DISSERTAÇÃO DE MESTRADO PROFISSIONAL

IDENTIFICAÇÃO AUTOMÁTICA DE CONLUIO EM PREGÕES DO COMPRASNET COM APRENDIZADO DE MÁQUINA

Rodrigo Vilela Fonseca de Souza

Programa de Pós-Graduação Profissional em Engenharia Elétrica

DEPARTAMENTO DE ENGENHARIA ELÉTRICA
FACULDADE DE TECNOLOGIA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

COLLUSION IDENTIFICATION IN COMPRASNET WITH MACHINE LEARNING

IDENTIFICAÇÃO AUTOMÁTICA DE CONLUIO EM PREGÕES DO COMPRASNET COM APRENDIZADO DE MÁQUINA

RODRIGO VILELA FONSECA DE SOUZA

ORIENTADOR: ALEXANDRE SOLON NERY, Ph.D
COORIENTADOR: FÁBIO LÚCIO LOPES DE MENDONÇA, Ph.D

DISSERTAÇÃO DE MESTRADO PROFISSIONAL EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO: PPEE.MP.048
BRASÍLIA/DF, JUNHO - 2023

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO PROFISSIONAL

IDENTIFICAÇÃO AUTOMÁTICA DE CONLUIO EM PREGÕES DO COMPRASNET COM APRENDIZADO DE MÁQUINA

Rodrigo Vilela Fonseca de Souza

Dissertação de Mestrado Profissional submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica

Banca Examinadora

Prof. Fábio Lúcio Lopes de Mendonça, Ph.D, FT/UnB _____
Presidente

Prof. Daniel Alves da Silva, Ph.D, FT/UnB _____
Examinador Interno

Prof. Fabiano Cavalcanti Fernandes, Ph.D, IFB _____
Examinador Externo

FICHA CATALOGRÁFICA

SOUZA, RODRIGO VILELA FONSECA DE
IDENTIFICAÇÃO AUTOMÁTICA DE CONLUIO EM PREGÕES DO COMPRASNET COM APRENDIZADO DE MÁQUINA [Distrito Federal] 2023.
xvi, 71 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2023).
Dissertação de Mestrado Profissional - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Pregão eletrônico	2. Conluio
3. Aprendizado de máquina	4. Cartel
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

SOUZA, R.V.F. (2023). *IDENTIFICAÇÃO AUTOMÁTICA DE CONLUIO EM PREGÕES DO COMPRASNET COM APRENDIZADO DE MÁQUINA* . Dissertação de Mestrado Profissional, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 71 p.

Rodrigo Vilela Fonseca de Souza
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

DEDICATÓRIA

À minha amada família, Ada, Mateus e Sofia, que sacrificaram inúmeros momentos de convívio durante este desafio, mas que também testemunharam que não há conquista sem esforço.

AGRADECIMENTOS

Em cada etapa de uma caminhada, é gratificante relembrar quantas pessoas contribuíram para a sua realização. Dessa forma, agradeço primeiramente aos meus pais, que não mediram esforços para apoiar os filhos sempre que necessário. Agradeço à minha família, pelo incentivo nos momentos de desânimo e pela compreensão nos momentos de ausência. Especialmente, agradeço à minha amada esposa Ada, exemplo de perseverança e persistência, por todo o apoio, paciência e incentivo nessa caminhada que escolhemos trilhar juntos. Também agradeço aos professores de uma vida toda, por serem a luz que ilumina o caminho do conhecimento. Em especial, agradeço ao meu orientador, Prof. Dr. Alexandre Solon Nery, e ao meu coorientador, Prof. Dr. Fábio Lúcio Lopes de Mendonça, pelo tempo dedicado nas orientações. Agradeço aos estimados amigos, Eraldo Luís Rezende Fernandes e Danielle de Menezes Maciel Coelho, pelo auxílio prestado na conclusão deste trabalho. Agradeço também, aos colegas da Controladoria Geral da União, por todo o apoio oferecido durante a realização desta pesquisa. Por fim, sou grato a Deus por iluminar meus caminhos e tornar tudo até aqui possível.

RESUMO

O Governo Federal Brasileiro executa um grande volume de licitações públicas da modalidade pregão por meio do Portal de Compras Comprasnet. No período de 2018 a 2021, aproximadamente R$ 144 bilhões foram licitados com a execução de mais de 122 mil processos desta modalidade. A auditoria destes certames é uma das atribuições da Controladoria Geral da União - CGU, que desenvolveu ferramentas para apoiar esta atividade, envolvendo o processamento de um grande volume de dados. Isso possibilita que os pregões eletrônicos sejam auditados e diversas irregularidades identificadas em tempo de serem sanadas. Entre os anos de 2019 e 2020, após atuações preventivas da CGU, licitações que totalizaram mais de R$ 6,7 bilhões foram revogadas, suspensas ou ajustadas. Entre as irregularidades que a CGU busca combater, a identificação de conluio entre licitantes é de difícil identificação, dado o volume diário de novas licitações e o conjunto de variáveis envolvidas no processo. A Inteligência Artificial aplicada à análise de dados, a partir de algoritmos de aprendizado de máquina, se apresenta como promissora na indicação de conluio entre os participantes de uma licitação. Neste trabalho, foi realizado o estudo de um conjunto de algoritmos de aprendizado de máquina aplicados na identificação de conluio, em 4 cenários distintos, sobre dois *datasets* gerados a partir do Comprasnet e outros *datasets* de conluio publicados. De forma geral, nos melhores cenários, os algoritmos baseados em *ensemble methods* obtiveram uma acurácia maior que 87%. Considerando todas as métricas adotadas no experimento, o algoritmo de melhor desempenho foi o *Extra Trees*, capaz de indicar novos possíveis casos de conluio em itens de pregões realizados no Comprasnet.

ABSTRACT

The Brazilian Federal Government executes a large volume of public procurements through the Comprasnet Procurement Portal, which is a website for electronic auctions available for bidders nationwide and abroad. In the period from 2018 to 2021, approximately R$144 billion bids were applied within Comprasnet, with the execution of more than 122 thousand processes of this modality. The audit of these events is one of the duties of the Federal Comptroller General (Controladoria Geral da União - CGU) agency, which has developed tools to support such audit activities, especially involving a large volume of data processing. Thus, it is possible for electronic trading sessions to be audited in time to identify irregularities and rectify them. Between 2019 e 2020, following CGU preventive actions, around R$ 6.7 billion auctions were revoked, suspended or adjusted. Among the irregularities, collusion is difficult to identify, given the set of variables involved in the process. Artificial Intelligence applied to data analysis, through Machine Learning algorithms, presents itself as a promising method towards the detection of collusion between the auction's participants. In this work, a study of machine learning algorithms was carried out, in 4 different scenarios, on two datasets extracted from Comprasnet and other published collusion datasets. In the best scenarios, ensemble methods algorithms achieved an accuracy greater than 87%. Considering all metrics applied, Extra Trees was the algorithm with the best performance, capable of indicating new possible collusion cases.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

b	Lance
n	Número de lances recebidos
sd	Desvio padrão
t	Item licitado
w	Quantidade licitada

Símbolos Gregos

Σ	Somatório

Subscritos

i	i-ésimo lance do item licitado
$losingbids$	Lances perdedores
max	Máximo
min	Mínimo

Sobrescritos

$-$	Valor médio

Siglas

ALICE	Analisador de Licitações e Editais
AM	Aprendizado de Máquina
CADE	Conselho Administrativo de Defesa Econômica
CEIS	Cadastro Nacional de Empresas Inidôneas e Suspensas
CGU	Controladoria-Geral da União
CNPJ	Cadastro Nacional da Pessoa Jurídica
CPF	Cadastro da Pessoas Físicas
CV	*Coefficient of Variation*
DIFFP	*Difference between the two lowest bids*
ESPECIF	Escore de especificidade
FN	*False Negative*
FP	*False Positive*
FREQ	Escore de frequência
IA	Inteligência Artificial
KSTEST	*Kolmogorov-Smirnov test*
KURT	*Excess Kurtosis*
LANC	Total de lances
MLP	*Multi Layer Perceptron*
OCDE	Organização para a Cooperação e Desenvolvimento Econômico
PIB	Produto Interno Bruto
PTE	*Pre tender cost estimate*
QUANT	Quantidade
RD	*Relative Distance*
SGD	*Stochastic Gradient Descent*
SKEW	*Skewness*
SPD	*Spread*
SSD	*Solid State Drive*
SVC	*Support Vector Classifier*
SVM	*Suport Vector Machines*
TCU	Tribunal de Contas da União
TN	*True Negative*
TP	*True Positive*

1 INTRODUÇÃO

As compras públicas representam uma grande parcela na execução da despesa pública. Nos países da Organização para a Cooperação e Desenvolvimento Econômico (OCDE), em média, estas despesas representam 12% do PIB por ano [2]. No Brasil, o governo federal emprega aproximadamente 5% do PIB na aquisição de bens e serviços. Este percentual pode chegar a 15% do PIB quando se inclui nos cálculos as despesas efetuadas por estados, municípios e estatais, o que equivale a R$ 1,3 trilhão considerando o PIB de 2021 [3, 4]. O governo federal brasileiro, entre os anos de 2018 e 2021, aplicou aproximadamente R$ 370 bilhões dos recursos públicos, por meio da execução de 450 mil processos licitatórios das diversas modalidades existentes. Deste total, R$ 144 bilhões (39% do total) foram aplicados com a modalidade *pregão*, executados em 122 mil processos distintos [5]. O pregão foi instituído no Brasil pela Lei 10.520/2002 e regulamentado, na forma de realização eletrônica, no âmbito da administração pública federal, pelo Decreto 10.024/2019 [6, 7]. A operacionalização dos pregões eletrônicos, com exceções admitidas pelo Decreto, ocorre pelo Portal de Compras do Governo Federal, o Comprasnet [7].

A Controladoria-Geral da União (CGU), conforme estipulado na Lei 13.844/2019, atua na defesa do patrimônio público, no controle interno, na auditoria pública, na prevenção e no combate à corrupção [8]. Assim, auditar as compras públicas está entre as competências da CGU. De forma mais específica, também cabe à CGU, o acompanhamento de procedimentos e processos administrativos em curso em órgãos ou entidades da administração pública federal. Bem como, a realização de inspeções e avocação de procedimentos e processos em curso na administração pública federal, para exame de sua regularidade, e proposição de providências ou correção de falhas [8]. Neste sentido, a CGU, na busca de respostas tempestivas do exercício de suas atribuições frente aos processos licitatórios publicados, desenvolveu uma ferramenta de análise de editais e licitações [9]. O Analisador de Licitações e Editais, batizado de ALICE e idealizado pela CGU em 2014, é um algoritmo autônomo que inspeciona diariamente as licitações publicadas no Comprasnet, assim como esquematizado na Figura 1.1 [10]. A ferramenta realiza o download das novas licitações (editais e anexos), determina o valor estimado, classifica as licitações por tema, aplica trilhas de auditoria para detecção de possíveis irregularidades e, por fim, envia alerta aos auditores para avaliação das licitações selecionadas.

Os alertas são gerados pela execução de trilhas de auditoria implementadas para identificar irregularidades ou suspeitas das mais diversas naturezas. Existem trilhas baseadas na estratégia de cruzamento de bases de dados do governo federal com os dados da licitação, como a identificação de empresas vencedoras de licitação, mas que possuem impedimento legal de contratação com administração pública. Outras trilhas se baseiam na identificação de termos na elaboração dos editais que indiquem um possível direcionamento ou restrição de competitividade. Algumas se apoiam em modelos de Inteligência Artificial (IA) para classificação do tema da licitação, como identificação de contratação de serviço de tecnologia da informação, compra de medicamentos, entre outros. Existem trilhas que simplesmente geram alerta de licitação de alto risco baseadas no valor estimado [12].

A partir dos alertas gerados pela ferramenta ALICE, os auditores da CGU executam auditorias preventivas direcionadas, selecionadas em um universo médio diário de 250 novas licitações. Somente em

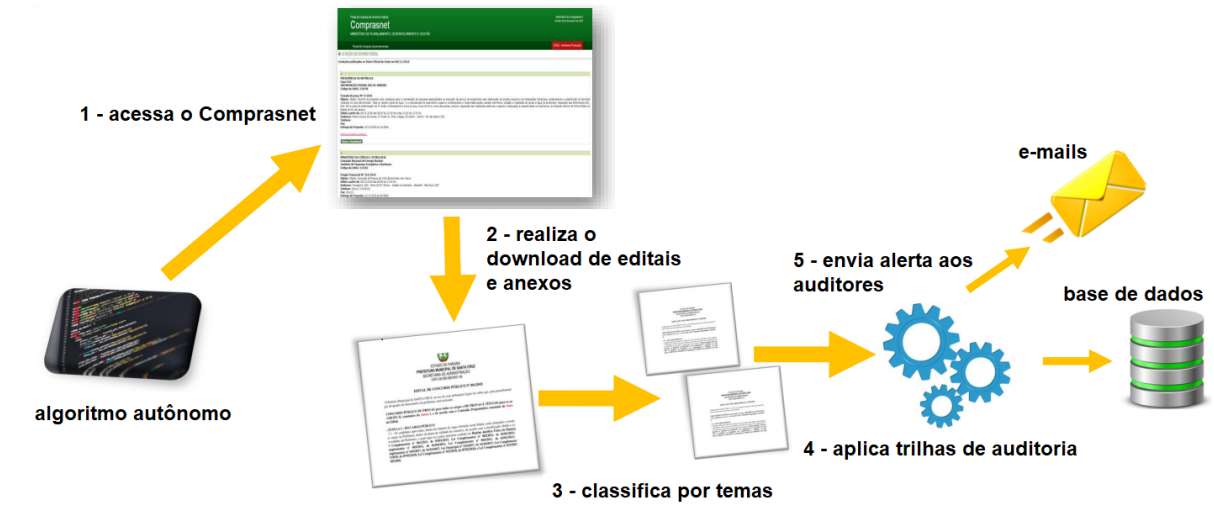

Figura 1.1: Fluxo diário de processamento e inspeção da ALICE sobre as licitações publicadas no Comprasnet. Adaptado de [11].

2019, os alertas gerados resultaram em um total de R$ 4,1 bilhões em licitações que foram auditadas preventivamente e demandaram ações da Controladoria [11]. Em 2020, os alertas gerados totalizaram um valor estimado de R$ 2,69 bilhões em licitações analisadas preventivamente [9]. Especificamente sobre as análises de licitações da modalidade pregão eletrônico pela ALICE, mesmo com as constantes melhorias promovidas pela CGU, ainda existem verificações que podem ser automatizadas pelo desenvolvimento de novas trilhas. Um possível avanço seria a indicação da provável prática de conluio entre as empresas que disputaram um item do certame analisado [12].

Diante deste cenário, pode-se dizer que um método que indique a possibilidade de conluio entre os participantes de licitações, apoiaria significativamente as instituições na execução dos processos de compra, mitigando os efeitos adversos, econômicos e sociais, desta prática.

1.1 MOTIVAÇÃO

A OCDE publicou, em um de seus relatórios, uma estimativa de sobrepreço entre 10 e 20% gerado pela atuação de cartéis em determinadas áreas, se comparado ao preço em mercados competitivos [13]. Apesar desta estimativa se basear em uma pesquisa não científica, a simples multiplicação do sobrepreço identificado pela quantidade comercializada pelo cartel, disponibiliza uma dimensão inicial do prejuízo gerado. Em alguns casos, o sobrepreço pode chegar a 50%, causando perdas anuais de centenas de bilhões de reais aos consumidores. Dois casos que podemos citar no Brasil são a Operação Lava Jato, que identificou prejuízo de bilhões de reais [14], e o Cartel de Portas de Segurança Giratórias, onde a colusão gerou sobrepreço de 25% [15].

Em contextos correlatos, aprendizado de máquina (AM), um dos subcampos da IA, aplicado à análise de dados, vem sendo empregado em outras instituições [16]. Casos bem sucedidos no Tribunal de Contas da União (TCU), Ministério Público Federal, Polícia Federal, e na própria CGU, estão obtendo resultados

animadores, aumentando a eficiência nos gastos públicos. O TCU, desde 2017, utiliza uma versão da ferramenta ALICE, buscando combater irregularidades em licitações, processando dados do Comprasnet e Diário Oficial da União. O sistema Mônica, desenvolvido pelo TCU, é um painel de monitoramento onde todas as compras públicas podem ser visualizadas, incluindo as que não são auditadas pela ALICE, como contratações diretas e inexigibilidades de licitação [17].

Apesar da reconhecida eficiência da ALICE [10, 12], ainda não existem trilhas que analisem individualmente item a item do pregão para identificar possível conluio entre os participantes. Muitas vezes, esse tipo de análise, com o intuito de identificar uma possível irregularidade em um item, é de difícil execução. Os auditores devem avaliar, em tempo hábil, muitas variáveis e um grande volume de informações [11]. Desta forma, apoiar os auditores com uma ferramenta que indique a possível atuação de cartéis na execução de compras públicas, evitaria que recursos públicos vultuosos sejam desperdiçados nestas ações criminosas. Dado que a janela de atuação preventiva de auditoria nestes certames pode ser de até 8 dias, considerando sua publicação no Comprasnet e a efetiva execução do pregão, o prazo para identificar e atuar nestes certames é um grande desafio [11].

1.2 OBJETIVO

O objetivo deste trabalho é identificar a possível prática de conluio para formação de cartel entre as empresas que disputaram um item de pregão eletrônico do Comprasnet, por meio da aplicação de modelos de aprendizado de máquina.

1.3 CONTRIBUIÇÕES

Neste trabalho, foi realizado o estudo de um conjunto de algoritmos de aprendizado de máquina aplicados na detecção de possível conluio entre participantes de licitações, em 4 cenários distintos. Estes cenários são compostos por variáveis extraídas do Comprasnet e também de *datasets* disponibilizados por instituições de Estado de 5 países diferentes. Além das variáveis extraídas, índices estatísticos foram incluídos nos *datasets* para potencializar o desempenho dos algoritmos de AM. Considerando as métricas adotadas, indicando como promissora a capacidade de identificação de conluio, os algoritmos de melhor desempenho foram: *Extra Trees*, *Random Forest* e *Ada Boost*. Com desempenho destacado, o algoritmo *Extra Trees*, em um cenário específico composto pelos dados comuns de todos os *datasets* do estudo, foi capaz de identificar novos casos com suspeição de conluio no Comprasnet. Dessa forma, podemos apontar os seguintes resultados obtidos:

- nova trilha de auditoria para a ferramenta ALICE da CGU, elevando a capacidade de detecção de conluio entre as empresas participantes de pregões do Comprasnet;
- revisão dos métodos de detecção de conluio em pregões;
- indicação de algoritmos de AM com melhor desempenho no cenário estudado;

- geração de *dataset* com lances colusivos a partir de pregões do Comprasnet.

1.4 ESTRUTURA DA DISSERTAÇÃO

As seções seguintes desta dissertação estão organizadas da seguinte forma. O Capítulo 2 apresenta a fundamentação teórica de AM, direcionando para os tipos de algoritmos utilizados neste trabalho. O Capítulo 3 apresenta a definição de conluio na legislação brasileira, os trabalhos relacionados à detecção de conluio em compras públicas e trabalhos que embasam o experimento realizado. O Capítulo 4 apresenta como o experimento foi conduzido para gerar os *datasets* de conluio com pregões do Comprasnet, o desempenho dos algoritmos selecionados para testar a detecção de conluio e as métricas aplicadas na comparação dos modelos. O Capítulo 5 apresenta a discussão dos resultados obtidos no experimento e as dificuldades encontradas. Por fim, o Capítulo 6 apresenta as conclusões obtidas com o trabalho e possíveis evoluções futuras.

2 FUNDAMENTAÇÃO TEÓRICA

O trabalho realizado é baseado na aplicação de AM na identificação de possível conluio entre empresas participantes da disputa de itens de pregões eletrônicos conduzidos no Comprasnet. Desta forma, neste capítulo, apresentamos uma introdução aos conceitos de AM. Além das classificações gerais AM, este capítulo introduz as principais técnicas utilizadas pelos algoritmos selecionados para a execução do experimento.

2.1 APRENDIZADO DE MÁQUINA

Aprendizado de máquina é uma área da Inteligência Artificial que fornece aos sistemas de computador a capacidade de aprender e evoluir sem a programação explícita de operações por um ser humano. Ou seja, o sistema dotado de modelos de AM é capaz de aprender uma determinada tarefa (ex: reconhecimento de padrões) a partir da experiência, isto é, dos dados apresentados a ele. Depois que um modelo de AM é treinado, ele pode ser aplicado na resolução de novos problemas usando relações identificadas durante o treinamento [18]. A aplicação de AM é bem sucedida na solução de problemas reais nas mais diversas áreas, como [19]:

- na saúde, prevendo diagnósticos de doenças;
- em finanças, analisando padrão de consumo e detecção de fraude;
- na indústria automotiva, com a condução autônoma de veículos;
- reconhecimento de fala e muitos outros.

No contexto de AM, um modelo consiste na representação matemática que descreve a relação entre um conjunto de entradas e saídas. Esses modelos são treinados a partir de um conjunto de dados de treinamento, que consiste em pares de entradas e saídas conhecidas, visando aprender a relação entre elas e, assim, ser capaz de generalizar para reconhecer novos exemplos [20]. Este tipo de treinamento é conhecido por Aprendizado Supervisionado, como será mostrado na seção 2.1.1. Assim, um conjunto de dados de entrada é processado para obter atributos (variáveis, campos ou *features*) compatíveis com os algoritmos de AM. As saídas associadas são chamadas de rótulos, responsáveis por identificarem as classes alvo da identificação. A adoção de métricas de avaliação de desempenho do modelo treinado é fundamental neste processo.

Com o treinamento do modelo, dois problemas comuns podem ocorrer em relação à sua capacidade de generalização [19]: *overfitting* e *underfitting*. No primeiro caso, o modelo apresenta forte especialização nos dados de treinamento, com baixa capacidade de generalização e identificação de novos casos em dados diferentes. Já no segundo, o modelo obtém baixas taxas de acerto, mesmo com os dados de treinamento.

Apesar de existirem outras classificações de AM [19], podemos assumir que existem três tipos básicos de Aprendizado de Máquina [21, 22, 23]: o Aprendizado Supervisionado, o Aprendizado Não Supervisionado e o Aprendizado por Reforço. O Aprendizado Supervisionado, o mais comum, é utilizado quando o conjunto de dados possui entradas e saídas conhecidas. O objetivo é treinar o modelo para prever as saídas corretas para novas entradas. O Aprendizado Não Supervisionado é utilizado quando não há saídas conhecidas no conjunto de dados. O objetivo é encontrar estruturas e padrões nos dados. Já o Aprendizado por Reforço é utilizado quando o modelo interage com um ambiente, gerando recompensa ou punição, de acordo com suas ações.

2.1.1 Aprendizado Supervisionado

Algoritmos de Aprendizado Supervisionado buscam identificar uma função que mapeia uma entrada para uma saída, com base em pares de entrada-saída de exemplo [22]. Esta função é inferida a partir de dados de treinamento previamente rotulados com as classes alvo da identificação. Em outras palavras, o aprendizado supervisionado envolve a alimentação do modelo com um conjunto de dados rotulados e as respostas esperadas para cada entrada.

O conjunto de dados de entrada é dividido em dados de treinamento e teste. Também chamado de *data-set*, tal conjunto de dados de entrada possui uma variável de saída que será prevista ou classificada. Dessa forma, os algoritmos aprendem algum tipo de padrão do conjunto de dados de treinamento e os aplicam ao conjunto de dados de teste para previsão ou classificação das entradas, assim como exemplificado na Figura 2.1.

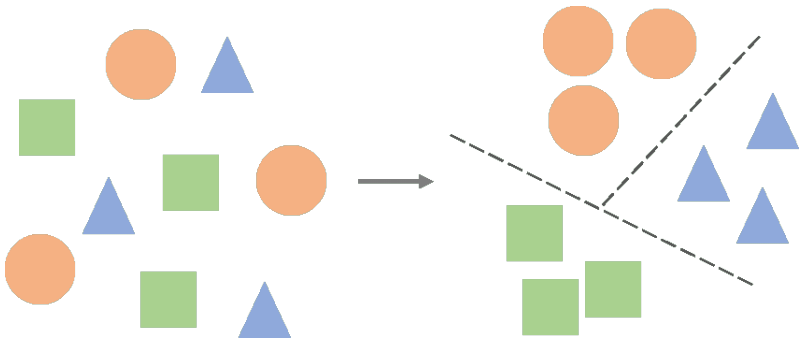

Figura 2.1: No Aprendizado Supervisionado, os algoritmos geram uma função que presenta a relação entre os dados de entrada e as saídas esperadas. Adaptado de [24].

Assim, após o treinamento, o modelo é avaliado usando o conjunto de teste, que contém dados que não foram vistos durante o treinamento. A precisão do modelo é medida comparando as previsões do modelo com as respostas corretas para o conjunto de teste. Se a precisão do modelo for alta o suficiente, ele poderá ser usado para fazer previsões precisas sobre novos dados de entrada.

2.1.2 Aprendizado Não Supervisionado

o Aprendizado não Supervisionado é uma técnica importante que permite aos modelos encontrarem padrões em dados não rotulados. Diferentemente do Aprendizado Supervisionado, onde os dados de entrada são rotulados com rótulos pré-definidos, o Aprendizado não Supervisionado é usado quando não há rótulos disponíveis para os dados [1]. Neste tipo de algoritmos de AM, não há respostas corretas e não há um treinador [21]. Assim, os algoritmos são deixados por conta própria para descobrir e apresentar a estrutura interessante nos dados, identificando padrões comuns, como ilustrado na Figura 2.2. Quando novos dados são introduzidos, ele usa os recursos aprendidos anteriormente para reconhecer a classe dos dados.

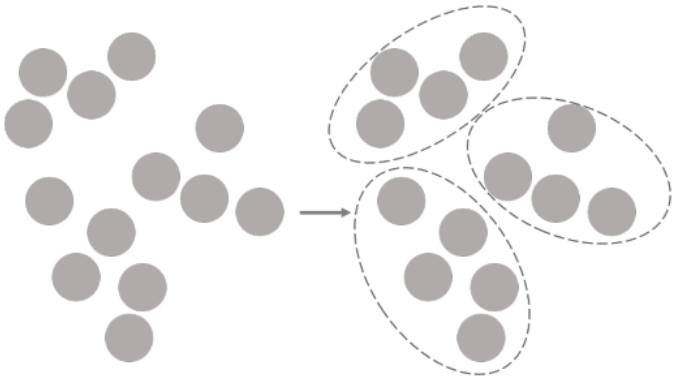

Figura 2.2: Algoritmos de Aprendizado Não Supervisionado identificam padrões que agrupam dados não rotulados em diferentes classes. Adaptado de [24].

As técnicas de Aprendizado não Supervisionado, como Agrupamento, Redução de Dimensionalidade e Detecção de Anomalias, permitem que os dados sejam organizados de forma mais eficiente e que as informações valiosas sejam extraídas dos dados sem a necessidade de rotulação prévia [23]. Embora o Aprendizado não Supervisionado seja um desafio, devido à falta de rótulos e direção nos dados, é uma técnica crucial para muitas aplicações onde os dados não são rotulados, onde a rotulação é muito cara ou difícil de obter.

2.1.3 Aprendizado por Reforço

O Aprendizado por Reforço é uma técnica de AM que se concentra em como os agentes devem tomar ações em um ambiente para maximizar uma recompensa numérica [20]. Ao contrário do Aprendizado Supervisionado, onde um modelo é treinado com exemplos rotulados, ou do Aprendizado não Supervisionado, onde o objetivo é encontrar padrões nos dados, o Aprendizado por Reforço se concentra em como um agente deve aprender a interagir com o ambiente ao longo do tempo para maximizar uma recompensa acumulada.

O Aprendizado por Reforço é comumente usado em aplicações de robótica, jogos e controle de processos industriais [20]. A abordagem é baseada em um agente que interage com um ambiente, recebendo informações sobre o estado atual do ambiente e tomando ações para mudar esse estado, como esquemati-

zado na Figura 2.3. O agente recebe uma recompensa numérica por suas ações, e seu objetivo é maximizar essa recompensa ao longo do tempo. O aprendizado por reforço é um processo iterativo em que o agente explora diferentes ações e observa os resultados dessas ações, a fim de aprender uma política que maximize a recompensa ao longo do tempo. No entanto, neste trabalho, algoritmos deste tipo não foram adotados.

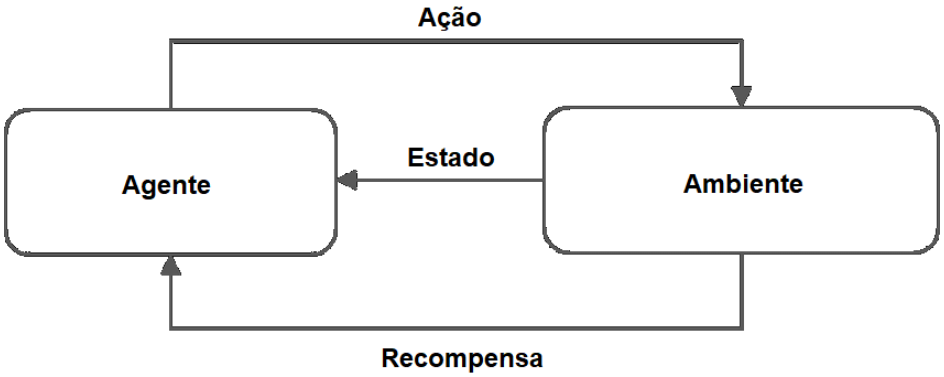

Figura 2.3: Aprendizado por Reforço baseado na busca de recompensa a partir da interação de um agente com o ambiente. Adaptado de [21].

2.2 TÉCNICAS E ALGORITMOS DE AM

O tipo de algoritmo empregado depende do tipo de problema que se deseja resolver, do número de variáveis, do tipo de modelo que melhor se adequa a ele e assim por diante. Abaixo, abordaremos algumas técnicas comumente aplicadas em algoritmos do paradigma de Aprendizado Supervisionado, também aplicadas neste trabalho. Algoritmos de AM baseados em Aprendizado Supervisionado foram escolhidos pela disponibilidade de *datasets* previamente rotulados que identificam a prática de conluio entre as empresas participantes de licitações alvo de cartéis.

2.2.1 Algoritmos baseados em distâncias

Algoritmos baseados nesta técnica consideram a proximidade entre os dados na predição pretendida [19]. Isto decorre de que dados similares tendem a estar concentrados em uma mesma região do espaço de entrada. Da mesma forma, dados não similares estarão distantes entre si. O algoritmo *K Nearest Neighbors* (em português, algoritmo dos k vizinhos mais próximos) é um exemplo desta estratégia [25], apresentado na Figura 2.4.

2.2.2 Algoritmos baseados em *Ensemble Learning*

O conceito de *Ensemble Learning*, também chamado de aprendizado por agrupamento, se baseia na ideia de combinar diversos modelos de predição mais simples, treiná-los para uma mesma tarefa, e produzir

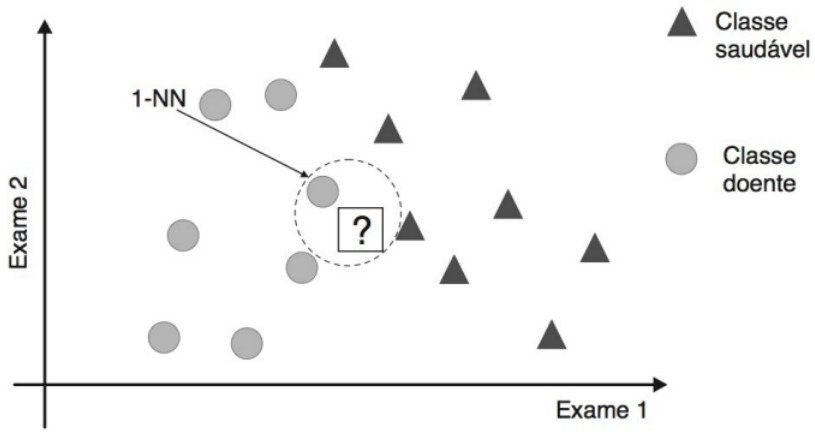

Figura 2.4: Exemplo do algoritmo *K Neighbors*, baseado na distância para classificação [19].

a partir desses um modelo agrupado mais complexo que é a soma de suas partes.

Esse agrupamento objetiva, em vez de usar apenas um modelo para prever as saídas, combinar um conjunto de modelos para fornecer previsões mais precisas e confiáveis. Portanto, os métodos de *ensemble* devem considerar a maneira com a qual eles agrupam os modelos, associando os algoritmos para minimizar suas desvantagens individuais no modelo final.

Existem vários tipos de algoritmos baseados em *ensemble methods*, com estratégias específicas, incluindo:

- *Bagging*: é um algoritmo que usa diferentes conjuntos de treinamento para cada modelo em um conjunto, visando reduzir a variância e o *overfitting*.
- *Ada Boost*: baseado no método *Boosting*, que treina modelos sequencialmente, ajustando os pesos dos exemplos de treinamento para dar mais ênfase aos exemplos que foram mal classificados pelos modelos anteriores [26].
- *Gradient Boosting*: também baseado em *Boosting*, combina vários modelos de árvore de decisão em uma série para melhorar a precisão geral [27].
- *Random Forests*: algoritmo que usa várias árvores de decisão para gerar previsões. Cada árvore é treinada com um subconjunto aleatório dos dados e usa uma seleção aleatória de recursos [28].
- *Extra Trees*: também baseado em árvore de decisão, aplicando várias árvores treinadas com subconjuntos aleatórios de dados e recursos [29].

Comumente usados em problemas de classificação e regressão, estes algoritmos podem melhorar significativamente a precisão e a estabilidade dos modelos de AM.

2.2.3 Algoritmos baseados em otimização

Técnicas de AM realizam a geração do modelo com base na otimização de alguma função [19]. Dessa forma, o problema de aprendizado é formulado como um problema de otimização que tem em vista minimizar (ou maximizar) uma função objetivo. Duas técnicas existentes em AM que recorrem à otimização de uma função em seu treinamento são as redes neurais e as *Suport Vector Machines* (SVM) [30].

Já o algoritmo *Stochastic Gradient Descent* (SGD), baseado em regressão linear, visa ajustar os parâmetros do modelo (como pesos e bias) para minimizar uma função de perda, sendo uma medida do quão bem o modelo está realizando sua tarefa (por exemplo, prever a classe correta de um dado de entrada). A ideia por trás do SGD é encontrar a direção na qual a função de perda é mais acentuada e, em seguida, ajustar os parâmetros do modelo nessa direção, diminuindo assim o valor da função de perda [31].

Já uma rede neural consiste em uma série de algoritmos que buscam reconhecer relacionamentos subjacentes em um conjunto de dados por meio de um processo que imita a maneira como o cérebro humano opera. Nesse sentido, as redes neurais referem-se a sistemas de neurônios, sejam de natureza orgânica ou artificial. As redes neurais podem se adaptar às mudanças de entrada; assim a rede gera o melhor resultado possível sem a necessidade de redesenhar os critérios de saída [19].

O *Multi Layer Perceptron* (MLP) [32], é uma rede neural sem retroalimentação (*feedforward*), com várias camadas ocultas usadas para aprender e mapear relações complexas entre os dados de entrada e saída. Uma representação gráfica dos neurônios desse tipo de rede é apresentada na Figura 2.5.

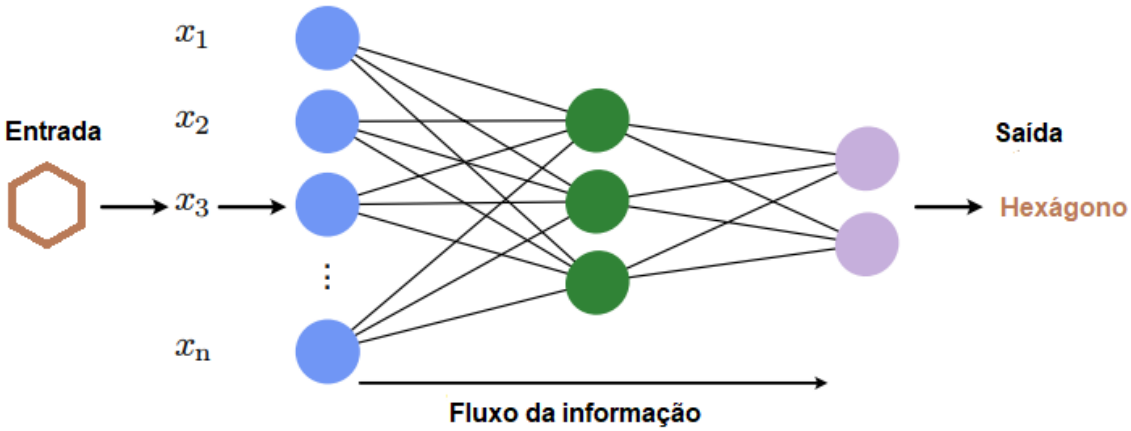

Figura 2.5: Rede Neural: representação básica da arquitetura. Adaptado de [20].

Já as redes neurais recorrentes são redes com *loops*, onde as informações que fluem pela rede persistem. Dessa forma, elas recebem novos dados como entrada, mas também se retroalimentam de dados que já processaram, mantendo uma espécie de memória na rede.

2.2.4 Algoritmos probabilísticos

Os métodos probabilísticos Bayesianos assumem a probabilidade de uma determinada classificação do dado de entrada em relação a valores específicos destes dados [19]. Tanto a frequência desta associação,

quanto a variação da combinação das propriedades de entrada possibilitam a identificação desejada.

Diferentemente de algoritmos baseados em distância, por exemplo, tomando como base a Figura 2.6, algoritmos probabilísticos desconsideram a distância e decidem a classificação dos elementos pela probabilidade de um elemento ser de uma classe considerando suas características. Dois algoritmos que exemplificam algoritmos probabilísticos são [32]: *Bernoulli Naive Bayes* e *Gaussian Naive Bayes*.

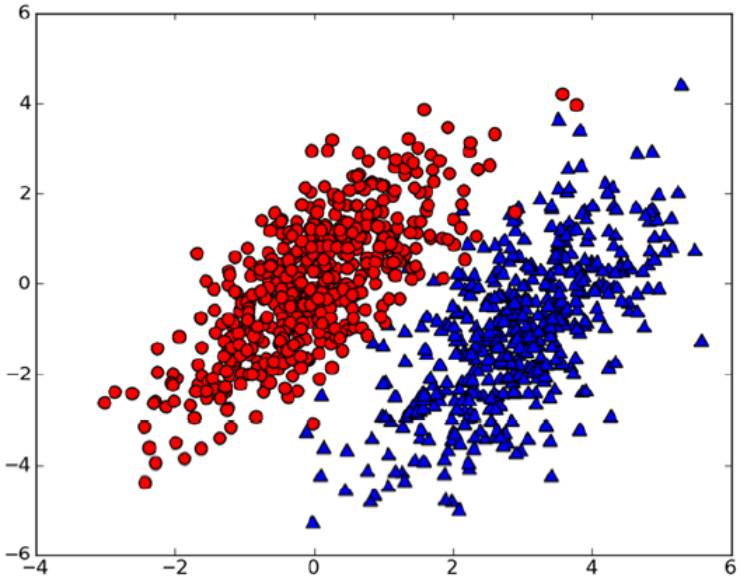

Figura 2.6: Duas distribuições probabilísticas baseadas em características conhecidas que as descrevem [1].

3 REVISÃO BIBLIOGRÁFICA

Neste capítulo, apresentamos a definição de conluio e a punição por formação de cartel na legislação brasileira. Adicionalmente, revisamos estudos recentes de aplicação de AM na detecção de ilícitos em compras públicas, entre eles, a identificação de conluio para fraudar licitações.

3.1 CONLUIO

A definição de conluio pode ser expressa como: "Combinação ou acordo estabelecido entre duas pessoas para prejudicar outrem; colusão, maquinação, trama" [33]. No contexto de licitações públicas, conforme definido pelo Conselho Administrativo de Defesa Econômica, o CADE, a formação de cartéis em licitações é caracterizada pelo conluio entre agentes econômicos para restringir ou eliminar a concorrência do processo de compra de bens, ou contratação de serviços pela Administração Pública. Além de limitar a efetiva concorrência natural entre as empresas nos certames, acarretando preços mais elevados, esta prática traz outros prejuízos ao Estado [15]. Acompanhado do prejuízo financeiro e desestruturação do mercado concorrente nos nichos de atuação do cartel, o Estado recebe produtos e serviços de menor qualidade. Com muita frequência, pela imposição dos preços elevados, o órgão licitante acaba adquirindo também uma quantidade menor do que a necessária [15].

Segundo a OCDE [13], a atuação de cartéis causa grandes danos a mercados consumidores, por meio da manipulação de preços ou restrição de oferta de produtos e serviços. Dessa forma, alguns consumidores deixam de adquirir o bem desejado ou o fazem em menor quantidade. Independentemente da decisão do consumidor, o cartel se fortalece, gerando desperdício de recursos e ineficiência. A proteção gerada pelos componentes do cartel à concorrência do mercado competitivo, acarreta aumento de gastos, prejuízos à inovação e perda de competitividade da economia.

Ainda segundo o CADE [15], as principais estratégias adotadas pelas empresas na formação de cartéis em licitações, muitas vezes aplicadas combinadamente, envolvem:

1. Propostas fictícias ou de cobertura (*cover bidding*), onde uma das empresas em conluio apresenta uma proposta mais elevada do que a escolhida para vencer a disputa;

2. Supressão de propostas (*bid suppresion*) e retirada de propostas (*bid withdrawal*), onde uma ou mais empresas em conluio desistem de concorrer, retirando uma proposta previamente apresentada, para que a empresa escolhida pelo cartel seja a vencedora;

3. Bloqueio em pregão presencial, onde as empresas em conluio buscam reduzir as chances de que outras licitantes não alinhadas sejam classificadas para a fase de lances em um pregão presencial, restringindo a concorrência na disputa;

4. Propostas rotativas ou rodízio (*bid rotation*), onde as empresas realizam a divisão dos certames,

combinando propostas alternadamente, de modo que cada uma seja vencedora de um item licitado, lote ou licitação;

5. Divisão de mercado (*market allocation* ou *market division*), onde as empresas em conluio combinam propostas para repartir o mercado de alguma forma (órgão público, produtos, serviços ou região geográfica);

6. Outras estratégias envolvendo mecanismos legais utilizados de forma anticompetitivas, como a formação de consórcios (reduzindo o universo da disputa) e a subcontratação (fragmentando o objeto entre as empresas em conluio).

Este problema é recorrente no mundo todo. Combater esta prática criminosa requer investigações de alto grau de complexidade e muitas vezes obter uma condenação é algo desafiador. Algoritmos também são utilizados pelos agentes envolvidos nestas ações anticompetitivas, pois possibilitam a formação e manutenção do conluio sem qualquer acordo formal ou interação humana [34, 35]. A punição deste ilícito está prevista no ordenamento jurídico brasileiro. A Lei n.º 12.529/2011, em seu artigo 36, § 3º, inciso I, alínea "d", especifica que a prática de cartel em licitação configura ilícito antitruste [36]. Esta conduta anticompetitiva, que fere a livre concorrência, é investigada e julgada pelo CADE, em esfera administrativa, que pode aplicar sanções às empresas e pessoas físicas, como multas ou impedi-las de participar de licitações, além de outras penalidades previstas na Lei [15].

No aspecto penal, a prática de cartel constitui crime contra a ordem econômica, previsto no artigo 116 da Lei n.º 12.529/2011, que revogou o artigo 4º da Lei n.º 8.317/1990, apurado judicialmente a partir de investigações das autoridades policiais e do Ministério Público. Neste caso, as penas previstas são de reclusão de dois a cinco anos e multa [36]. Ainda, esta pena pode ser aumentada caso o crime cause grave dano à coletividade, seja cometido por um servidor público ou relacionado a bens ou serviços essenciais para a vida ou para a saúde [37]. Outro enquadramento possível, no caso de cartéis em licitações, está previsto no artigo 178 da Lei n.º 14.133/2021, com pena de detenção de quatro a oito anos e multa [38].

3.2 IDENTIFICAÇÃO DE CONLUIO

Existem diversos modelos para detectar a prática de conluio na literatura. Nesta seção, revisaremos os modelos mais relevantes, que indicaram a existência de padrões de conluio de longo prazo entre empresas que formaram cartéis na disputa de licitações [39]. Esses modelos também são úteis para compreender como esses cartéis desencorajam as empresas de apresentarem propostas competitivas em mercados por eles dominados [40, 41]. Contudo, mesmo que esses modelos tenham conseguido detectar conluio, sua precisão é frequentemente questionada por conterem ruídos, informações insuficientes ou se basearem em informações de estruturas de custos particulares dos licitantes, como as estimativas de custos das empresas (ou *pre tender cost estimate - PTE*) [42, 43].

Diversas pesquisas no campo da teoria de leilões demonstram que as estratégias competitivas e/ou colusivas dos licitantes são fortemente influenciadas pelas estruturas de custos a que estão submetidos [39, 44]. McAfee e McMillan [45] foram os pioneiros na análise da colusão em esquemas de rotação de lances

estáticos, mesmo na ausência de pagamentos de compensação entre os membros do cartel. Baseados no trabalho de McAfee e McMillian [23], Aoyagi [46] e Skrzypacz e Hopenhayn [47] ampliaram o modelo, considerando a colusão repetida em esquemas dinâmicos de rotação de lances. Uma das primeiras tentativas de desenvolver um modelo baseado em evidências empíricas foi realizada por Porter e Zona [39], que procuraram mensurar a probabilidade de um licitante ganhar uma licitação quando alguns fatores de custo observáveis são conhecidos. No entanto, esse modelo não se destinava a determinar diretamente a existência de colusão, mas sim a antecipar a faixa de preços de lances competitivos futuros.

3.3 IDENTIFICAÇÃO DE ILÍCITOS EM LICITAÇÕES COM AM

A identificação de ilícitos em compras públicas vem sendo objeto de numerosos estudos recentes [48, 49, 50, 51, 52]. Nai *et al.* [48], em uma revisão sistemática de 15 trabalhos relacionados à identificação de fraude em licitações públicas, publicados entre 2016 e 2021, identificaram que a maioria dos artigos adotou métodos típicos de AM, enquanto dois grupos menores aplicaram principalmente redes neurais e análise de redes.

Outra revisão, explorando 23 artigos publicados entre 2016 e 2021, focou nos métodos mais utilizados para detectar diferentes tipos de corrupção em compras públicas, entre eles o conluio [49]. Na referida revisão, métodos de mineração de dados e AM foram usados sobre um grande volume de dados coletados de diferentes *datasets*, como registros de contratos, listas negras de operadores econômicos, registros de empresas e outros. Os métodos incluem técnicas de classificação, visando identificar ligações entre operadores econômicos e entidades contratantes, mas também para encontrar empresas que participaram em colusão, bem como regras de associações e algoritmos de bases de dados de grafos.

A quantidade de pesquisas e aplicações de AM na detecção de conluio e, de forma mais abrangente, fraudes gerais em compras públicas é crescente e muito explorada nos últimos anos [48, 49, 52]. A identificação de conluio em pregões eletrônicos do Comprasnet foi objeto de estudo de outros trabalhos, aplicando diferentes abordagens de IA na detecção deste ilícito [52, 53, 54, 55].

O primeiro modelo fundamental para detectar conluio é conhecido como triagem econométrica, proposto por Bajari e Ye [56]. Esse modelo visa antecipar como uma distribuição padrão competitiva de lances deve se parecer, com base nos parâmetros de custo dos licitantes participantes. Normalmente, esses parâmetros de custo são dados privados, difíceis de coletar e frequentemente mantidos em segredo pelos próprios licitantes. Assim, grande parte dos dados precisa ser inferida por especialistas da área, resultando em perda de precisão.

O modelo de Bajari e Ye [56] foi implementado como uma forma funcional reduzida de regressão linear, onde informações adicionais de histórico de lances, estimativas de custo (PTE) e dados financeiros dos licitantes eram necessários. No entanto, estes dados também trouxeram limitações importantes como: confiança excessiva na forma funcional escolhida ao implementar a análise de regressão; alta sensibilidade a informações ausentes; e facilidade de trapacear quando o cartel identifica o modo de operação dos lances de cobertura, por exemplo. Dessa forma, a ausência de tais dados impede que o modelo seja aplicável em contextos reais de licitação.

Outro modelo proposto por Ballesteros-Pérez *et al.* [57], foca na análise de possíveis dispersões anormais na distribuição dos lances, supondo que deveriam seguir uma distribuição uniforme. Esse método é usado como uma aproximação para detectar conluio em conjunto com outras abordagens. Ele utiliza estatísticas de ordem simplificadas, considerando apenas as distâncias relativas entre os lances e desprezando a ordem de grandeza absoluta dos valores. No entanto, essa abordagem pode ser burlada por meio de lances de cobertura que simulam um padrão uniforme, mesmo que, em média, ainda sejam anormalmente altos.

O modelo proposto por Signor *et al.* [58] é baseado em um método probabilístico que analisa as propostas em duas etapas. Na primeira etapa, o modelo verifica se a distribuição geral dos lances corresponde a um cenário de referência, como uma distribuição Lognormal. Em seguida, a localização da distribuição pode ser aproximada por meio de leilões históricos, caso haja dados disponíveis sobre as estimativas de custo. Dessa forma, o modelo examina a distância entre os lances apresentados e as estimativas de custo. Na segunda etapa, a teoria da estatística de ordem é aplicada para analisar a dispersão do lance mais baixo, comparando a probabilidade do lance vencedor (o menor lance) ser concretizado como se tivesse sido gerado a partir da mesma distribuição de referência da primeira etapa. Dessa forma, o lance vencedor real observado é comparado com a estatística de ordem mais baixa de uma distribuição de referência calibrada. Se o desvio estatístico for significativo, é improvável que tal oferta seja competitiva. Embora robusto, o método é limitado pela disponibilidade e confiabilidade de estimativas de custo de uma série de leilões competitivos anteriores e do leilão analisado.

Já o modelo desenvolvido por Imhof [59, 60], foi o primeiro a examinar a aplicação de AM para licitação e detecção de conluio. Utilizando um *dataset* suíço com dados de construção de estradas e um pequeno conjunto de variáveis estatísticas, dois tipos de algoritmos de AM foram aplicados: regressão de Lasso e uma média ponderada de algoritmos (*ensemble*); e árvores de regressão empacotadas, *random forests* e redes neurais.

Maia *et al.* [52] propôs uma metodologia para detecção de anomalias, buscando possíveis conluios entre empresas licitantes. O modelo é baseado na utilização de mineração de padrões frequentes, correlação de séries temporais multivariadas e análise conjugada de multicritérios, chamadas microexpressões [52]. Para tanto, os autores definiram um conjunto de variáveis que representam anomalias na contratação. Entre as anomalias identificadas, o sobrepreço foi definido como o somatório da diferença entre o valor oferecido pela primeira colocada e o valor real de celebração do contrato.

Uma limitação identificada na aplicação da proposta de Maia *et al.* [52], está relacionada ao fato de que nem todas as microexpressões definidas no estudo são obtidas a partir de dados disponíveis para cálculo abrangentemente, como o motivo de desclassificação de uma empresa que venceu um pregão. Com relação à forma de identificação de sobrepreço, caso o contrato seja celebrado com a primeira colocada, não será possível calcular um provável sobrepreço. Esta simplificação possibilita a análise, mas abrange apenas este cenário onde a contratação não é realizada com a empresa que foi a primeira colocada na disputa pelo item da licitação. Ou seja, caso os participantes da disputa estejam em uma ação combinada para fraudar a licitação, onde o primeiro colocado seja efetivamente a empresa que celebrará o contrato, o método não identificará o sobrepreço resultante do cartel estabelecido.

Lima [53] apresentou uma metodologia para a detecção de padrões de conluio em licitações públicas, direcionada ao contexto de realização de obras públicas, aplicando algoritmos de processamento de texto,

utilizando como fontes os dados repositórios oficiais públicos, como o Diário Oficial da União.

Ralha e Silva [54] aplicaram técnicas de mineração de dados multiagente e descoberta de conhecimento em banco de dados, com o intuito de detectar a cartelização de mercados públicos se baseando em dados extraídos do Comprasnet. Como resultado, identificaram regras que auxiliariam na detecção de cartel, apesar do tempo de processamento envolvido nesta extração em grandes volumes de dados.

Carvalho e Carvalho [55] também aplicaram técnicas de mineração de dados e algoritmos Bayesianos para a descoberta de regras de conhecimento na identificação de risco de corrupção em entidades do governo federal.

García Rodríguez *et al.* [50], em uma pesquisa que se destaca tanto pela seleção dos algoritmos de AM testados, quanto pelos *datasets* que caracterizam a prática de conluio por empresas em pregões, obteve os melhores resultados para a detecção de conluio com algoritmos de AM baseados em *ensemble learning*. No referido trabalho, 11 algoritmos de AM, foram massivamente testados em seis *datasets* que caracterizam a prática de conluio entre os participantes de licitações públicas, oriundos de investigações criminais no Brasil, Itália, Japão, Suíça e Estados Unidos. Os cenários de teste envolveram a variação dos dados disponíveis nos *datasets*, bem como, a inclusão de índices estatísticos calculados sobre os próprios *datasets*. Segundo os autores, os 3 melhores algoritmos foram: *Extra Trees*, *Random Forest* e *Ada Boost*. Considerando o cenário onde todos os dados dos *datasets* estavam disponíveis, as acurácias observadas para estes algoritmos variaram entre 81% e 95%, com acurácias balanceadas geralmente acima de 73% (exceto para o *dataset* dos Estados Unidos) [50].

Lima e Delen [51] também testaram algoritmos presentes no estudo de García Rodríguez *et al.* [50], buscando identificar corrupção de forma mais abrangente em 132 países, utilizando variáveis de entrada fornecidas por instituições reconhecidas e respeitadas, como Banco Mundial, Transparência Internacional e Fundação Heritage, compreendendo os anos de 2017 e 2018. No referido trabalho, o algoritmo *Random Forest* obteve a melhor acurácia: 85,77% [51].

Faria *et al.* [61], baseados em um modelo de variáveis quantitativas, demonstraram que alguns fatores são preponderantes na variação dos preços dos produtos contratados por pregão eletrônico. As variáveis número de fornecedores de um item, especificidade do item, quantidade e frequência deste item nas transações, apresentam, com o método dos mínimos quadrados ordinários, um poder de explicação conjunta de 67,4% da variação do preço na licitação [61].

Considerando a complexidade envolvida na caracterização de conluio, a variedade de métodos já aplicados, bem como, seus pontos fortes e limitações apresentadas, a identificação deste ilícito no contexto do Comprasnet também se apresenta como desafiadora. No entanto, em primeira análise, seria possível obtermos tal indicação, com base no experimento realizado por García Rodríguez *et al.* [50], tanto com a geração de mais um *dataset* de pregões onde ocorreu conluio no Comprasnet, quanto na aplicação dos *datasets* disponibilizados e os algoritmos de AM testados. Adicionalmente, de forma experimental, seria possível verificar a relevância das variáveis que indicam a variação de preço em pregões, apresentadas por Faria *et al.* [61], na caracterização de conluio no cenário apresentado, a partir de dados extraídos do Comprasnet.

Dessa forma, o capítulo seguinte apresentará os procedimentos realizados em busca da identificação

de possível conluio em pregões eletrônicos realizados no Comprasnet. Tal identificação será realizada com base na extração de um novo *dataset* de conluio do Comprasnet, a composição deste *dataset* com outros *datasets* de conluio disponibilizados no trabalho de García Rodríguez *et al.* [50], bem como, com a capacidade de detecção dos algoritmos selecionados e treinados com estes dados na detecção de novos casos suspeitos de conluio nos pregões eletrônicos realizados pelo Comprasnet.

4 MATERIAIS E MÉTODOS

Visando a identificação de possível conluio em pregões eletrônicos no contexto do Comprasnet e, com base no experimento de García Rodríguez *et al.* [50], bem como, na inclusão de novas variáveis identificadas pela variação de preços nestes pregões, definidas por Faria *et al.* [61], os procedimentos executados neste trabalho estão resumidos na Figura 4.1.

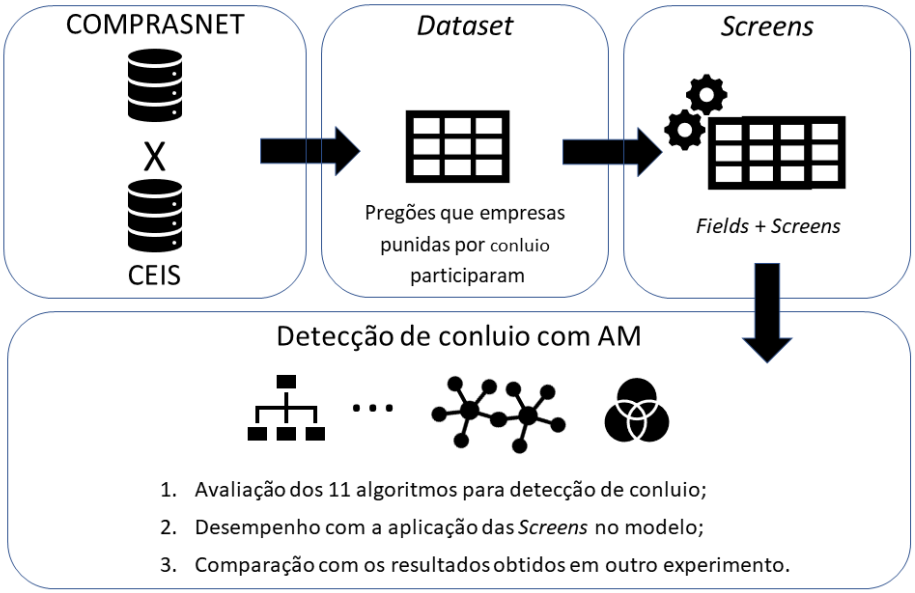

Figura 4.1: Metodologia para extração do *dataset* do Comprasnet e análise dos algoritmos.

Do experimento de García Rodríguez *et al.* [50], reutilizamos os algoritmos de AM selecionados para teste de detecção de conluio, a aplicação dos índices estatísticos (CV, SPD, DIFFP, RD, SKEW, KURT e KSTEST), dois *datasets* de conluio (Japão e *"All datasets"*), bem como, as métricas aplicadas na avaliação do desempenho dos algoritmos. Do estudo de Faria *et al.* [61], extraímos as variáveis do modelo identificado como responsável pela variação de preços em pregões. Ambos estudos foram adaptados e direcionados para a detecção de possível conluio em pregões eletrônicos realizados no Comprasnet.

A partir do cruzamento de pregões eletrônicos realizados no Comprasnet com a base de empresas punidas por conluio para formação de cartel em licitações públicas, geramos dois novos *datasets* com lances colusivos e competitivos. O primeiro, além dos lances ofertados nos pregões do Comprasnet, incluímos novas variáveis responsáveis por explicar a variação de preços nos pregões, segundo Faria *et al.* [61]. Já o segundo, foi gerado a partir da junção dos campos em comum do primeiro *dataset* com o *"All datasets"*.

Em seguida, incluímos as variáveis estatísticas (*screens*) nos *datasets* gerados e aplicamos os algoritmos de AM sobre 4 cenários distintos, variando a disponibilidade de campos para treinamento dos modelos. Por fim, comparamos o desempenho dos modelos gerados e aplicamos o de melhor desempenho sobre um

novo conjunto de pregões do Comprasnet, buscando verificar a capacidade de detecção de novos casos de possível conluio. Dessa forma, descrevemos as etapas do experimento em maiores detalhes nas seções seguintes.

4.1 IDENTIFICAÇÃO DE EMPRESAS PUNIDAS POR CONLUIO NO CEIS

O Cadastro Nacional de Empresas Inidôneas e Suspensas (CEIS) consiste em uma relação de empresas e pessoas físicas que sofreram sanções que implicaram a restrição de participar de licitações ou de celebrar contratos com a Administração Pública. Este cadastro, mantido pela CGU e de acesso público pelo Portal da Transparência, possui, entre outros tipos de sanções, a relação de entes punidos pela prática de conluio, enquadrados na Lei nº 12.529/2011. Dessa forma, aplicando os filtros categoria da sanção do tipo "Impedimento" e órgão sancionador como "CADE" no CEIS, identificamos o CNPJ das 24 empresas com impedimento de contratação com base na Lei Antitruste [62].

As empresas identificadas estão proibidas de contratar com instituições financeiras oficiais e participar de licitação tendo por objeto aquisições, alienações, realização de obras e serviços, concessão de serviços públicos, na administração pública federal, estadual, municipal e do Distrito Federal, bem como em entidades da administração indireta, por prazo não inferior a 5 (cinco) anos, conforme o inciso II, artigo 38, da Lei 12.529/2011, pelos seguintes períodos identificados nos dados extraídos do CEIS: 23/12/2020 a 23/12/2025, 18/08/2021 a 18/08/2026 e 01/09/2021 a 01/09/2026 [62].

4.2 EXTRAÇÃO DOS PREGÕES DO COMPRASNET

Com exceções admitidas no Decreto 10.024/2019 [7], o Comprasnet, Portal de Compras do Governo Federal, operacionaliza a realização dos pregões eletrônicos de compras públicas. Estes pregões, bem como seus itens e lances ofertados pelas empresas participantes, foram extraídos diretamente da base de dados do Comprasnet, via Acordo de Cooperação entre a CGU e o Ministério da Economia [63]. As principais fontes destes pregões são as seguintes tabelas:

- tbl_pregao: onde os pregões e seus atributos básicos são mantidos;
- tbl_pregaoitem: contempla os itens licitados no pregão, assim como o tipo do item licitado, a quantidade, valor estimado e outros;
- tbl_lance: contém cada lance enviado por um fornecedor para um item do pregão;
- tbl_proposta: contém os dados básicos da empresa participante do pregão;
- tbl_propostaitem: contém os valores de proposta inicial para cada item do pregão.

Os pregões desejados para a composição do *dataset* foram identificados pela participação de empresas previamente punidas pela prática de conluio, identificadas no CEIS.

4.3 FORMAÇÃO DOS *DATASETS*

De posse do CNPJ das empresas com a referida restrição no CEIS, realizamos um levantamento na base do Comprasnet, dos pregões eletrônicos que tais empresas participaram previamente à punição recebida, com base na consulta abaixo:

```
SELECT
p.prgCod, p.numprp, p.coduasg,
pri.ipgCod, l.lanCod, pro.prpCNPJ, l.lanValor,
(case when l.lanValor = pri.ipgValorMinClassif
then 1 else 0 end) as Winner,
l.lanData, pri.ipgValorRef,
(pri.ipgValorRef - l.lanValor) as Difference_Bid_PTE,
pri.ipgValorMinClassif, pri.ipgQuantidade,
pri.prgCod, pri.ipgItem, pri.codmat
FROM tbl_Pregao p
INNER JOIN tbl_pregaoitem pri
ON p.prgCod = pri.prgCod
INNER JOIN tbl_Lances l
ON pri.ipgCod = l.ipgCod
INNER JOIN tbl_Proposta pro
ON pro.prgCod = pri.prgCod
AND pro.Cliente_ID = l.cliente_id
WHERE pri.prgCod in (
    SELECT distinct proi.prgCod
    FROM tbl_Proposta proi
    WHERE proi.prpCNPJ in (
    '02293852000140', '01785999000194', '96355946000140',
    '35596501000167', '03187742000166', '62436282000121',
    '88309620000158', '01478038000137', '07386939000185',
    '11572018000184', '04717562000101', '04809827000100',
    '06332079000134', '25181298000104', '01140694000125',
    '08859696000118', '37517158000143', '03737267000154',
    '13385812000144', '02845074000154', '49254634000160',
    '44164606000138', '14744743000180', '02332985000188'))
AND ipgQuantidade > 0
AND l.lanStatus = 'V'
AND pri.ipgFormaJulg = 'V'
AND pri.ipgValorMinClassif is not null
ORDER BY pri.ipgCod, pri.ipgItem, l.lanValor
```

Os campos obtidos para composição do *dataset* Comprasnet foram: código do item licitado (ipgI-

tem), valor do lance (lanValor), data do lance (lanData), valor estimado (ipgValorRef), diferença do valor estimado e valor do lance (Difference_Bid_PTE), lance vencedor (Winner), quantidade licitada (ipgQuantidade) e o número de lances que o item recebeu, calculado a partir da contagem dos lances do item licitado.

Os lances ofertados pelas empresas punidas por conluio foram previamente classificados como colusivos. Já os lances ofertados pelas demais empresas participantes dos certames, foram classificados como competitivos. Dessa forma, geramos o *dataset* rotulado e formatado como os *datasets* do estudo base.

O critério de marcação de conluio dos lances apresentados pelas empresas punidas com base na Lei Antitruste é uma tentativa empírica de inferir que o comportamento anticompetitivo, comprovado pelas autoridades que as puniram, foi repetido em outras licitações que elas participaram. Assim, assumimos que estas empresas adotaram o mesmo comportamento colusivo nos certames disputados anteriormente às respectivas condenações.

Os pregões identificados no Comprasnet que obtiveram a participação de pelo menos uma das empresas punidas, compreendem certames realizados entre 2006 e 2015, onde foram licitados 445 itens, que geraram 11.049 lances individuais. Apenas 6,26% desses lances, o que corresponde a 692, foram assumidos como colusivos, seguindo a estratégia de marcação descrita acima. Os lances foram realizados por 147 empresas diferentes, entre elas, apenas 7 estavam entre as 24 identificadas no CEIS. Este *dataset*, extraído do cruzamento de dados entre Comprasnet e CEIS, será nomeado simplesmente como Comprasnet adiante.

No estudo base, um *dataset* chamado *"All datasets"*, foi composto pelos campos comuns aos seis *datasets* colusivos. Seguindo esta proposta, no presente estudo, uma nova versão deste *dataset* foi gerada, agregando os campos comuns do *dataset* do Comprasnet. Assim, esta nova versão do *"All datasets"*, expandida com os dados do Comprasnet, foi denominada *"All+Comprasnet"*. Portanto, este novo *dataset* gerado, contempla a variabilidade de lances colusivos e competitivos de 7 outros *datasets*.

4.4 *CÁLCULO DAS SCREENS*

A identificação de conluio a partir de algoritmos de AM é potencializada pela inclusão no modelo de *Screening Variables*, ou simplesmente *Screens*. As *screens* são índices estatísticos calculados a partir dos lances ofertados pelos licitantes do certame [60]. Em outros experimentos, as *screens* são reconhecidas por aumentarem a capacidade de detecção de conluio pelos algoritmos de AM selecionados [50, 59, 60].

Assim, a partir do *dataset* do Comprasnet gerado neste experimento, o próximo passo foi o cálculo das *screens* detalhadas abaixo, para cada item licitado *t*, com base no mesmo mecanismo utilizado do experimento precursor. Consideremos ainda: o cálculo de sd_t como desvio padrão dos lances de cada item *t* licitado; $\bar{b_t}$ como a média de todos os lances recebidos pelo item *t*; $b_{max,t}$ como o lance mais alto ofertado para o item *t*; $b_{min,t}$ como o lance mais baixo ofertado para o item *t*; b_{2t} como o segundo lance mais baixo ofertado para o item *t*; $sd_{losingbids,t}$ como o desvio padrão dos lances não vencedores do item *t*; n_t como o número de lances ofertados ao item *t*; b_{it} como o *i*-ésimo lance do item licitado *t*, quando ordenado do menor para o maior lance. O código-fonte utilizado para o cálculo das *screens* está disponível nos apêndices.

4.4.1 *Coefficient of Variation (CV)*

A *screen CV* representa a razão entre o desvio padrão dos lances do item e a média dos lances deste item.

$$CV_t = \frac{sd_t}{\bar{b_t}} \tag{4.1}$$

4.4.2 *Spread (SPD)*

A *screen SPD* representa a dispersão dos lances de um item.

$$SPD_t = \frac{b_{max,t} - b_{min,t}}{b_{min,t}} \tag{4.2}$$

4.4.3 *Difference between the two lowest bids (DIFFP)*

A *screen DIFFP* representa a diferença relativa entre os dois lances mais baixos do item.

$$DIFFP_t = \frac{b_{2t} - b_{min,t}}{b_{min,t}} \tag{4.3}$$

4.4.4 *Relative Distance (RD)*

A *screen RD* é um índice alternativo ao DIFFP, substituindo o termo no denominador pelo desvio padrão dos lances perdedores do item.

$$RD_t = \frac{b_{2t} - b_{min,t}}{sd_{losingbids,t}} \tag{4.4}$$

4.4.5 *Skewness (SKEW)*

A *screen SKEW* possibilita verificar a distribuição assimétrica dos lances do item.

$$SKEW_t = \frac{n_t}{(n_t - 1)(n_t - 2)} \sum_{i=1}^{n_t} \left(\frac{b_{it} - \bar{b_t}}{sd_t} \right)^3 \tag{4.5}$$

4.4.6 *Excess Kurtosis (KURT)*

A *screen KURT* representa a condensação de valores dos lances muito próximos (ou muito distantes) da média de lances do item, mas requer pelo menos 4 lances para o seu cálculo.

$$KURT_t = \frac{n_t(n_t+1)}{(n_t-1)(n_t-2)(n_t-3)} \sum_{i=1}^{n_t} \left(\frac{b_{it} - \bar{b_t}}{sd_t} \right)^4 - \frac{3(n_t-1)^3}{(n_t-2)(n_t-3)} \tag{4.6}$$

4.4.7 *Kolmogorov-Smirnov test (KSTEST)*

A *screen KSTEST* representa a semelhança dos valores dos lances para uma distribuição uniforme.

$$KSTEST_t = max(D_t^+, D_t^-) with D_t^+ = max_i \left(\frac{b_{it}}{sd_t} - \frac{i_t}{n_t+1} \right), D_t^- = max_i \left(\frac{i_t}{n_t+1} - \frac{b_{it}}{sd_t} \right) \tag{4.7}$$

4.5 INCLUSÃO DAS *SCREENS* DE VARIAÇÃO DE PREÇO

Buscando testar outras *screens* na identificação de conluio, adicionamos novas variáveis ao *dataset* do Comprasnet. Faria *et al.* [61] demonstraram que as variáveis a seguir apresentaram um poder de explicação conjunta de 67,4% das variações dos preços em pregões eletrônicos.

Tais variáveis foram calculadas a partir da base de dados do Comprasnet e adicionadas ao *dataset*, buscando dotar o modelo de características que indiquem a variação de preço de um item licitado. Ou seja, quanto menos variou o preço de um item, maior a probabilidade da existência de conluio na licitação, já que o objetivo dessa prática é garantir que o preço varie pouco, sem competição entre os participantes [15]. O código-fonte completo utilizado para realizar o cálculo destas variáveis também está disponível nos apêndices.

4.5.1 Total de lances (*LANC*)

A variável *LANC* representa a quantidade de lances apresentados na disputa de um item *t* do pregão. Esta variável possui uma interferência positiva, pois quanto mais lances um item recebe, maior a probabilidade do preço cair, indicando maior concorrência.

$$LANC_t = n_t \tag{4.8}$$

Esta variável pode ser extraída da base do Comprasnet por meio da seguinte consulta, recendo como entrada o código do pregão e retornando como saída a quantidade de lances por item:

```
SELECT pri.ipgCod, count(l.ipgCod) as qtdeLances
FROM tbl_pregaoitem pri
INNER JOIN tbl_Lances l
ON pri.ipgCod = l.ipgCod
WHERE pri.prgCod = :codigoDoPregao
```

```
AND l.lanStatus = 'V'
GROUP BY pri.ipgCod
```

4.5.2 Quantidade (*QUANT*)

A variável *QUANT* representa a quantidade *w* licitada do item *t* do pregão, ou seja, o quanto será comprado deste item. Sua interferência é positiva, pois quanto maior a quantidade a ser comprada, maior a probabilidade do custo unitário ser menor.

$$QUANT_t = w_t \tag{4.9}$$

Esta variável pode ser extraída da base do Comprasnet por meio da seguinte consulta, recebendo como entrada o código do pregão e retornando como saída a quantidade adquirida por item:

```
SELECT pri.ipgCod, pri.ipgQuantidade as quantidade
FROM tbl_pregaoitem pri
WHERE pri.prgCod = :codigoDoPregao
```

4.5.3 Escore de especificidade (*ESPECIF*)

A variável *ESPECIF* representa o escore de especificidade de um item *t* do pregão. Ela possui uma interferência negativa, pois quanto mais específico um item comprado, menor será o número de empresas concorrentes e, por isso, menor será a concorrência na licitação. No entanto, ela foi computada de forma invertida, pois foi totalizado a quantidade de vezes que a mesma compra realizada em *t* foi licitada em outros pregões. Dessa forma, quanto maior o valor obtido, menor será a especificidade da compra.

$$ESPECIF_t = (\sum_{i=1}^{t} f(t)) - 1, \text{ onde } f(t) = \begin{cases} 0 & \text{se compra de } t \text{ for diferente} \\ 1 & \text{se compra de } t \text{ for igual} \end{cases} \tag{4.10}$$

Esta variável pode ser extraída da base do Comprasnet por meio da seguinte consulta, recebendo como entrada o código do pregão e retornando como saída a quantidade de licitações diferentes dela própria, que licitou o mesmo material:

```
SELECT pri.codmat, count(1) as especificidade
FROM tbl_pregaoitem pri
INNER JOIN tbl_pregaoitem prif
ON pri.codmat = prif.codmat and prif.ipgCod != pri.ipgCod
WHERE pri.prgCod = :codigoDoPregao
GROUP BY pri.codmat
```

4.5.4 Escore de frequência (*FREQ*)

A variável *FREQ* representa o escore de frequência da empresa vencedora de um item *t* do pregão. Ela possui uma interferência positiva, pois fornecedores que prezam relacionamentos mais duradouros fazem maiores esforços para continuar o fornecimento e, para isso, são obrigados a reduzir seus preços. Esta variável foi calculada como a quantidade de vezes que a empresa vencedora do item *t* também foi a vencedora do mesmo tipo de compra em outras licitações.

$$FREQ_t = (\sum_{i=1}^{t} f(t)) - 1, \text{ onde } f(t) = \begin{cases} 0 & \text{se empresa vencedora diferir da vencedora de } t \\ 1 & \text{se empresa vencedora for também a vencedora de } t \end{cases} \tag{4.11}$$

Esta variável pode ser extraída da base do Comprasnet por meio da seguinte consulta, indicando o código do pregão, retornando a frequência por elemento licitado do pregão:

```
SELECT prif.codmat, count(1) as frequencia
FROM tbl_pregaoitem pri
INNER JOIN tbl_propostaitem ppi
ON pri.ipgCod = ppi.ipgCod and ppi.ippIndAdjudicado = 'S'
INNER JOIN tbl_proposta pp
ON pp.prpCod = ppi.prpCod
INNER JOIN tbl_pregaoitem prif
ON pri.codmat = prif.codmat and prif.ipgCod != pri.ipgCod
INNER JOIN tbl_propostaitem ppif
ON prif.ipgCod = ppif.ipgCod and ppif.ippIndAdjudicado = 'S'
INNER JOIN tbl_proposta ppf
ON ppf.prpCod = ppif.prpCod and pp.prpCNPJ = ppf.prpCNPJ
WHERE pri.prgCod = :codigoDoPregao
GROUP BY prif.codmat
```

4.6 ALGORITMOS E CENÁRIOS DE TESTE

Os algoritmos de AM testados no experimento, listados na Tabela 4.1, buscam resolver a classificação binária dos lances das licitações entre colusivos e competitivos (ou não colusivos). Os algoritmos baseados em *Ensemble methods* (*Extra Trees*, *Random Forest*, *Ada Boost* e *Gradient Boosting*) obtiveram os melhores resultados no experimento base.

Os algoritmos foram testados em 4 cenários diferentes para o *dataset* do Comprasnet, variando os campos selecionados em cada cenário da seguinte forma:

- Cenário 1 (todos os campos). Este cenário incluiu os campos código do item licitado, valor do lance, data do lance, lance vencedor, o número de lances que o item recebeu e as *screens* originárias do

Tabela 4.1: Algoritmos testados no experimento.

Método	Algoritmo
Linear Model	*SGD (Stochastic Gradient Descent)* [31]
Ensemble methods	*Extra Trees* [29]
	Random Forest [28]
	Ada Boost [26]
	Gradient Boosting [27]
Support Vector Machines	*SVC* [30]
Nearest Neighbors	*K Neighbors* [25]
Neural network models	*MLP (Multi-Layer Perceptron)* [32]
Naive Bayes	*Bernoulli Naive Bayes* [32]
	Gaussian Naive Bayes [32]
Gaussian Process	*Gaussian Process* [64]

estudo de Faria *et al.* [61]. Apesar de outros campos estarem disponíveis no *dataset*, alguns campos foram desconsiderados para permitir a comparação entre os cenários e não identificar o licitante, evitando um possível direcionamento dos algoritmos com esses dados.

- Cenário 2 (todos os campos + *screens*). Neste cenário temos os campos do cenário 1 mais as *screens* originais (CV, SPD, DIFFP, RD, SKEW, KURT e KSTEST).
- Cenário 3 (campos comuns). Somente os campos comuns aos *datasets* do estudo base: código do item licitado, o valor do lance, a data do lance, o lance vencedor e o número de lances por item licitado.
- Cenário 4 (campos comuns + *screens*). Os campos do cenário 3 mais as 7 *screens* iniciais adicionadas ao *dataset*.

Os cenários 1 e 3 são diferenciados apenas pelas *screens* propostas por Faria *et al.* [61]: total de lances, quantidade, escore de especificidade e escore de frequência. Dessa forma, a comparação do resultado dos algoritmos nestes dois cenários evidenciará a relevância delas na identificação de lances colusivos. No entanto, o total de lances recebidos por um item de pregão já faz parte do *dataset* no cenário 1. Assim, apenas as outras três variáveis diferenciam estes cenários efetivamente.

Para o *dataset "All+Comprasnet"*, apenas os cenários 3 e 4 serão aplicados, já que os campos exclusivos do Comprasnet, calculados como possíveis novas *screens*, não estão disponíveis na versão original do *"All datasets"*. Além disso, o campo data do lance não será considerado, pois não está disponível na formação de todos os *datasets* do estudo base.

4.7 MÉTRICAS UTILIZADAS

A comparação do desempenho dos algoritmos do experimento considerou 5 métricas de erro, definidas a seguir para este contexto de classificação: *Accuracy*, *Precision*, *Recall*, *Balanced Accuracy* e *F1 score*.

O objetivo final da classificação é realizar a distinção entre lances colusivos e competitivos entre os

n lances dos *datasets*. Assim, classificar um lance corretamente como colusivo consiste em realizar uma identificação *True Positive (TP)*. Classificá-lo como competitivo corretamente significa realizar uma identificação *True Negative (TN)*. Por outro lado, quando um lance competitivo é classificado como colusivo, temos uma identificação *False Positive (FP)*. E por fim, quando um lance é colusivo, mas classificado como competitivo, temos uma identificação *False Negative (FN)*. Portanto, podemos calcular as métricas relacionadas como nas equações abaixo.

4.7.1 *Accuracy*

A métrica *Accuracy* representa o percentual de classificações corretas, tanto de itens colusivos quanto de competitivos, em relação ao total de lances analisados.

$$Accuracy = \frac{TP + TN}{n} \tag{4.12}$$

4.7.2 *Precision*

A métrica *Precision* representa a habilidade do classificador de não determinar como colusivo um lance competitivo.

$$Precision = \frac{TP}{TP + FP} \tag{4.13}$$

4.7.3 *Recall*

A métrica *Recall* representa a habilidade do classificador de identificar todos os lances colusivos.

$$Recall = \frac{TP}{TP + FN} \tag{4.14}$$

4.7.4 *Balanced Accuracy*

A métrica *Balanced Accuracy* é um bom indicador para *datasets* desbalanceados, como no *dataset* extraído do Comprasnet, indicando a média de identificação positiva de lances de cada classe (colusivos/-competitivos).

$$BalancedAccuracy = \frac{1}{2}(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}) \tag{4.15}$$

4.7.5 *F1 score*

A métrica *F1 score* representa a média harmônica entre *Precision* e *Recall*, onde o score mais próximo de 1 representa o melhor valor e mais próximo de 0, o pior.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4.16}$$

4.8 COMPARAÇÃO COM OS *DATASETS* DO ESTUDO BASE

Os resultados obtidos com o experimento no *dataset* Comprasnet, serão comparados com os resultados obtidos no experimento base, restrito ao *dataset* oriundo do Japão. Este *dataset* é o que mais se parece com o obtido neste estudo, tanto em relação à quantidade de itens licitados e número de lances, quanto em relação ao percentual de itens colusivos e competitivos.

Da mesma forma, os resultados dos algoritmos sobre o *dataset "All+Comprasnet"* serão comparados aos obtidos no experimento base sobre o *"All datasets"*, a sua versão original.

Uma visão resumida sobre os dois *datasets* gerados neste trabalho (Comprasnet e *"All+Comprasnet"*) e dos outros dois utilizados como referência (Japão e *"All datasets"*), é apresentada na Tabela 4.2. As características apresentadas, como os campos utilizados de cada *dataset*, informações gerais sobre os lances dos itens licitados, distribuição de lances e valores envolvidos, revelam a composição dos *datasets* em alto nível.

4.9 TREINAMENTO E EXECUÇÃO

Os algoritmos de AM foram treinados e testados para detectar possível conluio no *dataset* gerado do Comprasnet nos 4 cenários apresentados. Assim como, para a detecção no *dataset "All+Comprasnet"* para os cenários 3 e 4. Cada algoritmo foi executado 50 vezes, aleatoriamente alterando o subconjunto de treinamento, composto por 80% das licitações, e o subconjunto de teste, pelos 20% restantes. Os parâmetros utilizados em cada algoritmo e todo o processamento do *dataset* estão disponíveis nos apêndices.

A cada iteração dos algoritmos, as métricas de erros foram calculadas e armazenadas. Dessa forma, as métricas apresentadas na próxima seção, correspondem às médias obtidas ao final dessas iterações para ambos os *datasets*: Comprasnet e *"All+Comprasnet"*.

Por fim, o modelo gerado pelo algoritmo de melhor desempenho no processamento de cada *dataset*, considerando as métricas utilizadas na comparação, foi aplicado diretamente na base de dados do Comprasnet, para um período diferente dos utilizados na composição dos *datasets* do experimento. Este processamento posterior, em forma de trilha de auditoria da ALICE, tem o objetivo de gerar alertas em pregões cujos itens possuam indício de conluio entre as empresas participantes do certame.

Os modelos foram treinados em ambiente Windows 10 Pro contendo o Anaconda3, Jupyter Notebook e Python 3.7, utilizando os seguintes pacotes e respectivas versões: numpy (1.19.2), pandas (1.1.3), joblib (0.17.0), scipy (1.5.2), matplotlib (3.3.2) e scikit_learn (0.24.2). A configuração de hardware que executou este ambiente contempla um processador Inter(R) Core(TM) i7-7700 CPU 3.60 GHz, 32GB de RAM e armazenamento SSD de 237GB.

Tabela 4.2: Características dos *datasets* gerados no experimento (Comprasnet e *"All+Comprasnet"*) e *datasets* utilizados para comparação (Japão e *"All datasets"*).

Grupo	**Característica**	***Datasets***			
		Japão	**Comprasnet**	***All datasets***	***All+Comprasnet***
Campos	Campos comuns	Código do item, valor dos lance, lance vencedor e número de lances do item.			
	Todos os campos do *dataset*	Campos comuns, valor estimado item, diferença do estimado e do valo do lance, localização e data.	Campos comuns, valor estimado item, diferença do estimado e valor do lance, data, quantidade licitada, escore de especificidade e escore de frequência.	Apenas campos comuns.	Apenas campos comuns.
	Número de campos utilizados	8	8	4	4
	Screens	*Coefficient of variation (CV), spread (SPD), percentage difference between the two lowest bids (DIFFP), relative distance (RD), skewness statistic (SKEW) and Kolmogorov–Smirnov test (KSTEST).*			
Informações gerais	Período	2003 a 2007	2006 a 2015	1980 a 2013	1980 a 2015
	Itens	1.080	445	9.781	10.226
	Lances	13.515	11.049	64.348	75.397
	Média de lances por item	12,51	24,83	6,60	7,39
	Mediana de lances por item	12,00	19,00	3,00	4,00
Dados Colusivos/ Competitivos	Itens colusivos	123 (11,39%)	54 (12,13%)	4.169 (42,62%)	4.223 (41,30%)
	Itens competitivos	957 (88,61%)	391 (87,87%)	5.612 (57,38%)	6.003 (58,70%)
	Lances colusivos	1.093 (8,09%)	692 (6,26%)	29.857 (46,40%)	30.549 (40,52%)
	Lances competitivos	12.422 (91,91%)	10.357 (93,74%)	34.491 (53,60%)	44.848 (59,48%)
Lances(N) por item	1<=N<=4	0 (0,00%)	79 (17,75%)	6.113 (62,50%)	6.192 (60,55%)
	5<=N<=10	474 (43,89%)	41 (9,21%)	2.612 (26,70%)	2.653 (25,94%)
	11<=N	606 (56,11%)	325 (73,03%)	1.056 (10,80%)	1.381 (13,50%)
Valor dos lances	Total de lances	€402.195.427	€15.128.289	€26.744.260.705	€26.759.388.994
	Total de lances colusivos	€91.405.888 (22,73%)	€2.776.395 (18,35%)	€16.156.304.504 (60,41%)	€16.159.080.899 (60,39%)
	Total de lances competitivos	€310.789.539 (77,27%)	€12.351.893 (81,65%)	€10.587.956.201 (39,59%)	€10.600.308.094 (39,61%)

5 RESULTADOS E DISCUSSÃO

Apresentamos a seguir, o resultado do processamento dos *datasets* gerados neste trabalho, Comprasnet e *"All+Comprasnet"*, pelos algoritmos listados na Tabela 4.1. A identificação dos algoritmos de melhor desempenho foi realizada com base nas métricas apresentadas no capítulo anterior. Em seguida, a partir do modelo gerado pela melhor combinação de algoritmo e cenário, para cada *dataset*, apresentamos o resultado da aplicação destes modelos a um novo conjunto de dados do Comprasnet.

5.1 *DATASET* COMPRASNET

Entre as métricas obtidas no experimento, *Accuracy*, *FP*, *FN* e *Balanced Accuracy*, assim como no estudo de García Rodríguez *et al.* [50], nos permitem analisar o desempenho dos algoritmos nos 4 cenários apresentados para o *dataset* extraído do Comprasnet. Além da comparação entre os cenários em si, a inclusão das métricas do *dataset* do Japão obtidas no experimento precursor, possibilita a análise dos resultados entre os dois *datasets*. A Figura 5.1 resume os dados obtidos nos cenários 1 e 2. Já a Figura 5.2 contempla os cenários 3 e 4.

Nestas figuras, as setas verdes indicam um melhor desempenho do algoritmo considerando a métrica no cenário, comparando-se o *dataset* do Comprasnet com o do Japão. Já as setas vermelhas, indicam uma piora nos índices em relação ao Comprasnet. O círculo verde indica métricas iguais em ambos os *datasets*.

Cenário 1 (todos os campos) e 2 (todos os campos + screens) - Algoritmos

		SGD				Extra Trees				Random Forest				Ada Boost				Gradient Boosting				SVC			
Métricas	Dataset\Cenário	1		2		1		2		1		2		1		2		1		2		1		2	
Accuracy(%)	Comprasnet	↓	84,4	↑	88	↓	94,6	↑	94,8	↑	94,2	↑	94,2	↑	93,6	↓	92	↑	94,2	↑	92,8	↓	76,8	↓	77,7
	Japão		87,8		87,8		94,7		94,5		93,1		93		93,5		93,1		90,5		89,2		87,8		87,9
False Positives (FP) (%)	Comprasnet	↓	9,5	↓	6,2	↑	0,3	↑	0,3	↑	1	↑	0,8	↑	1,2	↓	3,1	↑	0,3	↑	0,6	↓	19,4	↓	18,8
	Japão		5,6		5,6		0,9		0,8		2,6		2,6		2,4		2,5		4,8		5,8		9,4		9,3
False Negatives (FN) (%)	Comprasnet	↑	6,1	↑	5,8	↓	5	↓	5	↓	4,8	↓	5	↓	5,3	↓	5	↓	5,5	↓	6,6	↓	3,8	↓	3,5
	Japão		6,6		6,6		4,4		4,7		4,3		4,5		4,1		4,4		4,7		5		2,8		2,7
Balanced accuracy (%)	Comprasnet	↓	49,7	↓	49,5	↓	61,1	↓	61	↓	61,1	↓	60	↓	58,1	↓	57,7	↓	62,3	↓	59,8	↓	49,2	↓	67
	Japão		67,6		67,9		79,8		78,7		79,3		78,6		80,4		79,2		76,3		75,3		82,9		83,1

Cenário 1 (todos os campos) e 2 (todos os campos + screens) - Algoritmos

		K Neighbors				MLP				Bernoulli Naive Bayes				Gaussian Naive Bayes				Gaussian Process			
Métricas	Dataset\Cenário	1		2		1		2		1		2		1		2		1		2	
Accuracy(%)	Comprasnet	↑	94,5	↑	94,5	↑	93,9	↑	93,6	↑	93,6	↑	91,8	↓	84,3	↓	69	↑	92,4	↑	93,1
	Japão		92,5		92,5		88,7		88,8		88,7		88,6		94,6		94,6		89,5		88,9
False Positives (FP) (%)	Comprasnet	↑	0,4	↑	0,3	↑	0	●	0	●	0	↓	2	↓	10,3	↓	26,6	↓	0,1	↓	0,2
	Japão		3,7		3,6		0,1		0		0		0		0,7		0,7		0		0
False Negatives (FN) (%)	Comprasnet	↓	5,1	↓	5,2	↑	6,1	↑	6,4	↑	6,4	↑	6,2	↓	5,5	↑	4,4	↑	7,4	↑	6,7
	Japão		3,8		3,9		11,3		11,2		11,3		11,4		4,7		4,7		10,5		11,1
Balanced accuracy (%)	Comprasnet	↓	59,7	↓	58,6	●	50	↓	50	●	50	↑	50,4	↓	49,5	↓	50,2	↑	51,2	↑	51,2
	Japão		80,7		80,7		50		50,1		50		50,1		78,3		78,7		50		50

Figura 5.1: Métricas do experimento para os cenários 1 (todos os campos) e 2 (todos os campos + *screens*), comparando o desempenho entre os *datasets* do Comprasnet e Japão.

De modo geral, considerando todos os cenários, os melhores algoritmos foram: *Extra Trees*, *Random*

Cenário 3 (campos comuns) e 4 (campos comuns + screens) - Algoritmos

		SGD		Extra Trees		Random Forest		Ada Boost		Gradient Boosting		SVC	
Métricas	Dataset\Cenário	3	4	3	4	3	4	3	4	3	4	3	4
Accuracy(%)	Comprasnet	93,4	88,1	94,8	94,6	94,2	93,8	94,4	93,5	93	93,6	90,7	79,2
	Japão	83,9	83,7	94,5	94,5	93,2	93,4	93,3	92,3	90,7	87,9	85,5	82,5
False Positives (FP) (%)	Comprasnet	0,2	5,6	0,3	0,2	0,8	0,8	0,6	1,2	0,4	1,3	5,4	17,3
	Japão	8,2	8,2	1,1	0,6	2,5	2,2	2,2	3	5,3	7,7	11,5	14,5
False Negatives (FN) (%)	Comprasnet	6,4	6,2	4,9	5,2	5	5,4	4,9	5,3	6,6	5,1	3,9	3,5
	Japão	7,9	8,1	4,5	4,9	4,3	4,3	4,5	4,7	4	4,4	3	3
Balanced accuracy (%)	Comprasnet	49,9	50	59,1	58,4	59,5	59	58,9	58,5	58,3	61,3	59,6	65,2
	Japão	60,5	59,6	79,6	78,2	79,5	79,4	78,9	77,3	77,4	75,8	80,2	78,4

Cenário 3 (campos comuns) e 4 (campos comuns + screens) - Algoritmos

		K Neighbors		MLP		Bernoulli Naive		Gaussian Naive		Gaussian Process	
Métricas	Dataset\Cenário	3	4	3	4	3	4	3	4	3	4
Accuracy(%)	Comprasnet	94,4	94,7	93,7	93,7	93,9	92,9	93,4	66,6	92,3	95,5
	Japão	92,3	92,4	88,2	88,7	88,8	88,8	94	94,3	88,7	88,9
False Positives (FP) (%)	Comprasnet	0,4	0,4	0	0	0	1	0,8	29,3	0,3	0,2
	Japão	3,7	3,6	0	0	0	0	0,3	0,4	0	0
False Negatives (FN) (%)	Comprasnet	5,2	4,9	6,3	6,3	6,1	6,1	5,8	4,1	7,5	4,2
	Japão	4,1	3,9	11,2	11,3	11,2	11,2	5,6	5,4	11,3	11,1
Balanced accuracy (%)	Comprasnet	59	59,3	50	50	50	50,8	49,6	51,5	53,5	53,7
	Japão	80,2	80,5	50,1	50	50	50	74,4	75,7	50,2	50

Figura 5.2: Métricas do experimento para os cenários 3 (campos comuns) e 4 (campos comuns + *screens*), comparando o desempenho entre os *datasets* do Comprasnet e Japão.

Forest, *Ada Boost* e *Gradient Boosting*. Mas em cenários isolados, outros algoritmos obtiveram melhores resultados, como o caso do *Gaussian Process* no cenário 4. Em menos da metade dos casos as *screens* propiciaram um melhor desempenho dos algoritmos, tanto na acurácia quanto na taxa de *FP* e *FN*.

Em quase todos os cenários, podemos observar uma elevação dos índices de acurácia e melhora da taxa de *FP* na comparação do *dataset* do Comprasnet com o do Japão. Isso não ocorre quando analisamos a taxa de FN e a acurácia balanceada. Mesmo assim, para alguns cenários e algoritmos, identificamos a melhora ou estabilização de todas as métricas, como o algoritmo MLP no cenário 4.

Especificamente em relação às *screens* propostas por Faria *et al.* [61], ao compararmos os cenários 1 e 3, em menos da metade dos algoritmos as métricas foram melhoradas por elas. Assim, a contribuição dessas variáveis indica um menor impacto na detecção de possível conluio. No entanto, a ausência de dados refinados na base do Comprasnet para o cálculo preciso destes índices pode ter interferido na sua verificação. O cálculo da especificidade e da frequência dos itens licitados não foi suficientemente preciso nos dados acessados, pois os materiais licitados estavam indicados de maneira abrangente, sem indicar o tipo de material na sua especificação mais detalhada. Isto interferiu na precisão ao determinar se um dado item da licitação correspondia especificamente a um item já licitado anteriormente, pois o código do item foi disponibilizado de forma geral.

Analisando a acurácia balanceada, observamos que os melhores resultados para o *dataset* do Japão estão em torno de 80%. Já para o *dataset* do Comprasnet, os melhores resultados são consideravelmente inferiores, em torno de 60%. Esta diferença acentuada pode ter origem no forte desbalanceamento do *dataset* do Comprasnet, com apenas 6,26% de lances colusivos, contra 8,09% do *dataset* do Japão.

Diferentemente da estratégia de classificação adotada nos *datasets* do estudo de García Rodríguez *et al.* [50], onde todos os lances de um item licitado foram marcados como colusivos se um determinado percentual de lances fosse efetivamente identificado como tal nas investigações realizadas, somente marcamos

como colusivos os lances das empresas punidas por conluio. Assumimos, portanto, que os demais licitantes do item apresentaram lances competitivos. Esta estratégia de marcação restringiu o número de lances assumidos como colusivos, o que pode justificar a acentuada diferença da acurácia balanceada obtida neste experimento.

Com relação às métricas *Precision*, *Recall* e *F1 score*, a Figura 5.3 possibilita a comparação do desempenho dos melhores algoritmos do cenário 4, onde os campos comuns aos *datasets* e *screens* estão presentes, para os *datasets* do Comprasnet e Japão. Este cenário foi escolhido, por excluir os campos exclusivos dos cenários 1 e 2, que não contribuíram efetivamente para a identificação de conluio, mas ainda incluir as *screens*, possibilitando a comparação com o melhor cenário do *dataset* do Japão. As métricas foram representadas por uma cruz para cada algoritmo, onde o ponto de encontro representa a mediana da *Precision* e *Recall*. As extremidades das cruzes representam os mínimos e máximos obtidos para estas métricas em cada uma das 50 iterações dos algoritmos no experimento. Por fim, os valores das 3 métricas relacionadas no gráfico, pertencem ao interior do retângulo formado pela cruz de cada algoritmo. Os algoritmos que não geraram uma cruz no gráfico, foram suprimidos por gerarem valores baixos para as métricas nos *datasets* do Comprasnet (<5%) e Japão (< 50%).

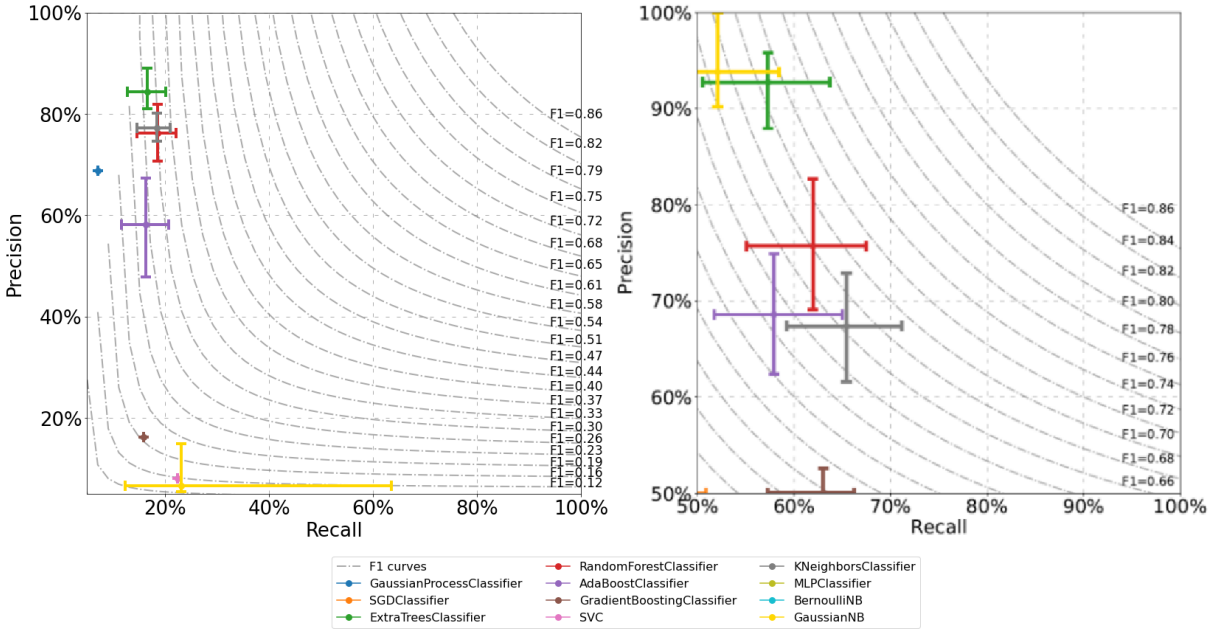

Figura 5.3: Comparação das métricas *Recall*, *Precision* e *F1 score* entre os *datasets* do Comprasnet (esquerda) e Japão (direita) para o cenário 4.

Identificamos que os algoritmos *Extra Trees*, *Random Forest* e *Ada Boost*, baseados em *ensemble methods*, considerando a métrica *Precision*, obtiveram valores entre 50% e 90% aproximadamente, mais próximos aos obtidos por García Rodríguez *et al.* [50]. Isso não foi observado para as métricas *Recall* e, consequentemente, *F1 score*. Estes indicadores com resultados piores podem estar associados à estratégia de marcação de conluio utilizada no *dataset* gerado. Outra possível justificativa para os índices piores neste caso, assim como observado por García Rodríguez *et al.* [50] no *dataset* dos Estados Unidos, é o baixo percentual de empresas colusivas obtido no experimento: 4,76% (7 colusivas em 147 empresas).

5.2 *DATASET "ALL+COMPRASNET"*

A comparação das métricas *Accuracy*, *FP*, *FN* e *Balanced Accuracy*, obtidas no processamento do *dataset "All+Comprasnet"*, com as métricas correspondentes do *"All datasets"*, possibilita a análise do desempenho dos algoritmos nos cenários 3 e 4 apresentados. A Figura 5.4 resume os dados obtidos nestes cenários para os dois *datasets*, seguindo a mesma representação apresentada anteriormente.

Cenário 3 (campos comuns) e 4 (campos comuns + screens) - Algoritmos

		SGD		Extra Trees		Random Forest		Ada Boost		Gradient Boosting		SVC	
Métricas	Dataset\Cenário	3	4	3	4	3	4	3	4	3	4	3	4
Accuracy(%)	All+Comprasnet	52,30	51,50	83,50	87,30	82,70	85,50	82,00	82,90	79,60	84,90	59,00	54,80
	All datasets	48,70	48,50	82,00	86,30	80,50	84,00	81,60	81,80	75,60	72,00	48,10	47,80
False Positives (FP) (%)	All+Comprasnet	20,5	24,3	8,6	6,9	8,5	7,4	9,6	8,9	11,2	8,7	12,3	14,6
	All datasets	25,2	24,8	9,7	8	9,7	8,7	10,2	9,9	9,7	9,6	42,3	45,8
False Negatives (FN) (%)	All+Comprasnet	27,1	24,2	7,9	5,8	8,8	7,1	8,4	8,2	9,2	6,4	28,8	30,5
	All datasets	26,1	26,7	8,2	5,7	9,8	7,3	8,2	8,3	14,7	18,4	9,6	6,4
Balanced accuracy (%)	All+Comprasnet	49,1	49,8	83	87,1	81,9	85	81,5	82,4	79,2	84,3	54,5	48,6
	All datasets	48,4	48,1	82,1	86,4	80,4	84	81,7	81,8	75,1	70,8	49,7	50,7

Cenário 3 (campos comuns) e 4 (campos comuns + screens) - Algoritmos

		K Neighbors		MLP		Bernoulli Naive Bayes		Gaussian Naive Bayes		Gaussian Process	
Métricas	Dataset\Cenário	3	4	3	4	3	4	3	4	3	4
Accuracy(%)	All+Comprasnet	63,50	63,20	58,00	58,80	59,00	64,80	58,20	44,40	57,00	57,10
	All datasets	59,20	59,50	52,50	52,60	53,70	58,80	53,60	53,10	53,30	52,60
False Positives (FP) (%)	All+Comprasnet	15,9	16,5	7,5	5,1	0	10,6	2,7	46,7	1,9	1,3
	All datasets	18,5	18,6	22,7	21,8	0	24,4	10,5	3,9	1,3	1,4
False Negatives (FN) (%)	All+Comprasnet	20,6	20,3	34,4	36,1	41	24,7	39,1	8,9	41,1	41,6
	All datasets	22,3	21,9	24,8	25,6	46,3	16,7	35,9	43	45,6	46
Balanced accuracy (%)	All+Comprasnet	61,2	61,2	52,1	51,4	50	60,4	49	49,8	48,9	49,2
	All datasets	58,7	59,1	53	53	50	59,3	51,6	50,1	49,3	49,1

Figura 5.4: Métricas do experimento para os cenários 3 (campos comuns) e 4 (campos comuns + *screens*) para o *dataset "All+Comprasnet"*.

De forma geral, a expansão do *dataset "All datasets"*, com os dados comuns do *dataset* Comprasnet, formando o *dataset "All+Comprasnet"*, melhorou as métricas na maioria dos cenários. Para os algoritmos baseados em *ensemble methods*, que estão entre os que obtiveram a melhor acurácia também neste caso, apenas o *Extra Trees* e *Ada Boost* registraram quedas em até duas métricas (*FN* e *Balanced Accuracy*). Mesmo assim, considerando todas as métricas, o melhor desempenho foi do algoritmo *Extra Trees* no cenário 4, onde os campos comuns e as *screens* estão presentes. A interferência positiva das *screens* neste cenário, pode ser observada na comparação com o cenário 3, assim como no estudo base.

As métricas *Precision*, *Recall* e *F1 score*, estão consolidadas na Figura 5.5 e possibilitam a comparação do desempenho dos melhores algoritmos do cenário 4, onde os campos comuns aos *datasets* e *screens* estão presentes, para os *datasets "All+Comprasnet"* e *"All datasets"*. Esta Figura segue a mesma forma de representação detalhada anteriormente, utilizada na Figura 5.3.

Assim como no processamento do *dataset* do Comprasnet, os algoritmos baseados em *ensemble methods*, também apresentaram o melhor desempenho para o *dataset "All+Comprasnet"*, considerando as métricas *Recall*, *Precision* e *F1 score*. O melhor desempenho também se mantém para o algoritmo *Extra Trees*, com os valores de *Precision* e *Recall* entre 80% e 90% aproximadamente, equivalentes aos obtidos por García Rodríguez *et al.* [50] no cenário 4. Consequentemente, a métrica *F1 score*, também possui valores próximos, variando entre 0,84 e 0,86.

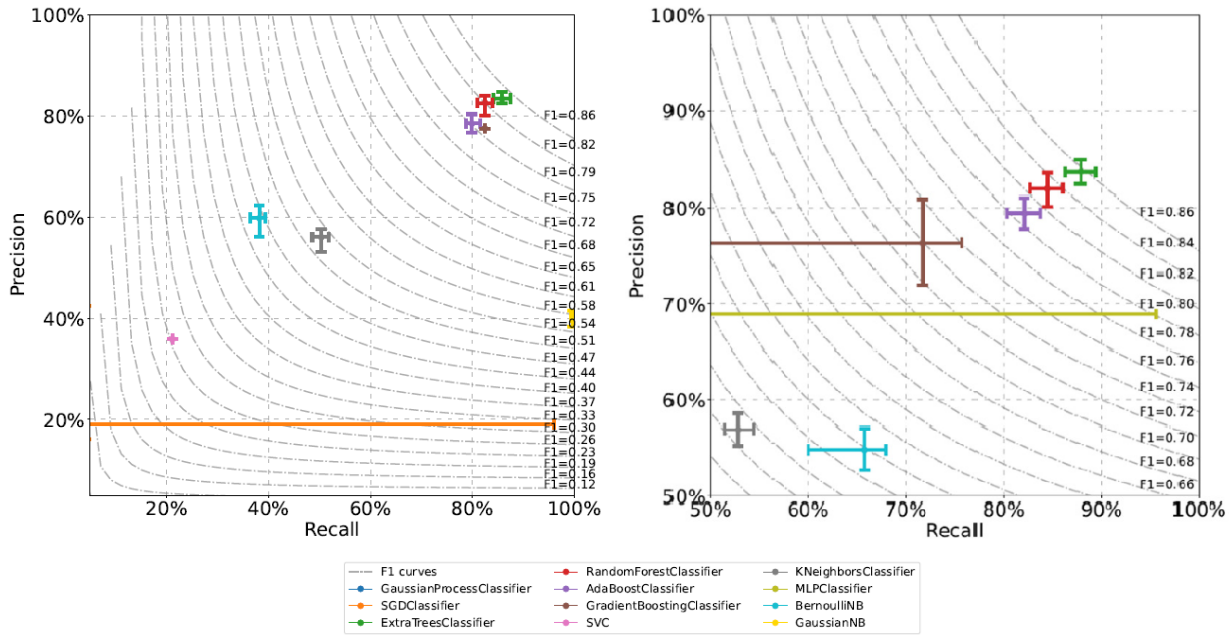

Figura 5.5: Comparação das métricas *Recall*, *Precision* e *F1 score* entre os *datasets "All+Comprasnet"* (esquerda) e *"All datasets"* (direita) para o cenário 4.

5.3 GERAÇÃO DE ALERTAS DE CONLUIO

Para o *dataset* do Comprasnet, os algoritmos baseados em *ensemble methods* obtiveram resultados semelhantes. Já para o *dataset "All+Comprasnet"*, um algoritmo desse mesmo grupo, o *Extra Trees*, obteve um desempenho destacado, alcançando uma acurácia de 87,3%, *FP* de 6,9%, *FN* de 5,8% e *Balanced Accuracy* de 87,1% no cenário 4. Também adotaremos este algoritmo para o *dataset* do Comprasnet no cenário 4, onde obteve acurácia de 94,6%, *FP* de 0,2%, *FN* de 5,2% e *Balanced Accuracy* de 58,4%. Dessa forma, o *Extra Trees* pode ser destacado como o algoritmo de melhor desempenho, considerando as métricas obtidas no processamento dos *datasets* Comprasnet e *"All+Comprasnet"*.

Os modelos gerados a partir do treinamento do algoritmo *Extra Trees* no cenário 4, para ambos os *datasets* (Comprasnet e *"All+Comprasnet"*), foram aplicados sobre um conjunto novo de pregões do Comprasnet, realizados entre 2020 e 2022. Esta verificação foi realizada no formato de uma nova trilha de auditoria, capaz de gerar alertas da ALICE aos auditores, ao identificar possível conluio entre participantes da disputa de um item do pregão analisado.

Para o modelo treinado a partir do *dataset* do Comprasnet, aplicado no referido período, não foram detectados novos possíveis conluios. Este resultado pode estar relacionado à incapacidade do modelo de detectar padrões semelhantes, indicando um *overfitting* no treinamento, bem como na ausência de uma maior variabilidade de exemplos de conluio no *dataset*. Um indício disso é a alta taxa de acurácia com baixos *FP* e *FN*. A ausência de dados mais precisos na caracterização dos itens licitados foi uma das dificuldades enfrentadas na formação do *dataset*.

Já a verificação de conluio com o modelo treinado a partir do *dataset "All+Comprasnet"*, resultou na identificação de outros possíveis casos de conluio, totalizados na Tabela 5.1. A composição desse *dataset*

abrange uma maior variabilidade de casos de conluio, aumentando a capacidade do algoritmo de identificar casos semelhantes.

Tabela 5.1: Total de alerta de conluio gerado no Comprasnet.

Ano	**Total de pregões analisados**	**Pregões com alerta de conluio**	**Percentual com alerta de conluio**	**Pregões com alerta de conluio + outros alertas**	**Percentual com alerta de conluio + outros alertas**
2020	12.382	678	5,48%	70	0,57%
2021	19.661	1.798	9,15%	113	0,57%
2022	18.483	1.584	8,57%	164	0,88%

O percentual de alertas gerados nos pregões analisados, mesmo abaixo de 10%, indica um grande volume de pregões sob suspeita de conluio. Caso a suspeita não se confirme, teríamos uma alta taxa de *FP*. Ainda assim, caso a quantidade de alertas seja alta, gerando uma demanda de análise maior que a capacidade dos auditores, o alerta pode ser associado à presença de outros alertas gerados pela ferramenta. Ou seja, o alerta de conluio, combinado com outros alertas de irregularidades já existentes, aumentam a possibilidade de que realmente exista um ilícito em andamento no pregão em questão. Esta estratégia de priorização, para o período verificado, reduziria a quantidade de pregões suspeitos para menos de 1%, aumentando a possibilidade de aprofundamento nas investigações por parte dos auditores na confirmação da suspeita de conluio entre os licitantes de um pregão.

Outra possível estratégia de priorização dos alertas está relacionada à existência de vínculos entre as empresas apontadas como colusivas na disputa de um item do pregão. Realizamos estas análises a partir do banco de vínculos Yggdrasil, disponível na CGU. Partindo do CNPJ das empresas e do CPF de pessoas relacionadas a elas, identificamos um total de 75 pregões com alerta de possível conluio na disputa dos seus itens, cujas empresas apontadas por tal prática estão relacionadas entre si. A distribuição deste total de alertas por pregão e empresas com vínculos entre si, que participaram da disputa no período verificado, está na Tabela 5.2.

Tabela 5.2: Total de pregões com alerta de possível conluio e percentual destes pregões onde as empresas estão vinculadas entre si.

Ano	Total de pregões com alerta de conluio	Pregões com alerta de conluio e vínculo entre as empresas apontadas
2020	678	8 (1,18%)
2021	1.798	40 (2,22%)
2022	1.584	27 (1,70%)

Um exemplo real deste período com vínculo entre as empresas participantes da disputa de um item do pregão com alerta de possível conluio, está esquematizado na Figura 5.6. Neste exemplo, os vértices indicam as pessoas físicas e jurídicas. Já as arestas direcionais, representam as ligações (ou tipo de vínculos) entre elas. Os dados reais foram trocados por rótulos para preservar a identidade dos envolvidos no cenário. A empresa representada pelo CNPJ-1 é a vencedora do item do pregão, por apresentar o menor preço. Esta empresa é representada por dois sócios: a PESSOA-1 e a PESSOA-2. No entanto, a PESSOA-2 é sócia da empresa CNPJ-2, também participante da disputa de tal item do pregão. Ainda, a PESSOA-1 é pai da

PESSOA-3, que é sócia da empresa CNPJ-3, que também ofertou lances no item. Este cenário de disputa, isoladamente, não apresenta uma ilegalidade por si só. Mas, aliado ao indicativo de conluio, ele desperta o questionamento da boa-fé das empresas, supondo um provável conluio na disputa, praticando cobertura de lances.

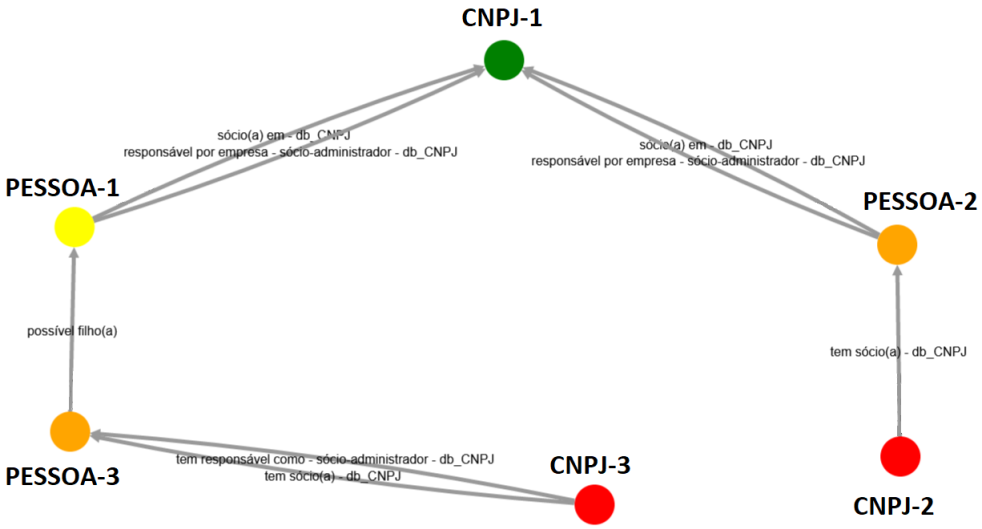

Figura 5.6: Relação identificada entre empresas participantes de um pregão cujo item possui alerta de conluio, indicado pelo modelo treinado com o *Extra Trees* e o *dataset "All+Comprasnet"*.

Esta evidência de vínculos entre as empresas, reforça a capacidade de detecção do modelo treinado neste trabalho, trazendo um elemento que pode justificar a suspeita de combinação de preços alertada. Por outro lado, a falta de vínculo direto entre as empresas apontadas por conluio na disputa de um item, não significa necessariamente um provável erro na classificação. A ausência de vínculos identificados pode estar associada às limitações das bases de dados do governo federal que compõem o Yggdrasil. Ainda, pode ser justificada pela inexistência de vínculos diretos entre as empresas, que combinaram preços para se beneficiarem com uma possível divisão de mercado.

6 CONCLUSÃO

Detectar a atuação de cartéis em licitações é importante pelo seu potencial de prejudicar a competitividade e a transparência do processo de licitação. Isso pode resultar em prejuízos para o setor público e para a sociedade em geral, já que os cartéis podem levar a maiores custos e menor qualidade nos produtos e serviços adquiridos com a execução de licitações.

É importante registrar que o uso do aprendizado de máquina para detectar cartéis em licitações é apenas uma ferramenta que pode ser usada como parte de um processo mais amplo de investigação e análise. É crucial que as autoridades responsáveis pelas licitações também utilizem outros métodos e fontes de informação para garantir que estes processos sejam justos e transparentes.

A escassez de dados confiáveis e completos, a dificuldade em identificar padrões que indiquem a existência de um cartel e a complexidade do processo de licitação, tornam a detecção de cartel em licitações um grande desafio. Portanto, a análise final de um auditor para confirmar ou descartar qualquer apontamento de possível conluio realizado pelos modelos de AM é fundamental.

Os algoritmos de AM baseados em *ensemble methods* obtiveram um desempenho expressivo no experimento realizado, indicando como promissora a identificação de conluio desejada neste trabalho. A acurácia alcançada nos 4 cenários para o *dataset* do Comprasnet foi maior que 90%, além da baixa taxa de *FP*, próxima a 1%, e *FN*, próxima a 5%. Porém, os valores inferiores da métrica *Balanced Accuracy*, podem indicar forte desbalanceamento no *dataset*. Já para o *dataset "All+Comprasnet"*, a acurácia obtida, variando entre 82% e 87,30%, foi maior do que as obtidas no processamento do *"All datasets"*. As métricas *FP*, *FN*, e *Ballanced accuracy*, com poucas exceções, foram melhoradas também. Estes resultados são compatíveis com os obtidos por García Rodríguez *et al.* [50] para este grupo de algoritmos.

Também alinhado aos resultados obtidos por García Rodríguez *et al.* [50], considerando as métricas *Precision*, *Recall* e *F1 score* para o *dataset* do Comprasnet, onde o algoritmo *Gradient Boosting*, obteve baixo desempenho entre os algoritmos baseados em *ensemble methods*, destacamos os três algoritmos com melhor desempenho, considerando todas as métricas adotadas: *Extra Trees*, *Random Forest* e *Ada Boost*. Com relação ao *dataset "All+Comprasnet"*, este mesmo grupo de algoritmos obteve resultados equivalentes ao processamento do *"All datasets"*, com destaque para o algoritmo *Extra Trees* no cenário 4.

Dessa forma, considerando todas as métricas utilizadas, concluímos que o *Extra Trees* obteve melhor desempenho na identificação de possível conluio entre os participantes de pregões. Em teste experimental dos modelos selecionados, este algoritmo foi capaz de identificar pregões suspeitos em um conjunto de dados não utilizado no treinamento. Os alertas gerados com o modelo do cenário 4 do *dataset "All+Comprasnet"*, seguindo o mesmo mecanismo de alerta da ALICE, necessitam de confirmação pelos auditores da ferramenta. De todo modo, a geração deste alerta caracteriza um avanço da identificação de conluio, ao identificar padrões semelhantes a casos confirmados de formação de cartel na disputa entre licitantes.

As *screens* influenciaram positivamente as métricas obtidas no cenário 4 do *dataset "All+Comprasnet"*. No entanto, as variáveis Total de lances, Quantidade, Escore de especificidade e Escore de frequência, pro-

postas no trabalho de Faria *et al.* [61], tratadas neste experimento como possíveis novas *screens* para aumentar a capacidade de detecção de conluio pelos algoritmos testados, não produziram efeitos significativos na comparação do cenário 1 com o cenário 3. Mesmo assim, consideramos a inclusão destas variáveis no experimento como uma contribuição obtida. Em trabalhos futuros, estas variáveis podem ter o seu cálculo aprimorado, aumentando a precisão da identificação de materiais equivalentes em outras licitações do Comprasnet.

O *dataset* com possíveis lances colusivos extraídos do Comprasnet, apresentados por empresas punidas com base na Lei 12.529/2011, também pode ser considerado uma contribuição obtida com o experimento, dada a escassez de *datasets* para este tipo de estudo [50]. No entanto, a estratégia empírica de marcação dos lances colusivos é uma possível fragilidade e demanda a confirmação da capacidade de treinar um modelo para a detecção efetiva de indícios de conluio entre os participantes de outros pregões realizados no Comprasnet.

A expansão do *"All datasets"*, incluindo os campos comuns do *dataset* do Comprasnet, formando o *"All+Comprasnet"* também pode ser contabilizado como um avanço, considerando a melhoria das métricas utilizadas. Mesmo assim, em trabalho futuro, o modelo gerado pode ser melhorado com a inclusão de novos casos obtidos a partir da confirmação ou refutação dos alertas analisados pelos auditores.

Outra possível linha de investigação para trabalhos futuros é a mudança na estratégia de marcação de conluio na geração do *dataset*, assumindo todos os lances de um item como colusivos quando um licitante punido estiver na disputa. Esta estratégia visa diminuir o desbalanceamento do *dataset* e, consequentemente, aumentar a acurácia balanceada obtida neste experimento.

REFERÊNCIAS BIBLIOGRÁFICAS

1 HARRINGTON, P. *Machine learning in action*. [S.l.]: Simon and Schuster, 2012.

2 OCDE. *Public procurement*. 2022. Disponível em: <https://www.oecd.org/gov/public-procurement/>. Acesso em: 21 novembro 2022.

3 RIBEIRO, C. G.; JR, E. I.; RAUEN, A. T.; LI, Y. Unveiling the public procurement market in brazil: A methodological tool to measure its size and potential. *Development Policy Review*, Wiley Online Library, v. 36, p. O360–O377, 2018.

4 GROUP, W. B. *A Fair Adjustment: Efficiency and Equity of Public Spending in Brazil*. [S.l.]: World Bank Group, 2017. Disponível em: <https://documents.worldbank.org/en/publication/documents-reports/documentdetail/643471520429223428/volume-1-overview>. Acesso em: 21 novembro 2022.

5 (CGU), B. C.-G. da U. *Licitações com contratação realizada*. 2022. Disponível em: <https://www.portaltransparencia.gov.br/licitacoes>. Acesso em: 17 novembro 2022.

6 BRASIL. Lei nº 10.520, de 17 de julho de 2002. *Diário Oficial da União*, Brasília, DF, 2002. ISSN 1677-7042. Disponível em: <https://www.planalto.gov.br/ccivil_03/leis/2002/l10520.htm>.

7 BRASIL. Decreto nº 10.024, de 20 de setembro de 2019. *Diário Oficial da União*, Brasília, DF, 2019. ISSN 1677-7042. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2019-2022/2019/decreto/D10024.htm>.

8 BRASIL. Lei nº 13.844, de 18 de junho de 2019. *Diário Oficial da União*, Brasília, DF, 2019. ISSN 1677-7042. Disponível em: <http://www.planalto.gov.br/ccivil_03/_Ato2019-2022/2019/Lei/L13844.htm>.

9 (CGU), B. C.-G. da U. *Relatório de Gestão: exercício 2020*. 2020. Disponível em: <https://repositorio.cgu.gov.br/handle/1/65266>. Acesso em: 16 novembro 2022.

10 DANTAS, D. d. Q.; MARTINS, L. B. *Ferramenta Alice: estudo sobre sua eficácia e eficiência no uso dessa ferramenta como fundamento para a prevenção e o combate à corrupção no âmbito da Controladoria-Geral da União*. [S.l.]: Universidade Católica de Brasília, 2022. Disponível em: <https://repositorio.cgu.gov.br/handle/1/68888>. Acesso em: 16 novembro 2022.

11 ROCHA, A. L. M. d. *Ferramenta Alice: Auditoria Preventiva em Licitações. Fórum: O Controle no Combate à Corrupção/2019*. 2019. Disponível em: <https://repositorio.cgu.gov.br/handle/1/43580>. Acesso em: 17 junho 2022.

12 PANIS, A. d. C. Inovação em compras públicas: estudo de caso do robô alice da controladoria-geral da união (cgu). 2020.

13 ORGANISATION FOR ECONOMIC CO-OPERATION AND DEVELOPMENT. *Fighting Hard-core Cartels Harm, Effective Sanctions and Leniency Programmes*. Paris: OECD Publishing, 2002. ISBN 1-280-08111-2. Disponível em: <https://www.oecd.org/competition/cartels/1841891.pdf>.

14 AGUILERA, R. C. A. O. de. Corruption perception index: Reflections in brazil before and during operation car wash. *Contextualizaciones Latinoamericanas*, v. 2, n. 21, 2019.

15 CADE. *Guia de Combate a cartéis em licitação*. [S.l.], 2019. Disponível em: <https://cdn.cade.gov.br/Portal/centrais-de-conteudo/publicacoes/guias-do-cade/guia-de-combate-a-carteis-em-licitacao-versao-final-1.pdf>.

16 BERRU, Y. T.; BATISTA, V. F. L.; TORRES-CARRIÓN, P.; JIMENEZ, M. G. Artificial intelligence techniques to detect and prevent corruption in procurement: A systematic literature review. In: BOTTO-TOBAR, M.; VIZUETE, M. Z.; TORRES-CARRIÓN, P.; LEÓN, S. M.; VÁSQUEZ, G. P.; DURAKOVIC, B. (Ed.). *Applied Technologies*. Cham: Springer International Publishing, 2020. p. 254–268. ISBN 978-3-030-42520-3.

17 DESORDI, D.; BONA, C. D. A inteligência artificial ea eficiência na administração pública. *Revista de Direito*, Universidade Federal Viçosa, v. 12, n. 2, p. 1–22, 2020.

18 BASAVARAJU, A.; DU, J.; ZHOU, F.; JI, J. A machine learning approach to road surface anomaly assessment using smartphone sensors. *IEEE Sensors Journal*, PP, p. 1–1, 11 2019.

19 FACELI, K.; LORENA, A. C.; GAMA, J.; CARVALHO, A. C. P. d. L. F. d. *Inteligência artificial: uma abordagem de aprendizado de máquina*. [S.l.]: LTC, 2011.

20 NEUPERT, T.; FISCHER, M. H.; GREPLOVA, E.; CHOO, K.; DENNER, M. M. Introduction to machine learning for the sciences. *arXiv e-prints*, p. arXiv–2102, 2021.

21 MAHESH, B. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, v. 9, p. 381–386, 2020.

22 KOTSIANTIS, S. Supervised machine learning: A review of classification techniques. *Informatica*, v. 31, p. 249–268, 2007.

23 MAHADEVKAR, S. V.; KHEMANI, B.; PATIL, S.; KOTECHA, K.; VORA, D. R.; ABRAHAM, A.; GABRALLA, L. A. A review on machine learning styles in computer vision—techniques and future directions. *IEEE Access*, v. 10, p. 107293–107329, 2022.

24 SAH, S. Machine learning: a review of learning types. Preprints, 2020.

25 ALTMAN, N. S. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, Taylor & Francis, v. 46, n. 3, p. 175–185, 1992.

26 FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, Elsevier, v. 55, n. 1, p. 119–139, 1997.

27 FRIEDMAN, J. H. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, JSTOR, p. 1189–1232, 2001.

28 BREIMAN, L. Random forests. *Machine learning*, Springer, v. 45, n. 1, p. 5–32, 2001.

29 GEURTS, P.; ERNST, D.; WEHENKEL, L. Extremely randomized trees. *Machine learning*, Springer, v. 63, n. 1, p. 3–42, 2006.

30 CORTES, C.; VAPNIK, V. Support-vector networks. *Machine learning*, Springer, v. 20, n. 3, p. 273–297, 1995.

31 ZHANG, T. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In: *Proceedings of the twenty-first international conference on Machine learning*. [S.l.: s.n.], 2004. p. 116.

32 HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. H.; FRIEDMAN, J. H. *The elements of statistical learning: data mining, inference, and prediction*. [S.l.]: Springer, 2009. v. 2.

33 DICIONáRIO Brasileiro da Língua Portuguesa. [S.l.]: Editora Melhoramentos, 2023. Disponível em: <https://michaelis.uol.com.br/busca?r=0&f=0&t=0&palavra=conluio>. Acesso em: 12 maio 2023.

34 OCDE. *Fighting Bid Rigging in IMSS Procurement: Impact of OECD Recommendations*. 2018. 1–256 p. Disponível em: <https://www.oecd.org/daf/competition/IMSS-procurement-impact-OECD-recommendations2018-ENG.pdf>. Acesso em: 28 novembro 2022.

35 ALGORITHMS, O. *Collusion: Competition Policy in the Digital Age*. [S.l.]: OECD: Paris, France, 2017. Disponível em: <https://www.oecd.org/daf/competition/Algorithms-and-colllusion-competition-policy-in-the-digital-age.pdf>. Acesso em: 28 novembro 2022.

36 BRASIL. Lei nº 12.529, de 30 de novembro de 2011. *Diário Oficial da União*, Brasília, DF, 2011. ISSN 1677-7042. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2011-2014/2011/lei/l12529.htm>.

37 BRASIL. Lei nº 8.137, de 27 de dezembro de 1990. *Diário Oficial da União*, Brasília, DF, 1990. ISSN 1677-7042. Disponível em: <https://www.planalto.gov.br/ccivil_03/leis/l8137.htm>.

38 BRASIL. Lei nº 14.133, de 1º de abril de 2021. *Diário Oficial da União*, Brasília, DF, 2021. ISSN 1677-7042. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2019-2022/2021/lei/l14133.htm>.

39 PORTER, R. H.; ZONA, J. D. Detection of bid rigging in procurement auctions. *Journal of political economy*, The University of Chicago Press, v. 101, n. 3, p. 518–538, 1993.

40 BLUME, A.; HEIDHUES, P. Modeling tacit collusion in auctions. *Journal of Institutional and Theoretical Economics (JITE)/Zeitschrift für die gesamte Staatswissenschaft*, JSTOR, p. 163–184, 2008.

41 HU, A.; OFFERMAN, T.; ONDERSTAL, S. Fighting collusion in auctions: An experimental investigation. *International Journal of Industrial Organization*, Elsevier, v. 29, n. 1, p. 84–96, 2011.

42 HENDRICKS, K.; PORTER, R.; TAN, G. Bidding rings and the winner's curse. *The RAND Journal of Economics*, Wiley Online Library, v. 39, n. 4, p. 1018–1041, 2008.

43 BAJARI, P.; SUMMERS, G. Detecting collusion in procurement auctions. *Antitrust LJ*, HeinOnline, v. 70, p. 143, 2002.

44 PESENDORFER, M. A study of collusion in first-price auctions. *The Review of Economic Studies*, Wiley-Blackwell, v. 67, n. 3, p. 381–411, 2000.

45 MCAFEE, R. P.; MCMILLAN, J. Bidding rings. *The American Economic Review*, JSTOR, p. 579–599, 1992.

46 AOYAGI, M. Bid rotation and collusion in repeated auctions. *Journal of economic Theory*, Elsevier, v. 112, n. 1, p. 79–105, 2003.

47 SKRZYPACZ, A.; HOPENHAYN, H. Tacit collusion in repeated auctions. *Journal of Economic Theory*, Elsevier, v. 114, n. 1, p. 153–169, 2004.

48 NAI, R.; SULIS, E.; MEO, R. Public procurement fraud detection and artificial intelligence techniques: a literature review. 2022.

49 MODRUšAN, N.; RABUZIN, K.; MRšIC, L. Review of public procurement fraud detection techniques powered by emerging technologies. *International Journal of Advanced Computer Science and Applications*, The Science and Information Organization, v. 12, n. 2, 2021. Disponível em: <http://dx.doi.org/10.14569/IJACSA.2021.0120272>.

50 García Rodríguez, M. J.; RODRíGUEZ-MONTEQUíN, V.; BALLESTEROS-PéREZ, P.; LOVE, P. E.; SIGNOR, R. Collusion detection in public procurement auctions with machine learning algorithms. *Automation in Construction*, v. 133, p. 104047, 2022. ISSN 0926-5805. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0926580521004982>.

51 LIMA, M. S. M.; DELEN, D. Predicting and explaining corruption across countries: A machine learning approach. *Government Information Quarterly*, v. 37, n. 1, p. 101407, 2020. ISSN 0740-624X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0740624X19302473>.

52 MAIA, P.; JR, W. M.; BARBOSA, B.; CRUZ, G. Multicriteria anomaly detection in government purchases. In: SBC. *Anais do VII Symposium on Knowledge Discovery, Mining and Learning*. [S.l.], 2019. p. 97–104.

53 LIMA, M. C. Deep vacuity: detecção e classificação automática de padrões com risco de conluio em dados públicos de licitações de obras. 2021.

54 RALHA, C. G.; SILVA, C. V. S. A multi-agent data mining system for cartel detection in brazilian government procurement. *Expert Systems with Applications*, Elsevier, v. 39, n. 14, p. 11642–11656, 2012.

55 CARVALHO, R. S.; CARVALHO, R. N. Bayesian models to assess risk of corruption of federal management units. In: CITESEER. *BMA@ UAI*. [S.l.], 2016. p. 28–35.

56 BAJARI, P.; YE, L. Deciding between competition and collusion. *Review of Economics and statistics*, MIT Press 238 Main St., Suite 500, Cambridge, MA 02142-1046, USA journals . . . , v. 85, n. 4, p. 971–989, 2003.

57 BALLESTEROS-PÉREZ, P.; SKITMORE, M.; DAS, R.; CAMPO-HITSCHFELD, M. L. del. Quick abnormal-bid-detection method for construction contract auctions. *Journal of Construction Engineering and Management*, American Society of Civil Engineers, v. 141, n. 7, p. 04015010, 2015.

58 SIGNOR, R.; BALLESTEROS-PÉREZ, P.; LOVE, P. E. Collusion detection in infrastructure procurement: A modified order statistic method for uncapped auctions. *IEEE Transactions on Engineering Management*, IEEE, 2021.

59 IMHOF, D.; KARAGÖK, Y.; RUTZ, S. Screening for bid rigging—does it work? *Journal of Competition Law & Economics*, Oxford University Press, v. 14, n. 2, p. 235–261, 2018.

60 HUBER, M.; IMHOF, D. Machine learning with screens for detecting bid-rigging cartels. *International Journal of Industrial Organization*, v. 65, p. 277–301, 2019. ISSN 0167-7187. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167718719300219>.

61 FARIA, E. R. d.; FERREIRA, M. A. M.; SANTOS, L. M. d.; SILVEIRA, S. d. F. R. Fatores determinantes na variação dos preços dos produtos contratados por pregão eletrônico. *Revista de Administração Pública*, SciELO Brasil, v. 44, n. 6, p. 1405–1428, 2010.

62 BRASIL, C. G. d. U. *Detalhamento das Sanções Vigentes*. 2022. Disponível em: <https://portaldatransparencia.gov.br/sancoes/consulta>. Acesso em: 20 maio 2022.

63 BRASIL, C. G. d. U.; BRASIL, M. d. E. M. *Acordo de Cooperação Técnica CGU/SEGES-ME Nº 135/2021*. 2021. Disponível em: <https://repositorio.cgu.gov.br/handle/1/66596>. Acesso em: 12 dezembro 2022.

64 RASMUSSEN, C. E.; WILLIAMS, C. K. I. *Gaussian processes for machine learning*. London, England: MIT Press, 2005. (Adaptive Computation and Machine Learning series).

APÊNDICES

I. CÓDIGO PARA EXTRAÇÃO DE DADOS DO COMPRASNET

```
from pathlib import Path

DAAS_COMPRASNET_DB_HOST = '' #Endereco DAAS do Comprasnet
DAAS_SIASGNET_DB_HOST = '' #Endereco DAAS do SIASGNET
DAAS_DB_USER = '' #Usuario DAAS
DAAS_DB_PASS = '' #Senha DAAS
db_comprasnet = DassDB(DAAS_COMPRASNET_DB_HOST)
db_siasgnet = DassDB(DAAS_SIASGNET_DB_HOST)

#Dataframe com todos os itens das licitacoes
dfi = None
#Arquivos csv com o dataframe
nome_arquivo = 'C:/tmp/brazilian_comprasnet.csv'

punidos = ['02293852000140', '01785999000194', '96355946000140',
           '35596501000167', '03187742000166', '62436282000121',
           '88309620000158', '01478038000137', '07386939000185',
           '11572018000184', '04717562000101', '04809827000100',
           '06332079000134', '25181298000104', '01140694000125',
           '08859696000118', '37517158000143', '03737267000154',
           '13385812000144', '02845074000154', '49254634000160',
           '44164606000138', '14744743000180', '02332985000188']

if os.path.isfile(nome_arquivo):
    dfi = pd.read_csv(nome_arquivo, index_col='Bid');
    #dfi = pd.read_csv(nome_arquivo);
else:
    #QTD_ADQUIRIDA
    itens = db_comprasnet.query(
    "SELECT p.prgCod, p.numprp, p.coduasg, pri.ipgCod, l.lanCod,  " +
    " pro.prpCNPJ, l.lanValor,  " +
    " (case when l.lanValor = pri.ipgValorMinClassif then 1 else 0 end) " +
    " as Winner, l.lanData, pri.ipgValorRef, (pri.ipgValorRef - l.lanValor) " +
    " as Difference_Bid_PTE, pri.ipgValorMinClassif, pri.ipgQuantidade, " +
    " pri.prgCod, pri.ipgItem, pri.codmat " +
    " FROM Comprasnet_VBL.tbl_Pregao p " +
    " INNER JOIN Comprasnet_VBL.tbl_pregaoitem pri ON p.prgCod = pri.prgCod " +
    " INNER JOIN Comprasnet_VBL.tbl_Lances l ON pri.ipgCod = l.ipgCod " +
    " INNER JOIN Comprasnet_VBL.tbl_Proposta pro ON pro.prgCod = pri.prgCod " +
    "   AND pro.Cliente_ID = l.cliente_id " +
    " WHERE pri.prgCod in ( " +
    "    SELECT distinct proi.prgCod " +
    "   FROM Comprasnet_VBL.tbl_Proposta proi " +
```

```
    "   WHERE proi.prpCNPJ in ( " +
    "      '02293852000140', '01785999000194', '96355946000140', " +
    "      '35596501000167', '03187742000166', '62436282000121', " +
    "      '88309620000158', '01478038000137', '07386939000185', " +
    "      '11572018000184', '04717562000101', '04809827000100', " +
    "      '06332079000134', '25181298000104', '01140694000125', " +
    "      '08859696000118', '37517158000143', '03737267000154', " +
    "      '13385812000144', '02845074000154', '49254634000160', " +
    "      '44164606000138', '14744743000180', '02332985000188')) " +
    " and ipgQuantidade > 0 " +
    " and l.lanStatus = 'V' " +
    " and pri.ipgFormaJulg = 'V' " +
    " and pri.ipgValorMinClassif is not null " +
    " order by pri.ipgCod, pri.ipgItem, l.lanValor ")
    #print('itens:')
    #print(itens)

    #forcando conversao de campos texto
    for item in itens:
        #print('Item {}: {} - (Valor: {} - Min: {})'.format(item['ipgCod'],
        #item['ipgItem'],item['lanValor'],item['ipgValorMinClassif']))
        item['prpCNPJ'] = str(item['prpCNPJ'])
        item['lanData'] = int(dateutil.parser.parse(
            item['lanData'], yearfirst=True).timestamp())
        item['lanValor'] = round(item['lanValor'],4)
        item['ipgValorMinClassif'] = round(item['ipgValorMinClassif'],4)
        item['Difference_Bid_PTE'] = round(item['Difference_Bid_PTE'],4)

    dfi = json_normalize(itens)
    dfi = dfi.rename(columns={'ipgCod': 'Tender',
                              'lanCod': 'Bid',
                              'prpCNPJ': 'Competitor',
                              'lanValor': 'Bid_value',
                              'lanData': 'Date',
                              'ipgValorRef': 'Pre_Tender'})
    dfi.set_index('Bid', inplace=True)

    dfi['atualizado'] = 0
    dfi['Number_bids'] = 0
    dfi['especificidade'] = 0
    dfi['frequencia'] = 0
    dfi['material'] = ' '
    dfi['Collusive_competitor_original'] = 0
    dfi['Collusive_competitor'] = 0

    filepath = Path(nome_arquivo)
    filepath.parent.mkdir(parents=True, exist_ok=True)
    dfi.to_csv(filepath)

prgCod = None
lances          = []
especificidades = []
```

```
96  frequencias     = []
97  materiais       = []
98  colusivo = 0
99  licitacoes = 0
100
101 for index, item in dfi.iterrows():
102     #print('Item {}: {} - ({}){}'.format(item['ipgCod'],
103     #item['ipgQuantidade'],item['prpCNPJ'],item['prpRazaoSocial']))
104
105     #Salta os registros atualizados
106     if item['atualizado'] == 1:
107         continue
108
109     #encontrou novo pregao
110     if item['prgCod'] != prgCod:
111         prgCod = item['prgCod']
112         print('LICITACAO: {}'.format(prgCod))
113
114         #QTD_LANCES
115         lances = db_comprasnet.query(
116         "SELECT pri.ipgCod, count(l.ipgCod) as qtdeLances " +
117         " FROM Comprasnet_VBL.tbl_pregaoitem pri " +
118         " INNER JOIN Comprasnet_VBL.tbl_Lances l ON pri.ipgCod = l.ipgCod " +
119         " WHERE pri.prgCod = " + str(prgCod) +
120         " AND l.lanStatus = 'V' " +
121         " GROUP BY pri.ipgCod ")
122         #print('lances:')
123         #print(lances)
124
125
126         #ESPECIFICIDADE
127         especificidades = db_comprasnet.query(
128         "SELECT pri.codmat, count(1) as especificidade " +
129         " FROM Comprasnet_VBL.tbl_pregaoitem pri " +
130         " INNER JOIN Comprasnet_VBL.tbl_pregaoitem prif ON " +
131         " pri.codmat = prif.codmat and prif.ipgCod != pri.ipgCod " +
132         " WHERE pri.prgCod = " + str(prgCod) +
133         " GROUP BY pri.codmat ")
134         #print('especificidades:')
135         #print(especificidades)
136
137
138         #FREQUENCIA
139         frequencias = db_comprasnet.query(
140         "SELECT prif.codmat, count(1) as frequencia " +
141         " FROM Comprasnet_VBL.tbl_pregaoitem pri " +
142         " INNER JOIN Comprasnet_VBL.tbl_propostaitem ppi ON " +
143         " pri.ipgCod = ppi.ipgCod and ppi.ippIndAdjudicado = 'S' " +
144         " INNER JOIN Comprasnet_VBL.tbl_proposta pp ON pp.prpCod = ppi.prpCod " +
145         " INNER JOIN Comprasnet_VBL.tbl_pregaoitem prif ON " +
146         " pri.codmat = prif.codmat and prif.ipgCod != pri.ipgCod " +
147         " INNER JOIN Comprasnet_VBL.tbl_propostaitem ppif ON " +
```

```
148         " prif.ipgCod = ppif.ipgCod and ppif.ippIndAdjudicado = 'S' " +
149         " INNER JOIN Comprasnet_VBL.tbl_proposta ppf ON " +
150         " ppf.prpCod = ppif.prpCod and pp.prpCNPJ = ppf.prpCNPJ " +
151         " WHERE pri.prgCod = " + str(prgCod) +
152         " GROUP BY prif.codmat ")
153         #print('frequencias:')
154         #print(frequencias)
155 
156         #MATERIAL
157         materiais = db_siasgnet.query(
158         "SELECT distinct ic.numeroitem, ic.codigoitemcatalogo " +
159         " FROM Siasgnet_dc_Compartilhado_VBL.itemcompra ic " +
160         " JOIN Siasgnet_dc_Compartilhado_VBL.versaocompraitemcompra vci ON " +
161         " vci.codigoitemcompra = ic.codigoitemcompra " +
162         " JOIN Siasgnet_dc_Compartilhado_VBL.versaocompra vc " +
163         " ON vc.codigoversaocompra = vci.codigoversaocompra " +
164         " JOIN Siasgnet_dc_Compartilhado_VBL.compra c ON " +
165         " c.codigocompra = vc.codigocompra " +
166         " WHERE c.codigomodalidadecompra||c.numerouasgorigem||" +
167         "CAST(c.numerocompra AS INTEGER)||c.anocompra = " +
168         "5" + str(item['coduasg']) + str(item['numprp']) +
169         " ORDER BY ic.numeroitem ")
170         #print('materiais de:'+ '5' + str(item['coduasg']) + str(item['numprp']))
171         #print(materiais)
172 
173         licitacoes = licitacoes + 1
174 
175         if (licitacoes % 100) == 0:
176             #Atualiza o arquivo a cada 100 pregoes processados
177             filepath = Path('C:/tmp/brazilian_comprasnet.csv')
178             filepath.parent.mkdir(parents=True, exist_ok=True)
179             dfi.to_csv(filepath)
180 
181     # Lance eh de empresa punida se estiver entre CNPJs punidos
182     if str(item['Competitor']) in punidos:
183         dfi.at[index,'Collusive_competitor_original'] = 1
184         dfi.at[index,'Collusive_competitor'] = 1
185 
186     for lance in lances:
187         if lance['ipgCod'] == item['Tender']:
188             dfi.at[index,'Number_bids'] = lance['qtdeLances']
189             break
190 
191     for especificidade in especificidades:
192         if especificidade['codmat'] == item['codmat']:
193             dfi.at[index,'especificidade'] = especificidade['especificidade']
194             break
195 
196     for frequencia in frequencias:
197         if frequencia['codmat'] == item['codmat']:
198             dfi.at[index,'frequencia'] = frequencia['frequencia']
199             break
```

```
200
201     for material in materiais:
202         if material['numeroitem'] == item['ipgItem']:
203             dfi.at[index,'material'] = str(material['codigoitemcatalogo'])
204             break
205
206     dfi.at[index,'atualizado'] = 1
207
208 dfi.info()
209
210 #Calcula as screens
211 screens = ['CV', 'SPD', 'DIFFP', 'RD', 'KURT', 'SKEW', 'KSTEST']
212 dfi = calculate_screen_variables(dfi, screens)
213 for screen in screens:
214     dfi[screen].replace([np.inf, -np.inf], np.nan, inplace=True)
215     dfi[screen] = dfi[screen].fillna(0)
216
217 dfi.info()
218
219 #Atualiza arquivo ao final
220 filepath = Path(nome_arquivo)
221 filepath.parent.mkdir(parents=True, exist_ok=True)
222 dfi.to_csv(filepath)
```

II. CÓDIGO PARA CÁLCULO DAS *SCREENS*

```
import numpy as np
import pandas as pd
from pathlib import Path
from scipy.stats import uniform
from scipy.stats import kstest
from scipy.stats import lognorm

#SCREENS
def ks_test_of_function(array_data, function):
    '''Calculate the function (uniform or lognorm) of
    array_data and then calculate Kolmogorov-Smirnov test'''

    if function is 'uniform':
        loc, scale = uniform.fit(array_data)
        n = uniform(loc=loc, scale=scale)
    elif function is 'lognorm':
        s, loc, scale = lognorm.fit(array_data)
        n = lognorm(s=s, loc=loc, scale=scale)
    else:
        return None
    return kstest(array_data, n.cdf)[0]

def calculate_screen_variables(df, screens, decimals=4):
    ''' Calculate the defined screens variables of the dataframe
    (columns are Tender and Bid_value). By default, 4 decimals of accuracy '''

    if 'CV' in screens:
        # Calculate the coefficient of variation (CV)
        mean_bid_value_by_tender = df.groupby(['Tender'])['Bid_value'].mean()
        std_by_tender = df.groupby(['Tender'])['Bid_value'].std()
        cv = pd.Series(std_by_tender / mean_bid_value_by_tender,
        name='CV').round(decimals=decimals)
        res = df.merge(cv, how='inner', left_on='Tender',
        right_on='Tender', sort=False)
        res.set_index(df.index,inplace=True)
        df = res
    if 'SPD' in screens:
        # Calculate the spread (SPD)
        max_bid_value_by_tender = df.groupby(['Tender'])['Bid_value'].max()
        min_bid_value_by_tender = df.groupby(['Tender'])['Bid_value'].min()
        spd = pd.Series((max_bid_value_by_tender - min_bid_value_by_tender) /
        min_bid_value_by_tender, name='SPD').round(decimals=decimals)
        res = df.merge(spd, how='inner', left_on='Tender',
        right_on='Tender', sort=False)
        res.set_index(df.index,inplace=True)
```

```
        df = res
    if 'DIFFP' in screens:
        # Calculate the differences between the two lowest bids
        # is the percentage difference (DIFFP)
        min_bid_value_by_tender = df.groupby(['Tender'])['Bid_value'].min()
        df_without_duplicates = df.drop_duplicates(subset=['Bid_value'])
        min2_bid_value_by_tender = df_without_duplicates.groupby(
        ['Tender'])['Bid_value'].nsmallest(2).groupby(['Tender']).last()
        diffp = pd.Series((min2_bid_value_by_tender - min_bid_value_by_tender) /
        min_bid_value_by_tender, name='DIFFP').round(decimals=decimals)
        res = df.merge(diffp, how='inner', left_on='Tender',
        right_on='Tender', sort=False)
        res.set_index(df.index,inplace=True)
        df = res
    if 'RD' in screens:
        min_bid_value_by_tender = df.groupby(['Tender'])['Bid_value'].min()
        std_by_tender = df.groupby(['Tender'])['Bid_value'].std()
        df_without_duplicates = df.drop_duplicates(subset=['Bid_value'])
        min2_bid_value_by_tender = df_without_duplicates.groupby(
        ['Tender'])['Bid_value'].nsmallest(2).groupby(['Tender']).last()
        df_max_bid_by_tenders = df.groupby(
        ['Tender'])['Bid_value'].transform('max')
        df_losing_bids = df[~(df['Bid_value'] == df_max_bid_by_tenders)]
        std_losing_bids_by_tender = df_losing_bids.groupby(
        ['Tender'])['Bid_value'].std()
        df_std_losing_bids_by_tender = pd.DataFrame(
        {'Tender': std_losing_bids_by_tender.index,
        'STD': std_losing_bids_by_tender.values})
        df_std_by_tender = pd.DataFrame({'Tender': std_by_tender.index,
        'STD': std_by_tender.values})
        #df_std_losing_bids_by_tender['STD'] = df_std_losing_bids_by_tender[
        #'STD'].replace(0, np.nan)
        df_std_losing_bids_by_tender['STD'] = df_std_losing_bids_by_tender[
        'STD'].fillna(df_std_by_tender['STD'])
        std_losing_bids_by_tender =  pd.Series(
        data=df_std_losing_bids_by_tender['STD'])
        rd = pd.Series(
        (min2_bid_value_by_tender.reset_index(drop=True) -
        min_bid_value_by_tender.reset_index(drop=True)) /
        std_losing_bids_by_tender.reset_index(drop=True), name='RD')
        rd = pd.DataFrame({'Tender': min_bid_value_by_tender.index,
        'RD': rd.values}).round(decimals=decimals)
        res = df.merge(rd, how='inner', left_on='Tender',
        right_on='Tender', sort=False)
        res.set_index(df.index,inplace=True)
        df = res
    if 'KURT' in screens:
        # Calculate Kurtosis statistic (KURTO)
        kurtosis_by_tender = df.groupby(['Tender'])['Bid_value'].apply(
        pd.DataFrame.kurt)
        kurtosis_by_tender = pd.Series(kurtosis_by_tender, name='KURT')
        kurtosis_by_tender = kurtosis_by_tender.fillna(0).round(decimals=decimals)
```

```
98          res = df.merge(kurtosis_by_tender, how='inner', left_on='Tender',
99          right_on='Tender', sort=False)
100         res.set_index(df.index,inplace=True)
101         df = res
102     if 'SKEW' in screens:
103         # Calculate Kewness statistic (SKEW)
104         skew_by_tender = df.groupby(['Tender'])['Bid_value'].skew()
105         skew_by_tender = pd.Series(skew_by_tender, name='SKEW')
106         skew_by_tender = skew_by_tender.fillna(0).round(decimals=decimals)
107         res = df.merge(skew_by_tender, how='inner', left_on='Tender',
108         right_on='Tender', sort=False)
109         res.set_index(df.index,inplace=True)
110         df = res
111     if 'KSTEST' in screens:
112         # Calculate Kolmogorov-Smirnov statistic (KSTEST) for verifying if the
113         # bids in a tender follow a distribution (uniform, lognorm, etc...)
114         kolmogorov_smirnov_by_tender = df.groupby(
115         ['Tender'])['Bid_value'].apply(lambda x: ks_test_of_function(x, 'uniform'))
116         kolmogorov_smirnov_by_tender = pd.Series(
117         kolmogorov_smirnov_by_tender, name='KSTEST').round(decimals=decimals)
118         res = df.merge(kolmogorov_smirnov_by_tender, how='inner',
119         left_on='Tender', right_on='Tender', sort=False)
120         res.set_index(df.index,inplace=True)
121         df = res
122     if 'KSTEST_L' in screens:
123         # Calculate Kolmogorov-Smirnov statistic (KSTEST) for verifying if
124         # the bids in a tender follow a distribution (uniform, lognorm, etc...)
125         kolmogorov_smirnov_by_tender = df.groupby(
126         ['Tender'])['Bid_value'].apply(lambda x: ks_test_of_function(x, 'lognorm'))
127         kolmogorov_smirnov_by_tender = pd.Series(
128         kolmogorov_smirnov_by_tender, name='KSTEST_L').round(decimals=decimals)
129         res = df.merge(kolmogorov_smirnov_by_tender, how='inner',
130         left_on='Tender', right_on='Tender', sort=False)
131         res.set_index(df.index,inplace=True)
132         df = res
133     return df
```

III. CÓDIGO UTILITÁRIO PARA ACESSO AO BANCO DE DADOS DO COMPRASNET

```
1  import os
2  import jaydebeapi
3  import jpype
4  import dateutil.parser
5
6  import pandas as pd
7  from pandas.io.json import json_normalize
8
9  #diretorio com jar de conexao jdbc com o Serpro
10 BASE_DIR = 'C:/Tools/serpro/'
11
12 class DassDB:
13
14     conn = None
15     url = None
16     TEIID_JAR = 'jboss-dv-6.3.0-teiid-jdbc.jar'
17
18     def __init__(self, url):
19         self.url = url
20         self.connect()
21
22     def __del__(self):
23         try:
24             if self.conn:
25                 self.conn.close()
26         except Exception as e:
27             logger.error('Erro fechar a conexao DAAS')
28             logger.exception(e)
29             pass
30
31     def connect(self):
32         jar_path = os.path.join(BASE_DIR, DassDB.TEIID_JAR)
33         if not jpype.isJVMStarted():
34             args = []
35             class_path = [jar_path]
36             class_path.extend(jaydebeapi._get_classpath())
37             args.append('-Djava.class.path=%s' % os.path.pathsep.join(class_path))
38             args.append('-Djavax.net.ssl.trustStore=%s' % os.path.join(
39             BASE_DIR, 'daas.serpro.gov.br.jks'))
40             jvm_path = jpype.getDefaultJVMPath()
41             jpype.startJVM(jvm_path, *args)
42
43         self.conn = jaydebeapi.connect("org.teiid.jdbc.TeiidDriver",
```

```
44                                      self.url,
45                                      [DAAS_DB_USER, DAAS_DB_PASS],
46                                      jar_path)
47
48      def query(self, sql):
49          try:
50              cursor = self.conn.cursor()
51              cursor.execute(sql)
52          except Exception as e:
53              logger.exception(e)
54              logger.error("Erro ao executar no DAAS: {}.".format(sql))
55              self.connect()
56              cursor = self.conn.cursor()
57              cursor.execute(sql)
58          desc = cursor.description
59          registros = [dict(zip([str(col[0]) for col in desc], row))
60                          for row in cursor.fetchall()]
61          cursor.close()
62          return registros
```

IV. CÓDIGO PARA DETECÇÃO DE CONLUIO

```
import os
import sys
import numpy as np
import pandas as pd
import datetime
import matplotlib.pyplot as plt
import warnings
from joblib import dump, load
from matplotlib.lines import Line2D
from sklearn.ensemble import
    ExtraTreesClassifier, RandomForestClassifier,
    AdaBoostClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GroupShuffleSplit
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    balanced_accuracy_score, f1_score, confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.gaussian_process import GaussianProcessClassifier
from collections import defaultdict
from scipy import stats
from itertools import cycle


# Font size to plot
default_font_size = 18
plt.rcParams.update({'font.size': default_font_size})

# Format to print
pd.options.display.float_format = '{:,.4f}'.format

# To hide warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)

# Paths and filenames of the datasets
path = os.path.abspath('') #os.path.dirname(os.path.abspath(__file__))
db_collusion_brazilian_comprasnet = os.path.join(
    path, 'DB_Collusion_Brazil_Comprasnet_processed.csv')
db_collusion_brazilian = os.path.join(
    path, 'DB_Collusion_Brazil_processed.csv')
db_collusion_italian = os.path.join(
    path, 'DB_Collusion_Italy_processed.csv')
```

```
db_collusion_american = os.path.join(
    path, 'DB_Collusion_America_processed.csv')
db_collusion_switzerland_gr_sg = os.path.join(
    path, 'DB_Collusion_Switzerland_GR_and_See-Gaster_processed.csv')
db_collusion_switzerland_ticino = os.path.join(
    path, 'DB_Collusion_Switzerland_Ticino_processed.csv')
db_collusion_japan = os.path.join(
    path, 'DB_Collusion_Japan_processed.csv')
db_collusion_all = os.path.join(path, 'DB_Collusion_All_processed.csv')

# To save plots (pdf format)
plot_pdf = True

# User's parameters for the functions
ml_algorithms = ['GaussianProcessClassifier', 'SGDClassifier',
                 'ExtraTreesClassifier', 'RandomForestClassifier',
                 'AdaBoostClassifier', 'GradientBoostingClassifier',
                 'SVC', 'KNeighborsClassifier', 'MLPClassifier',
                 'BernoulliNB', 'GaussianNB']

# Screening variables to use. There are seven:
# CV, SPD, DIFFP, RD, KURT, SKEW and KSTEST
screens = ['CV', 'SPD', 'DIFFP', 'RD', 'KURT', 'SKEW', 'KSTEST']
# Test and train sizes. The test_size is 1-train_size
train_size = 0.8
# Number of repetitions for each ML algorithm.
# Minimum value > 30. Recommended value > 100
repetitions = 50
# Number of estimators for ML algorithms
n_estimators = 300
# To plot precision-recall curves
precision_recall = True
# To load the error metrics (to load previous data experimentation)
load_data = False
# To save the error metrics (to persist the data experimentation)
save_data = True


def shuffle_tenders(df1):
    ''' Shuffle tenders. The reason is that maybe the colluded tenders are
    concentrated in some parts of the excel (dataframe)'''

    df = df1.copy()
    df = df.sample(frac=1).reset_index(drop=True)
    df['Tender'] = df['Tender'].astype(str)
    reindex_tenders = 1
    list_tenders = []
    for index, row in df.iterrows():
        if not row['Tender'] in list_tenders:
            df['Tender'].replace(row['Tender'], reindex_tenders, inplace=True)
            reindex_tenders = reindex_tenders + 1
            list_tenders.append(row['Tender'])
```

```
98      return df
99
100
101 def calculate_colluded_tenders_by_bidder(df):
102     ''' Calculate the colluded tenders by bidder and print the results '''
103
104     df_aux = df.copy()
105     df_tenders_by_bidder = df_aux.groupby(
106     ['Competitors']).size().reset_index(name='Total_tenders')
107     df_aux['Collusive_competitor'] = df_aux['Collusive_competitor'].apply(
108     lambda x: 1 if x > 0 else x)
109     df_tenders_by_bidder['Total_colluded_tenders'] = df_aux.groupby(
110     ['Competitors'])['Collusive_competitor'].sum()
111     df_tenders_by_bidder['Ratio'] = df_tenders_by_bidder[
112     'Total_colluded_tenders'] / df_tenders_by_bidder['Total_tenders'] * 100
113     with pd.option_context('display.max_rows', None, 'display.max_columns', None):
114         print(df_tenders_by_bidder)
115
116
117 def printScatterMatrix(df, color_labels, colors, labels_legend, dataset):
118     ''' Scatter matrix for a dataframe '''
119
120     plt.rcParams.update({'font.size': 11}) # Font size to plot
121     sm = pd.plotting.scatter_matrix(df, figsize=(12, 16), diagonal='kde',
122     alpha=0.35, color=color_labels, s=3, rasterized=True) # kde or hist
123     n = len(df.columns)
124     for x in range(n):
125         for y in range(n):
126             # to get the axis of subplots
127             ax = sm[x, y]
128             # to make x axis name vertical
129             #ax.xaxis.label.set_rotation(330)
130             # to make y axis name horizontal
131             #ax.yaxis.label.set_rotation(0)
132             # to make sure y axis names are outside the plot area
133             #ax.yaxis.labelpad = 40
134             ax.xaxis.labelpad = 20
135             # to show the half of the scatter matrix
136             if x < y:
137                 sm[x, y].set_visible(False)
138             # adjust xlim, ylim
139             if x == 1:
140                 ax.set_ylim([0, 0.4])
141             elif x == 2:
142                 ax.set_ylim([0, 0.5])
143             elif x == 3:
144                 ax.set_ylim([0, 0.4])
145             elif x == 4:
146                 ax.set_ylim([-750, 750])
147             elif x == 5:
148                 ax.set_ylim([-5, 10])
149             elif x == 6:
```

```
                ax.set_ylim([-3, 3])
            if y == 0:
                ax.set_xlim([0, 0.2 * 1000000000])
            elif y == 1:
                ax.set_xlim([0, 0.4])
            elif y == 2:
                ax.set_xlim([0, 0.5])
            elif y == 3:
                ax.set_xlim([0, 0.4])
            elif y == 4:
                ax.set_xlim([-750, 750])
            elif y == 5:
                ax.set_xlim([-5, 10])
            elif y == 6:
                ax.set_xlim([-3, 3])
    # More bottom margin to read x and y axis
    plt.subplots_adjust(hspace=0.1, wspace=0.1)
    # Legend
    handles = [plt.plot([], [], color=colors[i], ls='', marker='.',
    markersize=np.sqrt(90))[0] for i in range(len(colors))]
    plt.legend(handles, labels_legend, loc=(-1,4))
    # Draw
    plt.draw()
    if plot_pdf:
        name_file = dataset + '_Scatter_matrix.pdf'
        plt.savefig(name_file, format='pdf', dpi=1200, bbox_inches='tight')
        print('Generated and saved file called ' + name_file)
    plt.rcParams.update({'font.size': default_font_size}) # Font size to plot


def print_boxplot(df, dataset, column_names, groupby, min_ylim, max_ylim,
step_y, xlabel, percentage=True):
    plt.rcParams.update({'font.size': 10})
    fig = plt.figure(figsize=(2*0.7,6*0.7))
    ax = fig.gca()
    df.boxplot(column=column_names, by=groupby, ax=ax, fontsize=None, rot=0,
               grid=False, notch=True, widths=0.3, positions = (0.3, 0.9),
               layout=None, return_type=None, showfliers=False, meanline=True,
               showmeans=True, patch_artist=True, vert=True,
               medianprops=dict(linestyle='-', linewidth=3, color='limegreen'),
               meanprops=dict(linestyle='-', linewidth=3, color='firebrick'))
    # Configurate plotting
    ax.set_title(column_names.replace('_', ' '), fontweight='bold')
    fig.suptitle('')
    plt.xlabel(xlabel)
    ax.set_ylim([min_ylim, max_ylim])
    ax.set_yticks(np.arange(min_ylim, max_ylim+step_y, step_y))
    ax.yaxis.grid(True, linestyle='--', alpha=0.5)
    if percentage:
        # Percentage format
        ax.set_yticklabels(['{:.2f}%'.format(x) for x in plt.gca().get_yticks()])
    custom_lines = [Line2D([0], [0], color='firebrick', lw=3),
```

```
                    Line2D([0], [0], color='limegreen', lw=3)]
    names_lines = ['Mean', 'Median']
    ax.legend(custom_lines, names_lines, loc='center', bbox_to_anchor=(1.75,0.125))
    plt.draw()
    if plot_pdf:
        name_file = dataset + '_BoxPlot_' + column_names + '.pdf'
        fig.savefig(name_file, format='pdf', dpi=1200, bbox_inches='tight')
        print('Generated and saved file called ' + name_file)
    plt.rcParams.update({'font.size': default_font_size})


def predict_collusion_company(df, dataset, predictors_column_name,
targets_column_name, algorithm, train_size, n_estimators=None):
    ''' Predict collusion applying the ML algorithm '''

    # Datasets to have to simplify the process' time
    simplify_process = ['japan', 'italian', 'switzerland_gr_sg', 'american', 'all']

    # To assing the dataframes
    predictors = df[predictors_column_name]
    targets = df[targets_column_name]

    # We create the training and test sample, both for predictors and for the
    # objective variable, based on the tender group.
    # That is, the bids of a tender either all own to the train
    # group or the test group. They cannot be divided between both groups.
    gss = GroupShuffleSplit(n_splits=5, train_size=train_size)
    train_index, test_index = next(gss.split(
        predictors, targets, groups=df['Tender']))
    x_train = predictors.loc[train_index]
    y_train = targets.loc[train_index]
    x_test = predictors.loc[test_index]
    y_test = targets.loc[test_index]

    # Train the model with the selected algorithm
    if algorithm == 'ExtraTreesClassifier':
        classifier = ExtraTreesClassifier(n_estimators=n_estimators,
                            criterion='gini', max_depth=None,
                            min_samples_split=2, min_samples_leaf=1,
                            min_weight_fraction_leaf=0.0, max_features='auto',
                            max_leaf_nodes=None, min_impurity_decrease=0.0,
                            bootstrap=True, oob_score=True, n_jobs=-1,
                            random_state=None, verbose=0, warm_start=False,
                            class_weight='balanced', ccp_alpha=0.0,
                            max_samples=None)
    elif algorithm == 'RandomForestClassifier':
        classifier = RandomForestClassifier(n_estimators=n_estimators,
                            criterion='gini',  max_depth=None,
                            min_samples_split=2, min_samples_leaf=1,
                            min_weight_fraction_leaf=0., max_features=None,
                            max_leaf_nodes=None, min_impurity_decrease=0.,
                            bootstrap=True, oob_score=True, n_jobs=-1,
```

```
                            random_state=None, verbose=0,
                            warm_start=False, class_weight='balanced')
    elif algorithm == 'SGDClassifier':
        classifier = SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001,
                            l1_ratio=0.15, fit_intercept=True, max_iter=10000,
                            tol=0.001, shuffle=True, verbose=0, epsilon=0.1,
                            n_jobs=-1, random_state=None, learning_rate='optimal',
                            eta0=0.0, power_t=0.5, early_stopping=False,
                            validation_fraction=0.1, n_iter_no_change=5,
                            class_weight=None, warm_start=False, average=False)
    elif algorithm == 'AdaBoostClassifier':
        classifier = AdaBoostClassifier(base_estimator=None,
            n_estimators=n_estimators, learning_rate=1.0,
            algorithm='SAMME.R', random_state=None)
    elif algorithm == 'GradientBoostingClassifier':
        if dataset in simplify_process:
            learning_rate = 100
            tol = 10
            estimators = int(round(n_estimators / 3))
        else:
            learning_rate = 0.1
            tol = 0.0001
            estimators = n_estimators
        #creiterion='mae' alterado para squared_error
        classifier = GradientBoostingClassifier(loss='deviance',
                learning_rate=learning_rate, n_estimators=estimators,
                subsample=1.0, criterion='squared_error',
                min_samples_split=2, min_samples_leaf=1,
                min_weight_fraction_leaf=0.0, max_depth=None,
                min_impurity_decrease=0.0, init=None,
                random_state=None, max_features=None, verbose=0,
                max_leaf_nodes=None, warm_start=False,
                validation_fraction=0.1, n_iter_no_change=None,
                tol=tol, ccp_alpha=0.0)
    elif algorithm == 'SVC':
        classifier = SVC(C=1.0, kernel='rbf', degree=3, gamma='scale',
                coef0=0.0, shrinking=True, probability=False, tol=0.001,
                cache_size=200, class_weight='balanced', verbose=False,
                max_iter=-1, decision_function_shape='ovr',
                break_ties=False, random_state=None)
    elif algorithm == 'KNeighborsClassifier':
        classifier = KNeighborsClassifier(n_neighbors=10, weights='uniform',
                algorithm='auto', leaf_size=30, p=2, metric='minkowski',
                metric_params=None, n_jobs=-1)
    elif algorithm == 'MLPClassifier':
        classifier = MLPClassifier(hidden_layer_sizes=(240, 120, 70, 35),
                activation='logistic', solver='adam', alpha=0.0001,
                batch_size='auto', learning_rate='constant',
                learning_rate_init=0.001, power_t=0.5, max_iter=200,
                shuffle=True, random_state=None, tol=0.0001, verbose=0,
                warm_start=False, momentum=0.9, nesterovs_momentum=True,
                early_stopping=False, validation_fraction=0.1, beta_1=0.9,
```

```
306                 beta_2=0.999, epsilon=1e-08, n_iter_no_change=10,
307                 max_fun=15000)
308     elif algorithm == 'GaussianNB':
309         classifier = GaussianNB(priors=None, var_smoothing=1e-09)
310     elif algorithm == 'BernoulliNB':
311         classifier = BernoulliNB(alpha=0.5, binarize=0,
312                 fit_prior=True, class_prior=None)
313     elif algorithm == 'GaussianProcessClassifier':
314         if dataset in simplify_process:
315             max_iter_predict = 5
316             n_restarts_optimizer = 2
317         else:
318             max_iter_predict = 5000
319             n_restarts_optimizer = 50
320         classifier = GaussianProcessClassifier(kernel=None,
321                 optimizer='fmin_l_bfgs_b',
322                 n_restarts_optimizer=n_restarts_optimizer,
323                 max_iter_predict=max_iter_predict, warm_start=False,
324                 copy_X_train=True, random_state=None,
325                 multi_class='one_vs_rest', n_jobs=-1)
326
327     # We build the model for the train group
328     classifier = classifier.fit(x_train, y_train.values.ravel())
329
330     # We predict for the values of the test group
331     predictions = classifier.predict(x_test)
332     df_predictions = pd.DataFrame(data=predictions, index=y_test.index,
333         columns=['Forecast_collusive_competitor'])
334
335     # To calculate the error metrics for the classification binary model
336     accuracy = accuracy_score(y_test, predictions) * 100
337     balanced_accuracy = balanced_accuracy_score(y_test, predictions) * 100
338     precision = precision_score(y_test, predictions, pos_label=1, average='binary',
339         zero_division=1) * 100 # Ratio of true positives: tp / (tp+fp)
340     recall = recall_score(y_test, predictions, pos_label=1, average='binary',
341         zero_division=1) * 100 # Ratio of true positives: tp / (tp+fn)
342     f1 = f1_score(y_test, predictions, pos_label=1, average='binary',
343         zero_division=1) * 100 # F1 = 2 * (precision * recall) / (precision+recall)
344     confusion = confusion_matrix(y_test, predictions, normalize='all') * 100
345
346     return accuracy, balanced_accuracy, precision, recall, f1, confusion,
347         y_test, df_predictions
348
349
350 def algorithm_comparison(df, dataset, predictors, targets, algorithms,
351     train_size, repetitions, n_estimators, precision_recall=False,
352     load_data=False, save_data=False):
353     ''' Print table to compare Machine Learning algorithms '''
354
355     df = shuffle_tenders(df)
356
357     for setting in predictors:
```

```
358         print('')
359         print('Generating models for ' + setting)
360         accuracy = defaultdict(list)
361         balanced_accuracy = defaultdict(list)
362         false_positive = defaultdict(list)
363         false_negative = defaultdict(list)
364         precision = defaultdict(list)
365         recall = defaultdict(list)
366         f1 = defaultdict(list)
367         tenders_test = defaultdict(list)
368         tenders_predictions = defaultdict(list)
369
370         # Create namefile
371         namefile = dataset + '_ML_algorithms_experimentation_' +
372             setting + '_' + str(repetitions) + 'repetitions'
373
374         if load_data == False:
375             for algorithm in algorithms:
376                 print('Training algorithm ' + algorithm)
377                 df_copy = df.copy()
378                 if algorithm in ['GaussianProcessClassifier',
379                     'GradientBoostingClassifier', 'SVC']:
380                     loop = int(round(repetitions / 40))
381                     if dataset == 'all'
382                         and algorithm == 'GaussianProcessClassifier':
383                         # Exception: reduce the datataset to be able to compute
384                         # this dataset and algorithm
385                         df_copy = df_copy.sample(frac=0.5).reset_index(drop=True)
386                 else:
387                     loop = repetitions
388                 for i in range(loop):
389                     item_accuracy, item_balanced_accuracy, item_precision,
390                         item_recall, item_f1, confusion_matrix, item_tenders_test,
391                         item_tenders_predictions = \
392                                 predict_collusion_company(
393                                 df_copy, dataset, predictors[setting], targets,
394                                 algorithm, train_size, n_estimators)
395                     accuracy[algorithm].append(item_accuracy)
396                     balanced_accuracy[algorithm].append(item_balanced_accuracy)
397                     if confusion_matrix.shape[1] == 2:
398                         false_positive[algorithm].append(confusion_matrix[0][1])
399                         false_negative[algorithm].append(confusion_matrix[1][0])
400                     else:
401                         false_positive[algorithm].append(0)
402                         false_negative[algorithm].append(0)
403                     precision[algorithm].append(item_precision)
404                     recall[algorithm].append(item_recall)
405                     f1[algorithm].append(item_f1)
406                     tenders_test[algorithm].append(item_tenders_test)
407                     tenders_predictions[algorithm].append(item_tenders_predictions)
408
409             # Save dictionaries to persist the data experimentation
```

```
            if save_data:
                path_namefile = os.path.join(path, namefile + '.pkl')
                file = [accuracy, balanced_accuracy, false_positive, false_negative,
                    precision, recall, f1, df, tenders_test, tenders_predictions]
                dump(file, path_namefile, compress=6)

        else:
            # To load data
            pkl_file = os.path.join(path, namefile + '.pkl')
            [accuracy, balanced_accuracy, false_positive, false_negative,
                precision, recall, f1, df, tenders_test,
                tenders_predictions] = load(pkl_file)

        for algorithm in algorithms:
            # Print error metrics
            test_size = 1 - train_size
            print('Algorithm {} with train:test {:,.2f}:{:,.2f}, {} repetitions
                and {}: mean_accuracy={:,.1f}, mean_FP={:,.1f}, '
                'mean_FN={:,.1f}, mean_balanced_accuracy={:,.1f},
                mean_f1={:,.1f}, median_f1={:,.1f}, mean_precision={:,.1f}, '
                'median_precision={:,.1f}, mean_recall={:,.1f} and
                median_recall={:,.1f}'.format(
                algorithm, train_size, test_size, repetitions, setting,
                np.mean(accuracy[algorithm]), np.mean(false_positive[algorithm]),
                np.mean(false_negative[algorithm]),
                np.mean(balanced_accuracy[algorithm]), np.mean(f1[algorithm]),
                np.median(f1[algorithm]), np.mean(precision[algorithm]),
                np.median(precision[algorithm]), np.mean(recall[algorithm]),
                np.median(recall[algorithm])))

        # Print curve precision vs recall with iso-F1 lines
        if precision_recall:
            #plot_precision_vs_recall(dataset, algorithms, precision, recall,
            #    min_f1=0.4, max_f1=0.86, f1_curves=24, min_x_y_lim=0.5,
            #    max_x_y_lim=1, namefile=namefile)
            plot_precision_vs_recall(dataset, algorithms, precision, recall,
                min_f1=0.05, max_f1=0.86, f1_curves=24,
                min_x_y_lim=0.05, max_x_y_lim=1, namefile=namefile)


def plot_precision_vs_recall(dataset, algorithms, precision, recall,
min_f1, max_f1, f1_curves, min_x_y_lim, max_x_y_lim, namefile=None):
    ''' Plot the precision vs recall with F1 Score iso-curves to compare the ML
        algorithms. The point to cut both lines (precision and recall) is median
        of the F1 score. This is necessary to correspond the point with the F1
        Score iso-curves'''

    plt.rcParams.update({'font.size': 26}) # Font size to plot

    # Colors and markers for the plot to compare 11 algorithms
    colors = cycle(['tab:blue', 'tab:orange', 'tab:green', 'tab:red',
                    'tab:purple', 'tab:brown',
```

```
                    'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan', 'gold'])
    markers = cycle(['o', '.', 'v', '^', '<', '>', 's', 'p', 'D', 'P', 'X'])

    fig = plt.figure(figsize=(12, 12))
    f1_scores = np.linspace(min_f1, max_f1, num=f1_curves)
    lines = []
    labels = []

    # Create iso-F1 curves
    for f1_scores in f1_scores:
        x = np.linspace(0.01, 1)
        y = f1_scores * x / (2 * x - f1_scores)
        l, = plt.plot(x[y >= 0], y[y >= 0], color='gray', alpha=0.65,
            linestyle='-.', lw=2)
        plt.annotate('F1={0:0.2f}'.format(f1_scores),
            xy=(0.94, y[46] - 0.001), fontsize=17)
    lines.append(l)
    labels.append('F1 curves')

    # Convert to [0, 1]
    for item in algorithms:
        recall[item] =  [x / 100 for x in recall[item]]
        precision[item] =  [x / 100 for x in precision[item]]

    # Calculate the points to plot the two lines
    line_precision_x = defaultdict(list)
    line_precision_y = defaultdict(list)
    line_recall_x = defaultdict(list)
    line_recall_y = defaultdict(list)
    for item in algorithms:
        line_recall_x[item] = [np.percentile(recall[item], 25),
            np.percentile(recall[item], 75)]
        line_recall_y[item] = [np.median(precision[item]),
            np.median(precision[item])]
        line_precision_x[item] = [np.median(recall[item]),
            np.median(recall[item])]
        line_precision_y[item] = [np.percentile(precision[item], 25),
            np.percentile(precision[item], 75)]

    # Plot the two lines and the point to cut both lines
    for item, color in zip(algorithms, colors):
        # It can possible to use markers list
        l, = plt.plot(line_precision_x[item], line_precision_y[item],
            color=color, lw=4, marker='_',
            markersize=14, markeredgewidth=4)
        l, = plt.plot(line_recall_x[item], line_recall_y[item],
            color=color, lw=4, marker='|',
            markersize=14, markeredgewidth=4)
        l, = plt.plot(np.median(recall[item]), np.median(precision[item]),
            color=color, markersize=8, marker='o')
        lines.append(l)
        labels.append('{}'.format(item))
```

```
514
515     plt.xlim([min_x_y_lim, max_x_y_lim])
516     plt.ylim([min_x_y_lim, max_x_y_lim])
517     plt.grid(dashes=(5, 10), linewidth=1)
518     plt.xlabel('Recall')
519     plt.ylabel('Precision')
520     plt.legend(lines, labels, loc='lower center', bbox_to_anchor=(0.5, -0.3),
521         prop=dict(size=default_font_size - 2), ncol=3)
522
523     # Axis in percentage format
524     ax = fig.gca()
525     ax.set_xticklabels(['{:.0f}%'.format(x*100) for x in plt.gca().get_xticks()])
526     ax.set_yticklabels(['{:.0f}%'.format(x*100) for x in plt.gca().get_yticks()])
527
528     plt.draw()
529     if plot_pdf:
530         name_file = dataset + '_Precision_Recall_' + namefile + '.pdf'
531         fig.savefig(name_file, format='pdf', dpi=1200, bbox_inches='tight')
532         print('Generated and saved file called ' + name_file)
533
534
535 def plotTwoHistograms(data_1, data_2, label_1, label_2, max_range, bins,
536 max_xlim, density=True):
537     ''' Plot two histograms or density functions '''
538
539     # Fit lognormal distribution
540     data_1 = sorted(data_1.values)
541     data_2 = sorted(data_2.values)
542     shape, loc, scale = stats.lognorm.fit(data_1, loc=0)
543     data_1_prob_density_function_lognorm = stats.lognorm.pdf(
544         data_1, shape, loc, scale)
545     shape, loc, scale = stats.lognorm.fit(data_2, loc=0)
546     data_2_prob_density_function_lognorm = stats.lognorm.pdf(
547         data_2, shape, loc, scale)
548
549     # Plot histograms and density distributions
550     fig = plt.figure(figsize=(16, 12))
551     plt.hist(data_1, bins=bins, range=(0, max_range), alpha=0.3,
552         label='Histogram: ' + label_1, facecolor='g', density=density)
553     plt.hist(data_2, bins=bins, range=(0, max_range), alpha=0.3,
554         label='Histogram: ' + label_2, facecolor='r', density=density)
555     plt.plot(data_1, data_1_prob_density_function_lognorm,
556         label='Probability density function (log normal): ' + label_1,
557         color='g', linewidth=3)
558     plt.plot(data_2, data_2_prob_density_function_lognorm,
559         label='Probability density function (log normal): ' + label_2,
560         color='r', linewidth=3)
561     plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
562     plt.ylabel('Probability')
563     plt.xlabel('Number of bids by tender')
564     plt.xlim(0, max_xlim)
565     plt.xticks(np.arange(0, max_xlim, step=max_xlim/bins))
```

```
    plt.grid(True, linestyle='--', alpha=0.5)

    # Axis in percentage format
    ax = fig.gca()
    ax.set_yticklabels(['{:.2f}%'.format(x*100) for x in plt.gca().get_yticks()])

    if plot_pdf:
        name_file = dataset + '_Two_density_plots.pdf'
        plt.savefig(name_file, format='pdf', dpi=1200, bbox_inches='tight')
        print('Generated and saved file called ' + name_file)


def print_description_processed_dataset(df):
    ''' Print the most important information for the collusive dataset '''

    # General
    df_tenders = df.drop_duplicates(subset=['Tender', 'Number_bids'])
    number_tenders = len(df['Tender'].unique())
    print('')
    print('-------------------------------------')
    print('Information of the collusive dataset')
    print('Tenders: {0}'.format(number_tenders))
    if 'Date' in df:
        df_aux = df[df['Date'] > 0] # To avoid unavailable timestamps
        minimum_date = datetime.datetime.fromtimestamp(df_aux['Date'].min())
        maximum_date = datetime.datetime.fromtimestamp(df_aux['Date'].max())
        print('Temporal range: {0}, {1}'.format(minimum_date, maximum_date))
    number_bids = len(df)
    print('Bids: {0}'.format(number_bids))
    if 'Collusive_competitor_original' in df:
        column_name = 'Collusive_competitor_original'
    else:
        column_name = 'Collusive_competitor'
    number_collusive_bidders = len(df[df[column_name] == 1])
    mean_number_bids = np.mean(df_tenders['Number_bids'])
    print('Mean value of bidders per tender: {:,.2f}'.format(mean_number_bids))
    median_number_bids = np.median(df_tenders['Number_bids'])
    print('Median value of bidders per tender: {:,.2f}'.format(median_number_bids))
    if 'Competitors' in df:
        df_competitors = df.drop_duplicates(subset=['Competitors'])
        number_competitors = len(df_competitors)
        print('Competitors: {0}'.format(number_competitors))
        number_winners = len(df_tenders['Competitors'].unique())
        print('Winning competitors of tenders: {} ({:,.2f}%)'.format(
            number_winners, number_winners/number_competitors*100))

    # Collusive vs competitive bids and tenders
    number_collusive_tenders = len(
        df_tenders[df_tenders['Collusive_competitor'] == 1])
    print('Collusive tenders: {} ({:,.2f}%)'.format(
        number_collusive_tenders, number_collusive_tenders/number_tenders*100))
    number_competitive_tenders = len(
```

```
        df_tenders[df_tenders['Collusive_competitor'] == 0])
    print('Competitive tenders: {} ({:,.2f}%)'.format(
        number_competitive_tenders, number_competitive_tenders/number_tenders*100))
    print('Collusive bids: {} ({:,.2f}%)'.format(number_collusive_bidders,
        number_collusive_bidders/number_bids*100))
    number_competitive_bidders = len(df[df[column_name] == 0])
    print('Competitive bids: {} ({:,.2f}%)'.format(
        number_competitive_bidders, number_competitive_bidders/number_bids*100))
    if 'Competitors' in df:
        number_collusive_competitors = len(
            df_competitors[df_competitors['Collusive_competitor'] == 1])
        print('Collusive competitors: {} ({:,.2f}%)'.format(
            number_collusive_competitors,
            number_collusive_competitors/number_competitors*100))
        number_competitive_competitors = len(
            df_competitors[df_competitors['Collusive_competitor'] == 0])
        print('Competitive competitors: {} ({:,.2f}%)'.format(
            number_competitive_competitors,
            number_competitive_competitors/number_competitors*100))

    # Number of tenders by received offers: 1-4, 5-10, >10
    tenders_by_offer_group_1 = len(df_tenders[df_tenders['Number_bids'] <= 4])
    print('Bids by tender: 1<=N<=4: {} ({:,.2f}%)'.format(
        tenders_by_offer_group_1, tenders_by_offer_group_1/number_tenders*100))
    tenders_by_offers_group_2 = len(
        df_tenders[df_tenders['Number_bids'] <= 10]) - tenders_by_offer_group_1
    print('Bids by tender: 5<=N<=10: {} ({:,.2f}%)'.format(
        tenders_by_offers_group_2, tenders_by_offers_group_2/number_tenders*100))
    tenders_by_offers_group_3 = len(
        df_tenders[df_tenders['Number_bids'] > 10])
    print('Bids by tender: 11<=N: {} ({:,.2f}%)'.format(
        tenders_by_offers_group_3, tenders_by_offers_group_3/number_tenders*100))

    # Values of the winner's bid
    df_winners = df[df['Winner'] == 1]
    aggregated_bid_value = df_winners['Bid_value'].sum()
    print('Aggregated tender price: {:,.0f}'.format(aggregated_bid_value))
    aggregated_collusive_bid_value = df_winners[
        df_winners[column_name] == 1]['Bid_value'].sum()
    print('Aggregated collusive tender price: {:,.0f} ({:,.2f}%)'.format(
        aggregated_collusive_bid_value,
        aggregated_collusive_bid_value/aggregated_bid_value*100))
    aggregated_competitive_bid_value = df_winners[
        df_winners[column_name] == 0]['Bid_value'].sum()
    print('Aggregated competitive tender price: {:,.0f} ({:,.2f}%)'.format(
        aggregated_competitive_bid_value,
        aggregated_competitive_bid_value/aggregated_bid_value*100))
    mean_bid_value = np.mean(df_winners['Bid_value'])
    print('Mean tender price: {:,.2f}'.format(mean_bid_value))
    median_bid_value = np.median(df_winners['Bid_value'])
    print('Median tender price: {:,.2f}'.format(median_bid_value))
    print('-------------------------------------')
```

```
    print('')


def get_dataset(dataset):
    ''' Get the collusive dataset and their fields to use in the ML algorimths '''

    predictors = defaultdict(list)

    if dataset == 'brazilian_comprasnet':
        df_collusion = pd.read_csv(db_collusion_brazilian_comprasnet, header=0)
        df_collusion['Collusive_competitor'] = df_collusion[
            'Collusive_competitor_original']
        df_collusion.drop('material', inplace=True, axis=1)
        predictors['all_setting'] = ['Tender', 'Bid_value',
            'Winner', 'Date', 'especificidade',
            'frequencia', 'ipgQuantidade', 'Number_bids']
        predictors['all_setting+screens'] = predictors['all_setting'] + screens
        predictors['common'] = [
            'Tender', 'Bid_value', 'Winner', 'Date', 'Number_bids']
    elif dataset == 'brazilian':
        df_collusion = pd.read_csv(db_collusion_brazilian, header=0)
        predictors['all_setting'] = [
            'Tender', 'Bid_value', 'Pre-Tender Estimate (PTE)',
            'Difference Bid/PTE', 'Site', 'Date', 'Brazilian State',
            'Winner', 'Number_bids']
        predictors['all_setting+screens'] = predictors['all_setting'] + screens
        predictors['common'] = [
            'Tender', 'Bid_value', 'Winner', 'Date', 'Number_bids']

    elif dataset == 'switzerland_gr_sg':
        df_collusion = pd.read_csv(db_collusion_switzerland_gr_sg, header=0)
        predictors['all_setting'] = ['Tender', 'Bid_value',
            'Contract_type', 'Date', 'Winner', 'Number_bids']
        predictors['all_setting+screens'] = predictors['all_setting'] + screens
        predictors['common'] = [
            'Tender', 'Bid_value', 'Winner', 'Date', 'Number_bids']

    elif dataset == 'switzerland_ticino':
        df_collusion = pd.read_csv(db_collusion_switzerland_ticino, header=0)
        predictors['all_setting'] = ['Tender', 'Bid_value',
            'Consortium', 'Winner', 'Number_bids']
        predictors['all_setting+screens'] = predictors['all_setting'] + screens
        predictors['common'] = ['Tender', 'Bid_value', 'Winner', 'Number_bids']

    elif dataset == 'italian':
        df_collusion = pd.read_csv(db_collusion_italian, header=0)
        predictors['all_setting'] = ['Tender', 'Bid_value',
            'Pre-Tender Estimate (PTE)', 'Difference Bid/PTE', 'Site',
            'Capital', 'Legal_entity_type', 'Winner', 'Number_bids']
        predictors['all_setting+screens'] = predictors['all_setting'] + screens
        predictors['common'] = ['Tender', 'Bid_value', 'Winner', 'Number_bids']

```

```
722     elif dataset == 'american':
723         df_collusion = pd.read_csv(db_collusion_american, header=0)
724         predictors['all_setting'] = ['Tender', 'Bid_value',
725             'Bid_value_without_inflation',
726             'Bid_value_inflation_raw_milk_price_adjusted_bid',
727             'Date', 'Winner', 'Number_bids']
728         predictors['all_setting+screens'] = predictors['all_setting'] + screens
729         predictors['common'] = [
730             'Tender', 'Bid_value', 'Winner', 'Date', 'Number_bids']
731 
732     elif dataset == 'japan':
733         df_collusion = pd.read_csv(db_collusion_japan, header=0)
734         predictors['all_setting'] = ['Tender', 'Bid_value',
735             'Pre-Tender Estimate (PTE)', 'Difference Bid/PTE',
736             'Site', 'Date', 'Winner', 'Number_bids']
737         predictors['all_setting+screens'] = predictors['all_setting'] + screens
738         predictors['common'] = [
739             'Tender', 'Bid_value', 'Winner', 'Date', 'Number_bids']
740 
741     elif dataset == 'all':
742         df_collusion = pd.read_csv(db_collusion_all, header=0)
743         predictors['common'] = [
744             'Tender', 'Bid_value', 'Winner', 'Number_bids', 'Dataset']
745 
746     predictors['common+screens'] = predictors['common'] + screens
747 
748     # Output fields of the datasets to the ML algorithms.
749     targets = ['Collusive_competitor']
750 
751     return df_collusion, predictors, targets
752 
753 
754 #if __name__ == '__main__':
755 
756 # The user selectes the dataset to analyse
757 dataset = None
758 while dataset == None:
759     number_input = input('Insert the number to analyse the dataset ' \
760                          '[brazilian_comprasnet (0),brazilian (1), american (2), ' \
761                          'italian (3), switzerland_gr_sg (4), ' \
762                          'switzerland_ticino (5), japan (6), all datasets (7) ' \
763                          'or exit (E)]: ')
764     if number_input == 'E': sys.exit(0)
765     elif number_input == '0': dataset = 'brazilian_comprasnet'
766     elif number_input == '1': dataset = 'brazilian'
767     elif number_input == '2': dataset = 'american'
768     elif number_input == '3': dataset = 'italian'
769     elif number_input == '4': dataset = 'switzerland_gr_sg'
770     elif number_input == '5': dataset = 'switzerland_ticino'
771     elif number_input == '6': dataset = 'japan'
772     elif number_input == '7': dataset = 'all'
773 
```

```
# 1. Get the dataset processed ready to use with the ML algorithms
df_collusion, predictors, targets = get_dataset(dataset)

# 2. Print information of the processed datasets
print_description_processed_dataset(df_collusion)

# 3. Print list with the colluded tenders by bidder
#calculate_colluded_tenders_by_bidder(df_collusion)

# 4. Print Scatter Matrix
df_scatter_matrix = shuffle_tenders(df_collusion)
# Columns to plot (Collusive_competitor is deleted at the end)
columns_to_plot = ['Bid_value'] + screens + ['Collusive_competitor']
df_scatter_matrix = df_scatter_matrix[columns_to_plot]
# Replace labels for colors to print the scatter matrix
if 'Collusive_competitor_original' in df_scatter_matrix:
    df_scatter_matrix['Collusive_competitor'] =
        df_scatter_matrix['Collusive_competitor_original']
colors_legend = ['Green', 'Red']
labels_legend = ['Competitive bid', 'Collusive bid']
df_color_labels = df_scatter_matrix[
    ['Collusive_competitor']].replace(0, colors_legend[0])
df_color_labels = df_color_labels[
    ['Collusive_competitor']].replace(1, colors_legend[1])
list_color_labels = df_color_labels['Collusive_competitor'].values.tolist()
df_scatter_matrix.drop(columns=['Collusive_competitor'], inplace=True)
printScatterMatrix(
    df_scatter_matrix, list_color_labels, colors_legend, labels_legend, dataset)

# 5. Boxplots of screen variables
# Check with len of screens
max_ylim_screens = [0.4, 1.8, 0.3, 12, 4, 4, 1]
min_ylim_screens = [0, 0, 0, -12, -4, -4, 0]
step_y_screens = [0.4/10, 1.8/10, 0.3/10, 24/10, 8/10, 8/10, 1/10]
df_collusion_copy = df_collusion.copy()
df_collusion_copy['Collusive_competitor'].replace(0, 'Comp.', inplace=True)
df_collusion_copy['Collusive_competitor'].replace(1, 'Coll.', inplace=True)
for index, screen_variable in enumerate(screens):
    print_boxplot(df_collusion_copy, dataset, column_names=screen_variable,
        groupby='Collusive_competitor', min_ylim=min_ylim_screens[index],
        max_ylim=max_ylim_screens[index], step_y=step_y_screens[index],
        xlabel='Bids', percentage=True)

# 6. Histogram or density plot of Number of Bids by Tender.
# Each plot for collusive tenders and honest tenders.

df_hist = df_collusion
if 'Collusive_competitor_original' in df_hist:
    df_hist['Collusive_competitor'] = df_hist['Collusive_competitor_original']
competitive_bids = df_hist[df_hist['Collusive_competitor'] == 0]['Number_bids']
collusive_bids = df_hist[df_hist['Collusive_competitor'] != 0]['Number_bids']
plotTwoHistograms(competitive_bids, collusive_bids, label_1='competitive bids',
```

```
826     label_2='collusive bids', max_range=125, bins=25, max_xlim=125, density=True)
827
828 # 7. Execute algorithm comparison and print table comparison
829 algorithm_comparison(df_collusion, dataset, predictors, targets, ml_algorithms,
830     train_size, repetitions, n_estimators, precision_recall, load_data, save_data)
```

V. *DATASETS* E SCRIPTS

Os scripts utilizados neste experimento, adaptados do trabalho de García Rodríguez *et al.* [50], bem como os *datasets* gerados, estão disponíveis no github do autor: <https://github.com/rodrigovilela/collusion>.