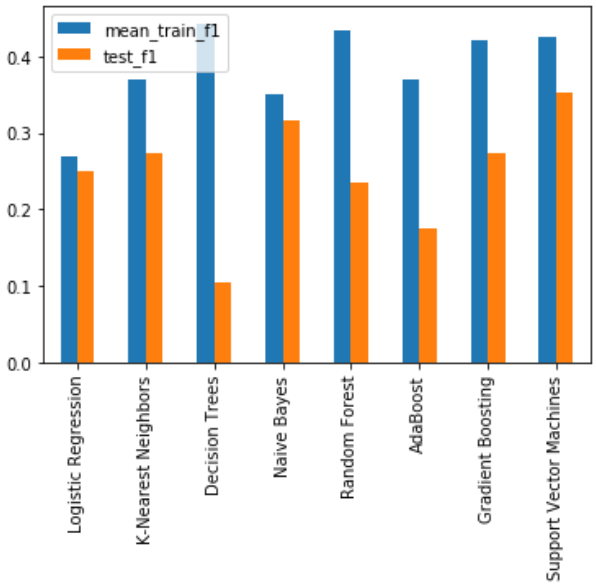

Figura 25. Gráfico contendo valores de F1-Measure no training set e no test set para cada algoritmo, após a utilização da técnica de PCA

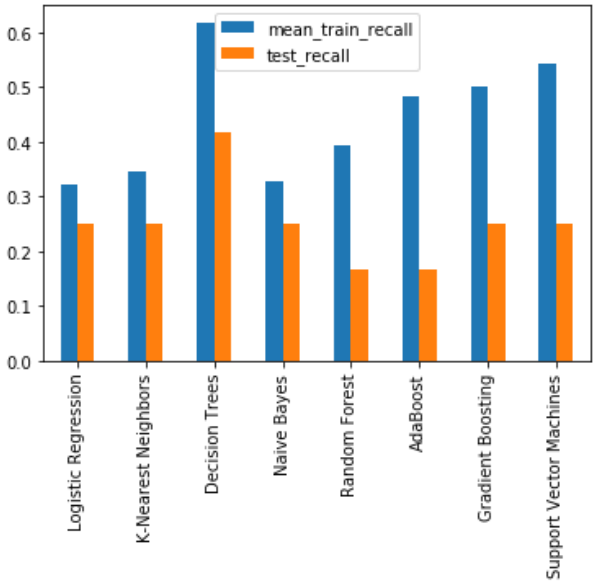

Figura 26. Gráfico contendo valores de Recall no training set e no test set para cada algoritmo, após a utilização da técnica de PCA

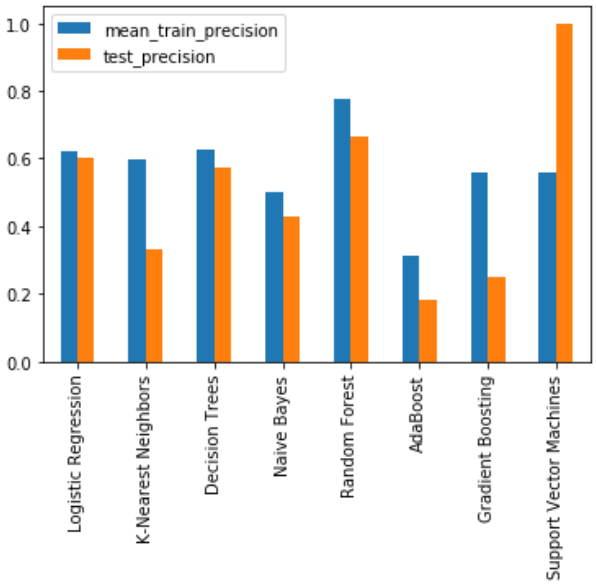

Figura 27. Gráfico contendo valores de Precision no training set e no test set para cada algoritmo, após a utilização da técnica de PCA

A figura 25 apresenta os resultados para a F1-Measure. Especificamente no que se refere às instâncias de treinamento, pode-se notar que as árvores de decisão, os ensemble models Random Forest e Gradient Boosting e o modelo SVM (todos modelos de maior complexidade) apresentaram melhor desempenho.

Todavia, com exceção do SVM, os modelos de melhor performance no training set apresentaram overfitting, não conseguindo apresentar o mesmo desempenho quando avaliado no test set. Perceba, do mesmo gráfico, que os modelos com os valores mais altos para a F1-Measure avaliados no test set foram Support Vector Machines, Naive Bayes e KNN.

Passando à análise dos resultados para a recall (figura 26), observa-se, para o conjunto de treinamento, um padrão muito similar ao apresentado para a F1: modelos mais complexos com performances superiores. As Decision Trees, em conjunto com Support Vector Machines, Gradient Boosting e AdaBoost, apresentaram os melhores resultados.

Entretanto, mais uma vez, a capacidade de generalização de alguns destes modelos se mostrou comprometida. As árvores de decisão apresentaram a melhor performance no conjunto de teste, mesmo com o alto valor para a medida de overfitting. Já os algoritmos de Logistic Regression e KNN apresentaram os mesmos valores para a recall do que Gradient Boosting, ainda que tenham performado muito pior no conjunto de teste.

Por fim, considerando a figura 27, observaremos os resultados para a precision. Primeiramente, para o training set, é possível notar que o melhor desempenho foi obtido pelo algoritmo Random Forest, seguido pelas Decision Trees e Logistic Regression. Já quando analisamos o test set, é surpreendente ver que SVM obteve o valor 1 para a métrica em análise, o que significa dizer que, das instâncias classificadas pelo algoritmo na classe positiva, todas foram classificadas corretamente.

Contudo, ao verificarmos com mais detalhes os resultados, compreendemos que o valor alto da precision se deveu principalmente ao fato de apenas 1 instância ter sido classificada na classe insuficiente (de um total de 12 casos de insuficiência). Ou seja, o algoritmo foi bastante conservador na previsão e classificou quase todas as instâncias na classe suficiente (possivelmente em decorrência do desbalanceamento dos dados).

5.4. RFECV

Em prosseguimento à análise dos resultados, a presente seção verifica os efeitos da aplicação da técnica descrita na seção 3.3.11. Em síntese, o algoritmo se inicia com o treinamento sendo realizado pela utilização de todas as variáveis independentes disponíveis. A cada iteração, a variável de menor importância é descartada e o modelo é retreinado até atingir o número mínimo de variáveis especificado. Ao final, é selecionado o subconjunto de variáveis que resultou o maior valor para a métrica selecionada.

É importante ressaltar que o algoritmo RFECV foi treinado em 3 oportunidades diferentes para alcançarmos o subconjunto de variáveis ótimo para cada uma das métricas testadas (F1, recall e precision). Além disso, foram testados 2 modelos como estimadores base: Logistic Regression e Gradient Boosting Classifier.

Após a aplicação de RFECV, foi utilizado um pipeline análogo àquele empregado no baseline para sintonia fina dos hiperparâmetros dos algoritmos testados. Em algumas poucas ocasiões o melhor resultado foi gerado pelo subconjunto de variáveis advindo da aplicação da RFECV com Logistic Regression, enquanto na maioria das ocasiões o melhor resultado foi obtido a partir da aplicação de RFECV com Gradient Boosting.

Para darmos sequência à análise dos resultados, apresentamos as figuras 28, 29 e 30, que materializam os resultados de cada modelo no training set e no test set, respectivamente, para as métricas F1-Measure, Recall e Precision.

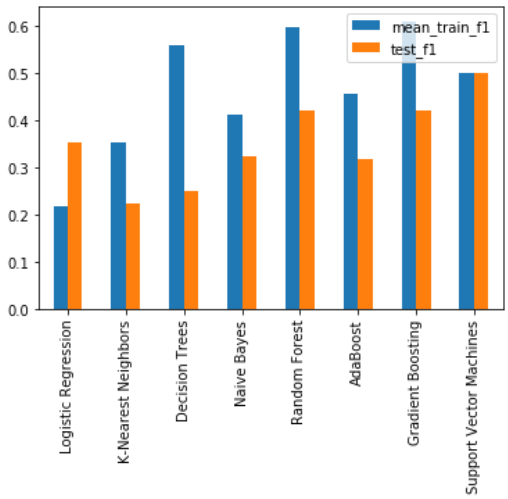

Figura 28. Gráfico contendo valores de F1-Measure no training set e no test set para cada algoritmo, após a utilização da técnica de RFECV

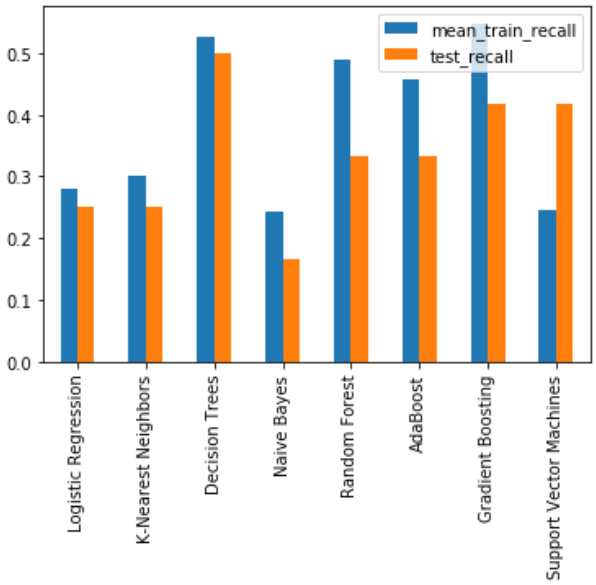

Figura 29. Gráfico contendo valores de Recall no training set e no test set para cada algoritmo, após a utilização da técnica de RFECV

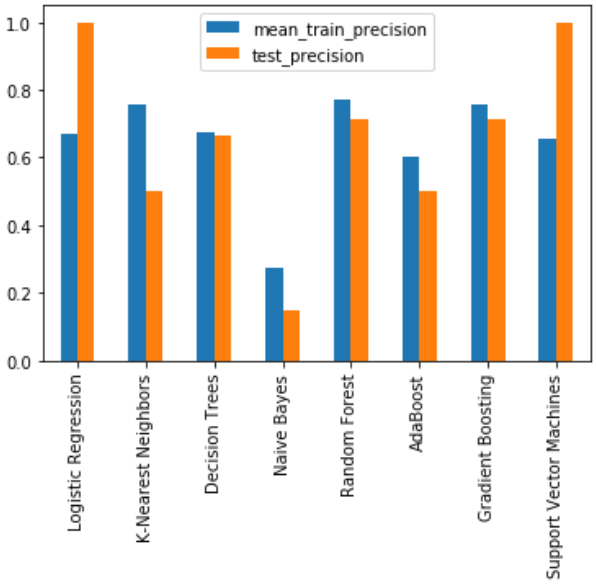

Figura 30. Gráfico contendo valores de Precision no training set e no test set para cada algoritmo, após a utilização da técnica de RFECV

A figura 28, que apresenta os resultados para a F1-Measure, demonstra que, no training set, o algoritmo com a melhor performance foi o Gradient Boosting, levemente superior ao Random Forest. Além destes, as Decision Trees também apresentaram resultados significativos. Entretanto, como tem sido recorrente nos resultados apresentados, tais algoritmos tem sofrido com a ocorrência de overfitting. Seu desempenho no test set, em geral, se mostrou bem abaixo da média coletada nos splits do training set. Assim, a melhor performance no training set se deu para SVM, que demonstrou performance idêntica em ambos os conjuntos.

Da análise da figura 29 (recall), pode-se ver que Gradient Boosting, Decision Trees e Random Forest apresentaram as melhores performances no training set. Como já havíamos mencionado e percebido nas análises anteriores, são modelos mais complexos e que tendem a apresentar overfitting. Já no test set, a melhor performance foi atingida pelas Decision Trees, que, desta vez, apresentaram desempenho bem similar ao obtido no conjunto de treinamento. A pior performance, em ambos os conjuntos, foi demonstrada por Naive Bayes.

Por fim, considerando as informações obtidas da figura 30 para a precision, quase a totalidade dos algoritmos – com exceção de Naive Bayes – apresentaram resultados semelhantes para o conjunto de treinamento, variando no intervalo de 60% a 70%. Ao focarmos a nossa análise para o test set, nota-se que Logistic Regression e SVM apresentaram o valor máximo para a precision.

Neste ponto, aproveitamos para recordar a análise da seção anterior, em que SVM também obteve precision igual a 1. Naquele caso, o classificador somente previu uma instância na classe positiva. Aqui, fato idêntico ocorreu para o modelo de Logistic Regression. Porém, aqui SVM foi menos conservador e classificou 4 instâncias na classe positiva (de um total de 12 casos de insuficiência), todas elas perfeitamente previstas.

5.5. SMOTE E OUTRAS TÉCNICAS PARA LIDAR COM DATASETS DESBALANCEADOS

Nesta seção, apresentaremos os resultados mais importantes e com um nível de detalhamento maior. Explico: ao contrário das seções anteriores, que ofereciam uma visão geral dos modelos e resultados para todas as métricas comentadas na seção 5.1, aqui a otimização dos hiperparâmetros via GridSearch foi realizada exclusivamente utilizando a recall como estratégia de avaliação de performance. A partir daí, comentaremos, para cada modelo, sua matriz de confusão e, em caso de valores idênticos para a recall, partiremos para uma análise dos resultados para as demais métricas.

O motivo para isso é que, embora, a depender do contexto, cada uma das métricas pode-se mostrar apropriada, sob o ponto de vista de avaliação regulatória, a recall é a métrica que melhor se adapta às necessidades do órgão regulador, dado todo o contexto apresentado nesse trabalho.

Uma recall de valor alto nos garante que poucas – ou nenhuma – empresas em condição de insuficiência estarão de fora do olhar da supervisão, o que, em última análise, permite uma ação proativa e evita que consumidores deixem de receber suas indenizações. Ademais, avaliar cada uma das métricas requer um esforço computacional muito maior, uma vez que acaba por ser necessário calibrar os modelos e seus hiperparâmetros para cada métrica em apreço.

Em sequência, cabe lembrar que um dos problemas mais comentados no decorrer desta tese está relacionado ao fato de que o dataset utilizado é bastante desbalanceado, isto é, composto por muito mais casos de empresas em situação de suficiência do que de insuficiência. Isso acarreta dificuldade em treinar modelos que consigam lidar com o desequilíbrio citado e terminam por obter uma performance muito ruim na classe minoritária.

Desta forma, fizemos uso de 3 técnicas (ou combinações entre elas) para dados desbalanceados, que objetivam reequilibrar o número de instâncias nas 2 classes existentes: RandomUnderSampler, RandomOverSampler e SMOTE. Todas estas técnicas foram apropriadamente descritas na seção

3.3.13. Os resultados a seguir demonstrados deixarão clara eficiência da aplicação das técnicas e a melhoria de performance dos classificadores.

O pipeline que gerou o melhor resultado consistiu inicialmente na utilização da técnica RFECV para redução da dimensionalidade do conjunto de dados. Neste ponto, assim como na seção anterior, permitimos ao algoritmo ser testado com 2 diferentes estimadores de base: Logistic Regression e Gradient Boosting Classifier.

Após a utilização de RFECV, ao subconjunto de dados resultante foi aplicado SMOTE e, em seguida, RandomUnderSampler. Os melhores resultados foram alcançados com a estratégia de reamostrar os dados para obtenção de uma taxa de número de instâncias na classe minoritária sobre o número de instâncias na classe majoritária de 30% em SMOTE e 90% ou 100% em RandomUnderSampler, dependendo do algoritmo a ser treinado. Além disso, o hiperparâmetro k_neighbors em SMOTE foi fixado em 5, o que significa dizer que foram utilizados 5 vizinhos no processo de construção das amostras sintéticas.

A figura 31 consolida os resultados apresentados por cada um dos algoritmos, no training set e no test set.

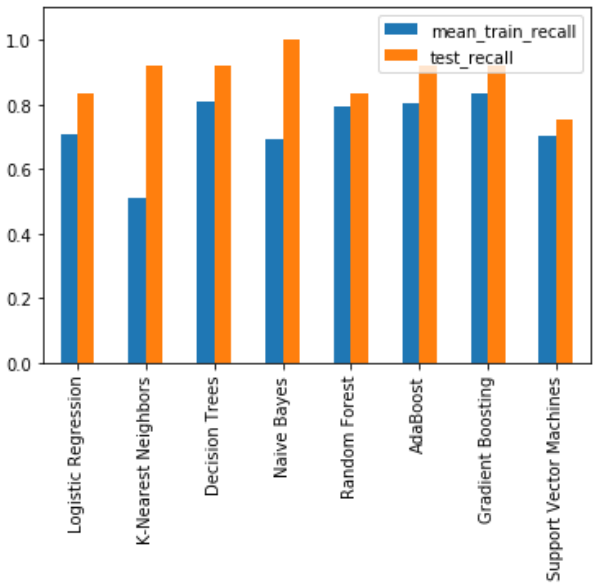

Figura 31. Gráfico contendo valores de Recall no training set e no test set para cada algoritmo, após a utilização das técnicas de RFECV, SMOTE e RandomUnderSampler

A primeira questão a se extrair da observação da figura 31 é que todos os algoritmos apresentaram performance superior no test set em relação ao training set, fato este bastante atípico em modelos de previsão. No intuito de investigar melhor este acontecimento, verificamos os resultados abertos em cada um dos splits (recordando que o modelo de Time Series Cross-Validation, definido na seção 3.3.10, aplicado ao data set gerou 4 splits) e ficou demonstrado que, em geral, os modelos apresentaram desempenho mais baixo em um dos splits, o que contribuiu para baixar o valor médio da recall no training set.

Considerando os resultados para o training set, o melhor desempenho foi obtido pelo modelo Gradient Boosting Classifier com uma recall média de 0,83. No que se refere à performance no test set, o maior valor foi atingido pelo algoritmo Naive Bayes (1,00), seguido por quatro outros algoritmos (KNN, Decision Trees, AdaBoost e Gradient Boosting) que alcançaram o valor de 0,92. A pior performance no training set foi obtida por KNN com um valor equivalente a 0,51 para a recall.

Todos estes valores serão detalhados a partir de agora para que possamos entender qual modelo apresentou o resultado mais consistente. Para tal, faremos uso da matriz de confusão aplicada ao test set, recurso que nos permitirá entender melhor os resultados.

Iniciando a análise, Logistic Regression alcançou uma recall de 0,83 no test set. A seguir, apresentamos a tabela 7 com a matriz de confusão do referido modelo.

		Predito	
		Negativa	Positiva
Real	Negativa	76	37
	Positiva	2	10

Tabela 7. Matriz de confusão dos resultados de Logistic Regression no test set

Claramente, vemos que o modelo foi capaz de prever 10 das 12 empresas insuficientes, mas, para tanto, classificou 47 casos na classe positiva, penalizando a precision (0,21) e, consequentemente, a F1-Measure (0,34).

Prosseguindo na análise, K-Nearest Neighbors alcançou uma recall de 0,92 no test set, mesmo tendo obtido o pior desempenho no training set (0,51). A seguir, apresentamos a tabela 8 com a matriz de confusão do referido modelo.

		Predito	
		Negativa	Positiva
Real	Negativa	103	10
	Positiva	1	11

Tabela 8. Matriz de confusão dos resultados de KNN no test set

Neste caso, é possível notar que o resultado obtido foi impressionante. Das 12 situações de insuficiência no ano de análise, apenas 1 não foi capturada pelo modelo. Além disso, o modelo classificou apenas 21 companhias na classe positiva, alcançando os maiores valores para a precision (0,52) e F1-Measure (0,67).

Em continuidade, as Decision Trees atingiram uma recall de 0,92 no test set. A seguir, apresentamos a tabela 9 com a matriz de confusão do referido modelo.

		Predito	
		Negativa	Positiva
Real	Negativa	91	22
	Positiva	1	11

Tabela 9. Matriz de confusão dos resultados de Decision Trees no test set

À semelhança do ocorrido com KNN, apenas uma situação de insuficiência não foi capturada pelo modelo. Contudo, o modelo alcançou valores para a precision (0,33) e F1-Measure (0,49) inferiores.

Passando agora ao modelo Naive Bayes, o resultado demonstrado no test set é inicialmente surpreendente, mas uma análise mais minuciosa permite desmistificar a situação. A recall equivalente a 1 é obtida ao custo da penalização das demais métricas, com grande parte das instâncias sendo classificadas na classe positiva. A seguir, apresentamos a tabela 10 com a matriz de confusão do referido modelo.

		Predito	
		Negativa	Positiva
Real	Negativa	23	90
	Positiva	0	12

Tabela 10. Matriz de confusão dos resultados de Naive Bayes no test set

Em termos práticos, seria praticamente inviável intensificar a fiscalização em 102 empresas pelo fato destas terem sido classificadas como insuficientes.

Em sequência, Random Forest atingiu uma recall de 0,83 no test set. A seguir, apresentamos a tabela 11 com a matriz de confusão do referido modelo.

		Predito	
		Negativa	Positiva
Real	Negativa	97	16
	Positiva	2	10

Tabela 11. Matriz de confusão dos resultados de Random Forest no test set

Neste caso, podemos perceber que o modelo foi capaz de prever 10 dos 12 casos de insuficiência e, para tal, classificou 26 instâncias na classe positiva, alcançando uma precision de 0,38 e uma F1-Measure de 0,53.

Considerando um outro ensemble model, AdaBoost, este obteve uma performance de 0,92 na recall. A seguir, apresentamos a tabela 12 com a matriz de confusão do referido modelo.

		Predito	
		Negativa	Positiva
Real	Negativa	96	17
	Positiva	1	11

Tabela 12. Matriz de confusão dos resultados de AbaBoost no test set

De forma análoga a outro modelos analisados, AdaBoost alcançou desempenho muito significativo ao deixar de classificar apenas 1 companhia insuficiente nesta classe. Seus valores de precision e F1-Measure também foram expressivos, equivalentes a 0,39 e 0,55, respectivamente.

O próximo ensemble model em análise é Gradient Boosting, que apresentou excelente performance, com recall equivalente a 0,92 no test set. A seguir, apresentamos a tabela 13 com a matriz de confusão do referido modelo.

		Predito	
		Negativa	Positiva
Real	Negativa	99	14
	Positiva	1	11

Tabela 13. Matriz de confusão dos resultados de Gradient Boosting no test set

A diferença principal de Gradient Boosting para AbaBoost é que aquele foi capaz de obter o mesmo número de instâncias corretamente classificadas na classe positiva com uma precision (0,44) e F1-Measure (0,59) superiores.

Por fim, apresentamos os resultados para SVM, que demonstrou uma recall de 0,75 no test set. A seguir, apresentamos a tabela 14 com a matriz de confusão do referido modelo.

		Predito	
		Negativa	Positiva
Real	Negativa	84	29
	Positiva	3	9

Tabela 14. Matriz de confusão dos resultados de SVM no test set

Dos modelos em análise, SVM foi o que obteve pior desempenho, embora consistente com os resultados apresentados no training set. Seu resultado para a F1-Measure (0,36) só se mostrou superior ao modelo Naive Bayes.

Por todos os resultados expostos, é possível notar que a aplicação de técnicas de rebalanceamento do data set promoveu melhorias significativas nos resultados para a recall. Sob a ótica de um órgão regulador, trata-se de um resultado excelente conseguir prever com antecedência 11 das 12 companhias que entrarão em situação de insuficiência de capital.

A escolha do melhor modelo, neste caso, envolve a combinação dos melhores resultados no training set e test set. Apesar de ter obtido resultados muito bons no test set, KNN obteve a pior performance no training set. Dentro desta perspectiva, Gradient Boosting apresentou o melhor resultado combinado, dados os seguintes pontos de avaliação:

1. Maior valor de recall média quando avaliado nos 4 splits do training set: 0,83;
2. Maior valor de recall quando avaliado no test set (desconsiderando Naive Bayes pelos motivos relatados nesta seção): 0,92

3. Dos modelos que obtiveram recall equivalente a 0,92 no test set, só apresentou F1-Measure inferior ao KNN: 0,59

5.6. TPOT CLASSIFIER

O referido modelo, que realiza uma busca em um espaço com diferentes pipelines, apesar de promissor, não apresentou resultados tão consistentes quanto o esperado. Para o processo de otimização, optamos pela utilização da F1-Measure, tendo em vista que a recall costumava levar a modelos que penalizavam muito as demais métricas, classificando a maior parte das instâncias na classe positiva.

Realizamos 3 diferentes tentativas de treinamento do modelo. Na primeira, definimos apenas 10 gerações e cada geração com uma população de tamanho 10. Na segunda, utilizamos 10 gerações com tamanho da população equivalente a 20. Por fim, estabelecemos uma população de tamanho 20 e permitimos ao modelo ser treinado livremente durante 1 hora, sem definição exata do número de gerações. A tabela a seguir apresenta os resultados para cada uma das tentativas.

	GENERATIONS: 10 POPULATION_SIZE: 10	GENERATIONS: 10 POPULATION_SIZE: 20	MAX_TIME_MINS: 60 POPULATION_SIZE: 20
CV SCORE (LAST GENERATION)	0,60	0,61	0,64
TEST SET SCORE	0,53	0,36	0,30

Tabela 15. Valores da F1-Measure para 3 tentativas de otimização do modelo TPOT Classifier

Os resultados demonstraram que os modelos tenderam a aumentar a ocorrência de overfitting à medida que mais gerações foram treinadas. De todas as tentativas, a primeira apresentou bons resultados e a matriz de confusão apresentada na tabela 16 a seguir nos fornece um cenário mais detalhado.

		Predito	
		Negativa	Positiva
Real	Negativa	103	10
	Positiva	4	8

Tabela 16. Matriz de confusão dos resultados de TPOT Classifier no test set

6. CONCLUSÕES

Neste trabalho, propusemos o desafio de utilizar modelos de machine learning para realizar previsão de insolvência de sociedades seguradoras, usando dados do mercado brasileiro no intervalo de 2011 a 2018. Trata-se de um trabalho inovador e de enorme relevância para a Susep, uma vez que o Decreto-Lei nº 73/66, que dispõe sobre o Sistema Nacional de Seguros Privados, determina que a política de seguros deve se dar no interesse dos segurados e beneficiários dos contratos de seguros, bem como objetiva preservar a liquidez e a solvência das sociedades seguradoras.

A sinalização antecipada da condição de insuficiência de capital de uma companhia seguradora permite aos supervisores uma atuação tempestiva e garante condições para reverter um quadro de crise financeira. Um sistema que alerte com antecedência as empresas que passam por uma situação mais delicada permite ainda uma melhor alocação de recursos para as atividades de fiscalização, tem em conta principalmente o fato de que o quantitativo de servidores destinados a atividades fiscalizatórias é limitado e a necessidade de priorização é de observância mandatória em todos os anos.

Neste contexto, o projeto consistiu em transformar o problema real em um problema de classificação binária. Para tal, construímos um indicador que informa se o Patrimônio Líquido Ajustado (PLA) de uma empresa é menor que o seu Capital Mínimo Requerido (CMR). Neste caso, fica configurada uma situação de insuficiência de capital e a nossa variável dependente assume o valor 1. Caso contrário, uma situação de suficiência de capital é caracterizada e variável assume valor 0.

Uma revisão das principais abordagens utilizadas na literatura foi realizada e discutida. A partir desta revisão, buscamos inspiração tanto em modelos quanto em variáveis comumente utilizadas para tentar explicar situações de comprometimento da saúde econômico-financeira das empresas.

A metodologia CRISP-DM e suas 6 etapas serviram de base para todo o processo de modelagem. Ademais, realizou-se uma descrição pormenorizada de todos os modelos utilizados, com apresentação do referencial teórico e discussão dos hiperparâmetros mais importantes para a sintonia fina dos algoritmos em suas implementações disponíveis em Python.

Dessa forma, a modelagem foi realizada lançando mão de 8 algoritmos de uso consagrado. Ainda, considerando a existência de uma componente temporal nos dados, vimo-nos obrigados a realizar cross-validation de forma diferente da utilizada de forma convencional. Assim, utilizamos a Time Series Cross Validation. Em sequência, no que se refere à grande quantidade de variáveis disponíveis, procedemos à utilização de técnicas para redução de dimensionalidade, notadamente PCA e RFECV.

Por fim, o maior problema com que tivemos que lidar foi o fato de que os dados são muito desbalanceados, com muito mais instâncias pertencentes à classe negativa do que à classe positiva, o que dificulta a aprendizagem dos algoritmos. Para mitigar o problema, implementamos algumas técnicas que permitem rebalancear os dados e equilibrar o número de instâncias em cada classe.

Inicialmente, as 3 principais métricas (recall, precision e F1) foram avaliadas e foram descritos os cenários a que cada uma melhor se adapta. Entretanto, na análise final, os esforços foram concentrados na avaliação dos modelos sob o uso da recall, pela maior aderência aos interesses do órgão regulador.

Em conclusão, foi possível notar a eficiência dos algoritmos em realizar previsões para o problema descrito, especialmente após a utilização conjugada de RFECV – para redução do número de variáveis – e SMOTE e RandomUnderSampler – para equilibrar o número de observações pertencentes a cada classe.

O algoritmo Gradient Boosting Classifier, espécie do gênero ensemble model, foi quem obteve os resultados mais consistentes numa avaliação conjunta da recall nos conjuntos de dados de treinamento e de teste. Este foi capaz de identificar 11 das 12 situações de insuficiência, sem prejudicar as demais métricas.

6.1. LIMITAÇÕES E RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Todos o processo de modelagem descrito neste trabalho foi possível devido à disponibilização dos dados pela Susep. Mais ainda, a construção da variável dependente se deu a partir das regras disponíveis nos normativos para Patrimônio Líquido Ajustado (PLA) e Capital Mínimo Requerido (CMR). Embora sejam conceitos fundamentais na modelagem, no intervalo em que os dados foram disponibilizados, houve mudança nas fórmulas de cálculo do PLA e do CMR, seja pela dedução de novas contas contábeis do PL, seja pela introdução de novas parcelas de capital.

Esta alteração gera alguma fragilidade e dificulta a aprendizagem pelos algoritmos por ser um fator complexo de ser inserido ou desenhado por meio de uma variável explicativa. Além disso, no mesmo período, houve mudança nos formatos de fiscalização, o que também tem impacto na variável dependente, pois anos com fiscalização mais severa implicam mais ajustes contábeis.

É desejável que este trabalho seja atualizado no futuro para contemplar um período maior com todas as parcelas de capital já implementadas e com o mercado supervisionado mais maduro em avaliações prudenciais baseadas no pilar I (requisitos quantitativos associados a capital de solvência e capital mínimo) do projeto Solvência II.

7. BIBLIOGRAFIA

Albon, C. (2018). Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning. 1st Edition. O'Reilly Media, Inc.

Azevedo, A. I. R. L., & Santos, M. F. (2008). KDD, SEMMA and CRISP-DM: a parallel overview. IADS-DM.

Bergmeir, C., & Benítez, J. M. (2012). On the use of cross-validation for time series predictor evaluation. Information Sciences. Vol. 191, pp. 192-213.

Bezerra, F. A., & Corrar, L. J. (2006). Utilização da análise fatorial na identificação dos principais indicadores para avaliação do desempenho financeiro: uma aplicação nas empresas de seguros. Revista Contabilidade & Finanças - USP. No. 42., pp. 50 - 62.

Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

Brockett, P. L., Cooper, W. W., Golden, L. L., & Pitaktong, U. (1994). A Neural Network Method for Obtaining an Early Warning of Insurer Insolvency. The Journal of Risk and Insurance, Vol. 61, No. 3., pp. 402-424.

Brockett, P. L., Golden, L. L., Jang, J., & Yang, C. (2006). A comparison of neural network, statistical methods, and variable choice for life insurers' financial distress prediction. The Journal of Risk and Insurance, Vol. 73, No. 3, 397-419

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., & Wirth, R. (2000). CRISP-DM 1.0: step-by-step data mining guide. Retrieved May 26, 2021, from https://www.kde.cs.uni-kassel.de/wp-content/uploads/lehre/ws2012-13/kdd/files/CRISPWP-0800.pdf

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P..(2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research 16, pp. 321–357.

Constantino, F. F. S. (2020). Análise da insolvência de seguradoras brasileiras sob a ótica da regulação. Tese (Doutorado em Ciências Contábeis) - Universidade Federal do Rio de Janeiro.

Cummins, J. D., Rubio-Misas, M., & Vencappac, D. (2017). Competition, efficiency and soundness in European life insurance markets. Journal of Financial Stability. Vol. 28, pp. 66-78.

Danieli, L., & Jakubik, P. Early warning system for the european insurance sector. Retrieved May 20, 2021, from EIOPA: eiopa_fsr_december_2018_thematic_article.pdf (europa.eu)

EIOPA (2009). Directive 138/2009/EC (Solvency II Directive). Retrieved from | Eiopa (europa.eu)

Fernández, A., García, S., Galar, M., Prati, R. C., Krawczyk, B., & Herrera, F. (2018). Learning from Imbalanced Data Sets. Springer.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. The Annals of Statistics, Vol. 29, No. 5, pp 1189–1232.

Friedman, J., Hastie, T., & Tibshirani, R. (2009). The Elements of Statistical Learning: Data Mining, Inference and Prediction. Springer.

Géron, A. (2019). Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow - Concepts, Tools, and Techniques to Build Intelligent Systems. 2nd Edition. O'Reilly Media, Inc.

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. Journal of machine learning research 3, pp. 1157-1182.

He, H., & Ma, Y. (2013). Imbalanced Learning: Foundations, Algorithms, and Applications. IEEE Press.

Jadi, D. M. (2015). An empirical analysis of determinants of financial performance of insurance companies in the United Kingdom. Retrieved April 13, 2019, from https://bradscholars.brad.ac.uk/handle/10454/14383

Keith, M. (2017). Model Validation Techniques for Time Series: Data splitting, cross validation, model optimization, and dynamic predictions to validate forecasting models. Retrieved February 5, 2021, from Towards Data Science: https://towardsdatascience.com/model-validation-techniques-for-time-series-3518269bd5b3

Kelleher, J. D., Namee, B. M., & D'Arcy, A. (2015). Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies. The MIT Press.

Kouiroukidis, N., & Evangelidis, G. (2011). The effects of dimensionality curse in high dimensional knn search. 15th Panhellenic Conference on Informatics, pp. 41-45. IEEE.

McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, Numpy, and IPython. 2nd Edition. O'Reilly Media, Inc.

Melo, E. F. L., & Neves, C. R. (2012). Solvência no mercado de seguros e previdência: coletânea de estudos. Funenseg.

Mitchell, T. M. (1997). Machine Learning. McGraw-Hill Science/Engineering/Math.

Müller, A. C., & Guido, S. (2016). Introduction to Machine Learning with Python: A Guide for Data Scientists. 1st Edition. O'Reilly Media, Inc.

Okamura, M., & Lima, G. A. S. F. (2013). Um modelo de previsão de insolvência para as seguradoras brasileiras utilizando a regressão logística. In: 10º Congresso de Iniciação Científica em Contabilidade da USP.

Olson, R. S., Bartley, N., Urbanowicz, R. J., & Moore, J.H. (2016). Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. GECCO '16: Proceedings of the Genetic and Evolutionary Computation Conference 2016, pp. 485–492.

Olson, R. S., Urbanowicz, R. J., Andrews, P. C., Lavender, N. A., Kidd, L. C., & Moore, J.H. (2016). Automating Biomedical Data Science Through Tree-Based Pipeline Optimization. Applications of Evolutionary Computation. EvoApplications 2016. Lecture Notes in Computer Science, Vol. 9597. Springer, Cham.

Pacheco, D. P. A. P. (2018). A influência das variáveis financeiras e macroeconómicas na insolvência das PME's portuguesas. Retrieved May 25, 2021, from https://sigarra.up.pt/fep/en/pub_geral.show_file?pi_doc_id=179908

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. The Journal of Machine Learning research. Vol. 12, pp. 2825-2830.

Piatetsky, G. (2014). CRISP-DM, still the top methodology for analytics, data mining, or data science projects. Retrieved May 27, 2021, from https://www.kdnuggets.com/2014/10/crisp-dm-top-methodology-analytics-data-mining-data-science-projects.html

Raschka, S., & Mirjalili, V. (2019). Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2. Packt Publishing.

Rubio-Misas, M., & Fernández-Moreno, M. (2017). Solvency surveillance and financial crisis: evidence from the Spanish insurance industry. Spanish Journal of Finance and Accounting, 46:3, pp. 272-297, DOI: 10.1080/02102412.2017.1291167

Rustam, Z., & Saragih, G. (2021). Prediction Insolvency of Insurance Companies Using Random Forest. Journal of Physics: Conference Series , Volume 1752.

Schapire, R. E., & Freund, Y. (2012). Boosting: Foundations and Algorithms. The MIT Press.

Shrivastava, S. (2020). Cross Validation in Time Series. Retrieved February 4, 2021, from Medium: https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4

SUSEP (2020). 8º Relatório de Análise e Acompanhamento dos Mercados Supervisionados. Retrieved May 20, 2021, from https://www.gov.br/susep/pt-br/arquivos/arquivos-dados-estatisticos/relatorios-de-analise-e-acompanhamento-dos-mercados-supervisonados/8relat_acomp_mercado_2020.pdf

SUSEP (2021a). Síntese Mensal – fevereiro de 2021. Retrieved May 25, 2021, from https://www.gov.br/susep/pt-br/central-de-conteudos/dados-estatisticos/anos/Sintese022021.pdf

SUSEP (2021b). Resolução CNSP nº 432, de 2021. Retrieved January 15, 2022, from https://www2.susep.gov.br/safe/scripts/bnweb/bnmapi.exe?router=upload/26185

Sutton, C. D. (2005). Classification and regression trees, bagging, and boosting. Handbook of statistics. Vol. *24*, pp. 303-329.

Tian, Y., Yang, W., Lai, G., & Zhao, M. (2019). Predicting non-life insurer's insolvency using non-kernel fuzzy quadratic surface support vector machines. Journal of Industrial and Management Optimization. Vol. 15, No. 2, pp. 985-999.

Wanke, P., & Barros, C. P. (2016). Efficiency drivers in Brazilian insurance: A two-stage DEA meta frontier-data mining approach. Economic Modelling. Vol. 53, pp. 8-22.

Zhou, Z.-H. (2012). Ensemble Methods: Foundations and Algorithms. CRC Press.