Universidade Federal Fluminense

DAYANE BRAGA ALVES

Identificação de diversidade de chicotes veiculares borda de linha através da mineração de dados

VOLTA REDONDA

2022

DAYANE BRAGA ALVES

Identificação de diversidade de chicotes veiculares borda de linha através da mineração de dados

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional em Ciência e Tecnologia da Universidade Federal Fluminense, como requisito parcial para obtenção do título de Mestre em Modelagem Computacional em Ciência e Tecnologia. Área de Concentração: Modelagem Computacional.

Orientador:

Eliane da Silva Christo

Coorientador:

Tiago Araújo Neves

Universidade Federal Fluminense

VOLTA REDONDA

2022

Ficha catalográfica automática - SDC/BEM
Gerada com informações fornecidas pelo autor

A474i Alves, Dayane Braga
Identificação de diversidade de chicotes veiculares borda de linha através da mineração de dados / Dayane Braga Alves. - 2022.
79 f.

Orientador: Eliane da Silva Christo.
Coorientador: Tiago Araújo Neves.
Dissertação (mestrado)-Universidade Federal Fluminense, Escola de Engenharia Industrial e Metalúrgica de Volta Redonda, Volta Redonda, 2022.

1. Diversidade de chicotes. 2. Mineração de dados. 3. Regras de associação. 4. Produção intelectual. I. Christo, Eliane da Silva, orientadora. II. Neves, Tiago Araújo, coorientador. III. Universidade Federal Fluminense. Escola de Engenharia Industrial e Metalúrgica de Volta Redonda. IV. Título.

CDD - XXX

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

Identificação de diversidade de chicotes veiculares borda de linha através da mineração de dados

Dayane Braga Alves

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional em Ciência e Tecnologia da Universidade Federal Fluminense, como requisito parcial para obtenção do título de Mestre em Modelagem Computacional em Ciência e Tecnologia. Área de Concentração: Modelagem Computacional.

Aprovada por:

Prof. Eliane da Silva Christo, D.Sc. / MCCT-UFF
(Orientadora)

Prof. Tiago Araújo Neves, D.Sc. / MCCT-UFF
(Co-orientador)

Prof. Cecilia Toledo Hernandez, D.Sc. / MCCT-UFF

Prof. Rafael Alves Bonfim de Queiroz, D.Sc. / DECOM-UFOP

Prof. Leonardo Goliatt da Fonseca, D.Sc. / MAC-UFJF

Volta Redonda, Dezembro de 2022.

"Há uma força motriz mais poderosa que o vapor, a eletricidade e a energia atômica: a vontade."

Albert Einstein

Agradecimentos

Gostaria de agradecer primeiramente à Deus, pois Ele me deu forças e me mostrou que eu era capaz de realizar todos meus objetivos sem medir esforços, foi Ele quem me deu forças durante acontecimentos inesperados em minha vida pessoal e me deu estabilidade para lidar com as dificuldade e mudanças diversas, e ter energias para continuar de cabeça erguida para concluir meu curso deixando os problemas de lado.

À minha Família, principalmente à minha mãe Irani, meu pai Jorge e, não poderia faltar, meu noivo Thiago, que acompanhou toda minha luta bem de perto, sempre acreditando, apoiando e incentivando a comprimir todas minhas metas e objetivos, me ajudando sempre que possível ao longo dessa jornada.

Aos professores, principalmente Eliane Christo e Tiago Neves, que me deram todo o suporte, mostrando os melhores caminhos, orientando da melhor forma possível, com seus conhecimentos e compartilhamento de experiências, que enriqueceram meus estudos para este trabalho, contribuindo com toda a base para a realização deste sonho.

E aos amigos, principalmente do trabalho anterior, que deram origem para esta proposta, e que me acompanharam direta ou indiretamente, e em especial à engenheira Fabiana Antunes que acompanhou mais de perto toda a evolução. E claro, a todos os professores do MCCT que fizeram parte dessa minha trajetória.

Obrigada!

Dayane Braga Alves

Dedico meu trabalho à minha família, que com todo carinho me apoiaram e acompanharam de perto todo meu esforço.

Resumo

Todo veículo automotor é composto de partes mecânicas e elétricas, sendo toda ligação da parte elétrica e comunicação dos módulos eletrônicos realizada por conjuntos de fios, denominados de chicotes elétricos, na qual compõem todos os circuitos responsáveis pela comunicação da rede e alimentação dos mesmos. Na fase de industrialização dos veículos, dependendo da versão de veículo que será comercializada, a quantidade de circuitos a serem disponibilizados em cada um destes chicotes varia dependendo da quantidade de módulos eletrônicos e/ou características específicas de cada versão, ou seja, dentro de um projeto veicular, durante a fase de concepção é necessário prever toda a diversidade (ou variantes) de chicotes para atender cada versão a ser produzida, garantindo a funcionalidade e integração de todas as funções de determinada versão. Com o intuito de automatizar e agilizar este processo da engenharia, a proposta dessa dissertação foi a modelagem e desenvolvimento de um algoritmo para rápida identificação dessa diversidade de chicotes elétricos borda de linha para maior família de chicotes (chicote principal), a partir dos dados entrada provenientes da definição do produto, com a lista de características do projeto, amarrando todos os *features* impactados por versão de veículo nos projetos. A partir da aplicação da Mineração de Dados nos dados entrada de cada projeto, por meio das regras de associação, com modelos interpretáveis e tradicionalmente explicáveis, obteve-se os resultados esperados, com a disponibilização das informações de diversidade de chicotes a partir de simulação com dados sintéticos de entrada. Dados robustos desde a fase inicial de um projeto para primeiros *approaches* com fornecedores e recebimento de boas amostras de peças para construção de mulas (carros prototipados para testes nas fases iniciais), evita-se futuros retrabalhos e renegociações comerciais, o que geraria custos extras não previstos inicialmente, reduzindo principalmente além de custos, o tempo de concepção do projeto.

Abstract

All motor vehicle is contained by mechanical and electrical parts, being all electrical connection and electronic modules communication performed by sets of wires named wire harness, they compose all circuits responsible for network communication and power supply. In the vehicles industrialization step, according to the version that will be commercialized, the number of circuits to be provided in each variant of wire harness changes according the number of modules and/or specific features of each version. It means according to each project, during the conception step, it's required to identify all variants of wire harness to attend each version to be assembled, to ensure the functionalities and integration of all functions for each specific version. Having the purpose of automating and expediting the engineering process, the proposal of this dissertation was modeling and development of an algorithm to expedite the variants identification of the assembly line edge's wire harness of the biggest harness family (main harness), according to the inputs from product's definition that provides the list with all projects' features and have all these features linked with all possible vehicles versions defined. Starting from the application of Data mining in the inputs data from each project, by the association rules, with interpretable and traditionally explainable models, the expected results were obtained, it was provided the information regarding the wire harness variants from the pseudo data inputs. Robust data since the initial phase of a project to start approaches with suppliers and make prototypes cars, it avoids future rework and commercial renegotiation, which could generate extra unforeseen costs, reducing mainly the time of engineering conception.

Palavras-chave

1. Diversidade de chicotes;
2. Mineração de Dados;
3. Regras de Associação.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

RFQ	*Request For Quotation* (Solicitação de Cotação para fornecedor)
KDD	*Knowledge Discovery in Database* (Conhecimento em Bases de Dados)
ARL	*Association Rules Learning* (Aprendizado de Regras de Associação)
TID	*Transactions Identification* (Identificador de transação)
VBA	*Visual Basic for Applications* (Visual Basic para Aplicações)
CPU	*Central Processing Unit* (Unidade Central de Processamento)

Glossário

Features	Características ou atributos
Approach	Abordagem ou primeira tratativa de uma proposta ou pedido
Chicote	Conjuto de fios da ligação elétrica veicular
peça borda de linha	peça pronta para que o operador possa montar no veículo
Ranking	Classificação
Inputs	Dados de Entrada
Clustering	Agrupamento
String	Texto
Insights	Percepções
Engine	Motor
Vehicle	Veículo
Steering	Direção
Brakes	Freios
Motherboard	Placa-mãe
Machines	Máquinas
Appliances	Eletrodomésticos
Wheels	Rodas
Thank	Tanque

Seats	Bancos
Camera	Câmera
Cooler	Refrigerador
Computer	Computador
Monitor	Monitor
Pens	Canetas
Ink	Tinta
Pencils	Lapis
Desk	Mesa
Typewriter	Máquina de escrever
Brushes	Escovas
Office	Escritório
Black Boxes	Caixas preta
Null	Nulo

Capítulo 1

Introdução

O desenvolvimento de produtos é um processo pelo qual uma organização transforma as oportunidades de mercado e suas possibilidades técnicas em informações para a fabricação de um produto comercial [3]. Sendo este processo um conjunto de atividades por meio das quais se busca chegar às especificações do projeto de produto e de seu processo de produção. [3].

Mundialmente a indústria automobilística é reconhecida como uma das alavancas da economia [4]. No Brasil não poderia ser diferente, justificada pela ascensão que o segmento obteve a partir de 2006, quando o Brasil era apenas o décimo colocado no ranking automotivo mundial e ao final de 2010 o quarto colocado em unidades vendidas, atrás apenas da China, dos Estados Unidos e do Japão [4].

Neste seguimento, dentre alguns importantes desenvolvimentos de peças, tem-se o desenvolvimento de chicotes elétricos, conjunto de fios responsáveis por toda ligação elétrica e comunicação em rede de todos componentes elétricos e eletrônicos dos veículos, conforme pode ser observado na Figura 1.1. Para atendimento dos requisitos dos projetos, tem-se como parte do seu processo a identificação da diversidade de chicotes, na qual garante-se para cada versão o completo funcionamento de todo o sistema e documentos robustos para lançamento das RFQs (*Request For Quotation* - Solicitação de Cotação para fornecedor) e negociação com potenciais fornecedores.

De acordo com Wright, Kroll e Parnell (2000) *apud* [3], uma vantagem competitiva pode ser sustentada por diferenciação ou liderança de custos. Ambas as estratégias genéricas pressupõem a complexa atividade de desenvolver produtos. Logo, o fato de se ter informações robustas e precisas desde os dados de entrada iniciais de um projeto, dar-se-á uma vantagem competitiva, evitando-se custos com correções de definições defi-

Figura 1.1: Diagrama ilustrativo do desenvolvimento do produto comercial

cientes e ganho de tempo, recurso importante para o planejamento de desenvolvimentos de produtos.

A problemática apresentada utiliza como referência uma empresa do ramo automotivo, que atualmente, faz muito de seus mecanismos de controles através de diversas planilhas com diferentes formatações, que variam de acordo com cada responsável por determinado projeto/documento. Hoje não há nenhuma metodologia ou algum tipo de solução aplicada neste ramo que auxilie os engenheiros nessa atividade. A proposta de solução para esta problemática é justamente automatizar esse processo gerando um algoritmo que reduza o tempo gasto nesta etapa de desenvolvimento dos chicotes.

O foco principal é aplicação inicialmente do algoritmo para identificação da maior família de chicotes, o chicote principal que, conforme será apresentado no capítulo 2.1, é o chicote que faz a maior trajetória e interconexão com todos, ou maioria, das demais famílias de chicotes. Sendo assim, o mesmo é o que mais gera necessidades de variantes, o que consequentemente, gera o maior tempo gasto nesse processo de identificação de variantes durante o desenvolvimento dos projetos.

1.1 Objetivos

O objetivo principal desta dissertação é identificação de diversidade de chicotes principais borda de linha de forma automatizada, com a boa relação de todos os módulos de circuitos das características envolvidas para cada versão de veículo de acordo com cada projeto.

1.1.1 Objetivos Específicos

Como objetivos específicos deste estudo:

1. Modelar a partir do uso de métodos de regra de associação e agrupamento através de modelos interpretáveis e facilmente compreensíveis;
2. Desenvolver um algoritmo para ferramenta Excel® normalmente utilizada pela empresa utilizada como referência; e
3. Obter a rápida identificação das variantes (ou diversidade) de chicotes veículares por projeto, com a automatização do processo, reduzindo o tempo desta etapa do processo.

1.2 Limitações

Por motivos de confidencialidade e esta atividade de concepções de chicotes estar ligada a novos projetos, a empresa automobilística utilizada como referência para este estudo não será identificada, e as informações apresentadas serão meros exemplos a partir de dados artificiais baseados nos conceitos gerais de industrialização de veículos, a fim de que o algoritmo apresentado possa ser adaptado futuramente e reutilizado em outras empresas automobilísticas.

1.3 Estrutura da Dissertação

Esta dissertação está estruturada em 6 capítulos, sendo o primeiro com a Introdução e contextualização do tema abordado, incluindo a apresentação dos objetivos e suas limitações. O segundo capítulo contém toda a fundamentação teórica, de forma que o leitor conheça um pouco mais sobre os conceitos relacionados à concepção de chicotes e características industriais de uma automobilista. Já no capítulo 3, explica-se os métodos a serem utilizados como base para implementação do algoritmo a ser apresentado, como solução para obter-se os objetivos traçados, contendo todas as adaptações necessárias. No quarto capítulo serão apresentados os resultados obtidos e análise destes dados. Por fim, no capítulo cinco, serão apresentados as conclusões e propostas de trabalhos futuros, e no próximo e último capítulo, temos a relação de todas as referências bibliográficas utilizadas durante a realização deste trabalho.

Capítulo 2

Fundamentação Teórica

Os chicotes elétricos conectam todos os componentes elétricos e eletrônicos de um veículo, possibilitando o fluxo de comunicação e informação, bem como seu fornecimento de energia. No passado, o chicote costumava ser um produto comódite, orientado pelo preço, no entanto, a direção autônoma e a eletrificação estão agora induzindo uma mudança de paradigma para esta indústria [5].

Na Figura 2.1, uma imagem ilustrativa de um chicote veicular baseado em um veículo Bentley® Bentayga, com suas peças de chaparia ocultas, demonstrando apenas como seria todo o percurso do chicote no interior do veículo.

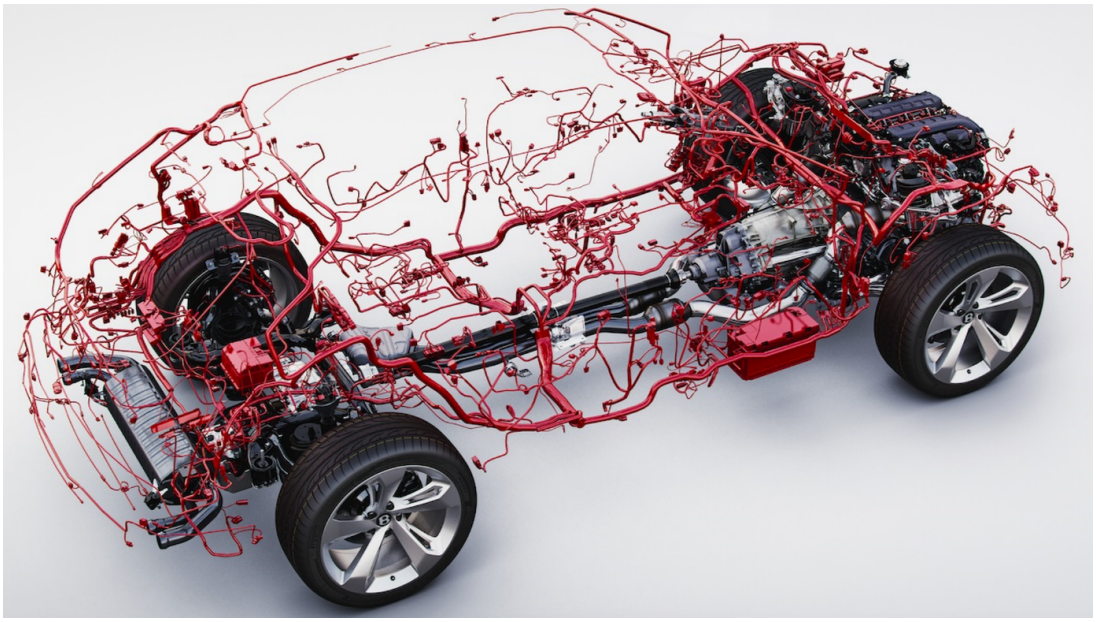

Figura 2.1: Exemplo de chicote veicular. Fonte: [6]

O estudo de caso apresentado por [5], realizado em 2019 com 54 especialistas, incluindo

26 empresas de 7 países, faz o levantamento das maiores fraquezas hoje envolvidas no desenvolvimento de chicotes veiculares. Dentre os resultados, como pode ser observado na Figura 2.2, em primeiro lugar vem como maior fraqueza a descontinuidade do processo e dos dados, na qual muitos participantes afirmam que não existe um processo holístico para o desenvolvimento do chicote com formatos de dados díspares que dificultam um fluxo de processo contínuo. Em segundo lugar está a complexidade, visto a quantidade crescente de componentes que leva ao aumento de variantes (diversidade) de chicotes, sendo assim, lidar com essa complexidade e gerenciamento de mudanças frequentes de forma eficiente afeta sobremaneira suas operações no que diz respeito à agilidade, qualidade e custos, no desenvolvimento e na produção destes chicotes.

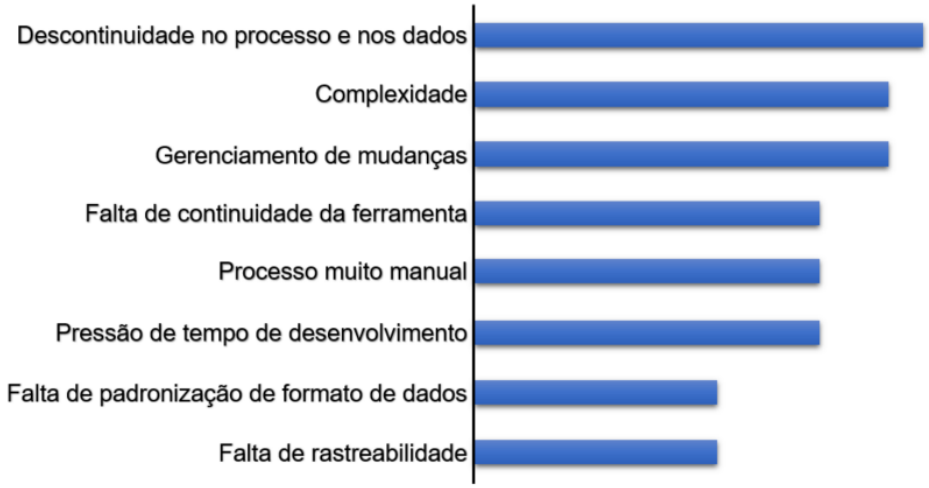

Figura 2.2: Maiores fraquezas do desenvolvimento de chicotes. Fonte adaptada de [5].

Conforme colocado por [5], em terceiro lugar como fraqueza no desenvolvimento de chicotes está a o gerenciamento de mudanças, na qual os entrevistados concordam que este impacto negativo se deve ao fato das mudanças serem gerenciadas manualmente na maioria das vezes e provocadas pelo processo de desenvolvimento.

A empresa automotiva abordada como referência ao estudo, utiliza no processo de concepção de chicotes, para o gerenciamento de todas as variantes ou diversidade de chicotes, um processo manual gerenciado pelo engenheiro de chicotes responsável por determinado projeto, tendo como ferramenta o Excel®, com manipulação dos dados de cada um dos chicotes por planilhas, correlacionando manualmente quais módulos de circuitos eletrônicos devem pertencer ou não a cada versão de chicotes por famílias.

Na Figura 2.3, uma foto com um exemplo de chicote veicular para auxiliar na compreensão do conceito e seguimento dos capítulos seguintes ilustrando aos leitores que ainda

não tinham claramente a ideia desse conjunto de fios. Nela é possivel identificar vários componentes associados ao chicote, como a caixa de fusíveis/relés e diferentes tipos de conectores, para conectar em módulos e/ou interconexões entre chicotes.

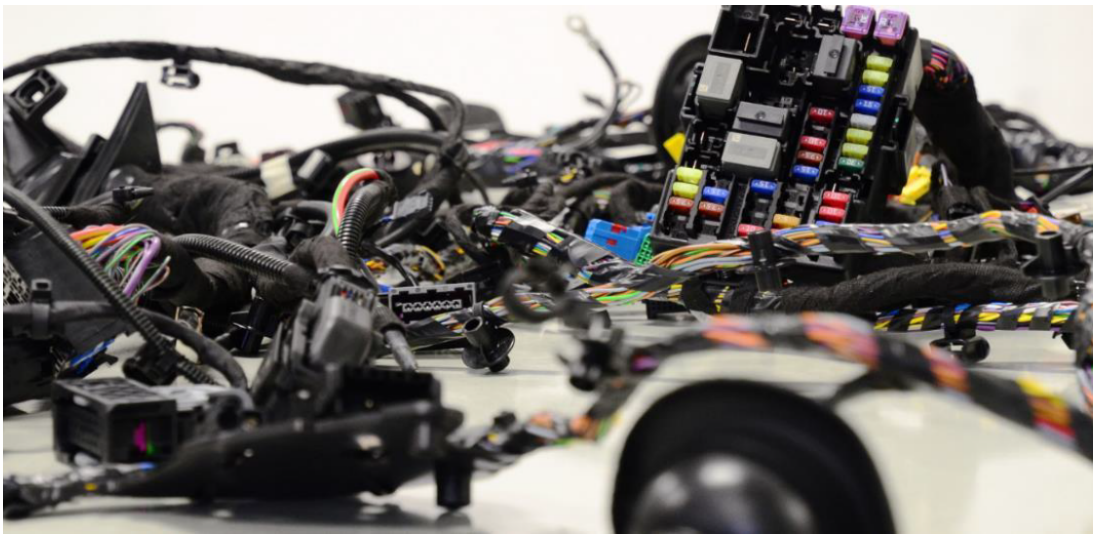

Figura 2.3: Foto ilustrativa de um chicote veicular. Fonte: [5].

2.1 Etapas do processo de desenvolvimento de chicotes

No processo de desenvolvimento de um chicote elétrico veicular existem várias etapas, e para se chegar nas variantes ou diversidades de chicotes necessárias para se montar todas as versões de veículo, são indispensáveis alguns os dados de entrada do projeto. Dependendo do tipo de segmento do veículo e a qual mercado ele está destinado, influencia na quantidade de *features* (características técnicas e/ou opcionais) que esse veículo possa ter, logo o time de arquitetura deste projeto precisa identificar qual o tipo de arquitetura será aplicado, dado de entrada importante para o time de chicotes identificar o passo inicial de como será para a concepção do mesmo, a divisão por famílias de chicotes.

Os chicotes são divididos por famílias com interconexões entre os mesmos, ou seja, conectores responsáveis pela continuidade dos circuitos de um chicote para outro, visto que fazer um chicote único para todo veículo seria inviável para montabilidade de um operador devido seu alto peso, e ainda acessos para roteamento e passagens de todos os conjuntos de circuitos por todo o perímetro do veículo até o destino final. Para isso usualmente, um chicote principal faz a ligação da parte interna da zona da cabine com a parte externa da zona motor, veja ilustração da Figura 2.4, atravessando todo o extensão do veículo, e demais chicotes conectam-se a este para continuidade do percurso, sejam

para irem até as portas, painel de instrumentos ou ainda motor.

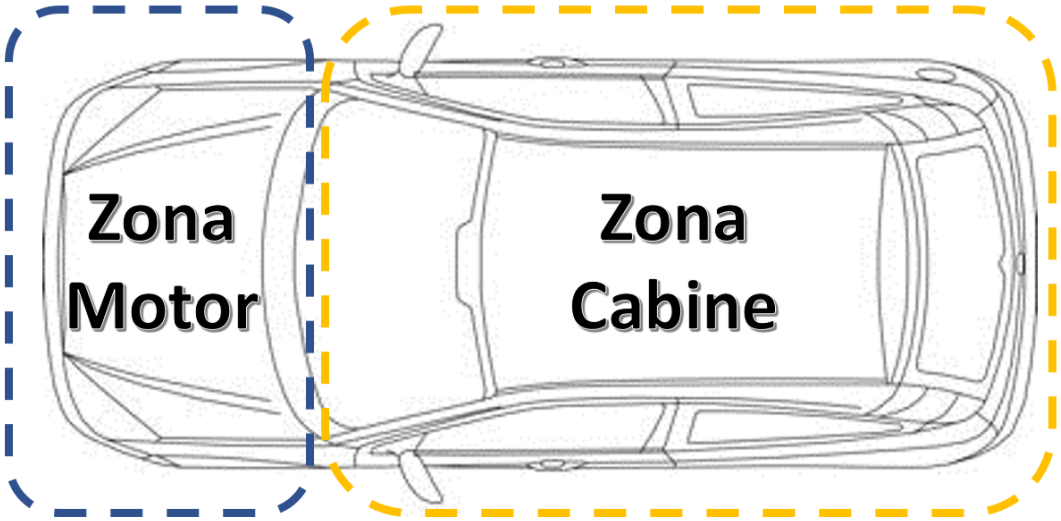

Figura 2.4: Ilustração da divisão para zona interna (cabine) e externa (motor) do veículo.

Na Figura 2.5, exemplifica-se como seria uma divisão de chicotes a partir de uma aplicação de determinada arquitetura veicular, podendo ter até 8 famílias de chicotes:

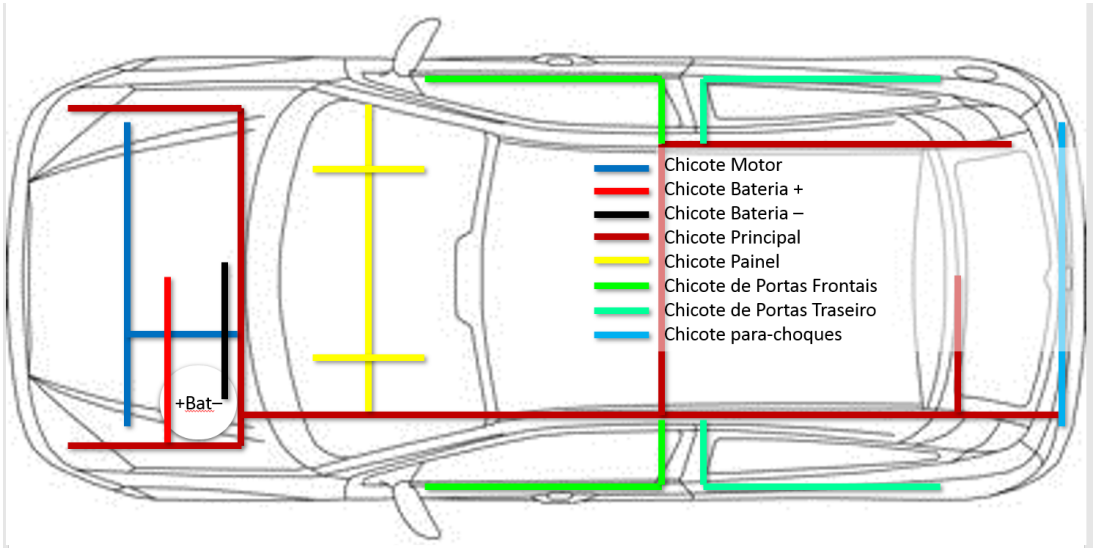

Figura 2.5: Exemplo de divisão de famílias de chicotes.

Conforme ilustrado na Figura 2.5, como o chicote principal é o que faz a maior trajetória e interconexão com todos, ou maioria, das demais famílias de chicotes, o mesmo é o que mais tem variantes, ou seja, há maior diversidade de peças borda de linha a ser disponibilizado devido a variedade de versões possíveis de veículos a serem montadas. Na

qual o engenheiro precisa garantir que todos os módulos de circuitos necessários estejam devidamente considerados ou não em cada versão.

O time de produto é quem disponibiliza o documento que informará as engenharias quais versões de veículos existirão para determinado projeto e quais as características técnicas associadas a cada versão de veículo, sejam elas, versão que terá multimídia, quantos alto falantes (apenas frontais ou traseiros também), se a versão terá ou não câmera de ré, quais os tipos de motores por versão, etc. A partir destes dados o arquiteto elétrico prepara todos os desenhos por módulos e disponibiliza ao time de chicotes todos os circuitos necessários para cada módulo, com origem e destino para que a engenharia de chicotes faça a amarração destes módulos por grupo de famílias de chicotes.

2.2 Características técnicas e seus códigos industriais

De acordo com a definição do projeto a ser desenvolvido, conforme já relatado anteriormente, o engenheiro de chicotes deve identificar toda a diversidade de chicotes e os módulos de circuitos por famílias a partir das características técnicas aplicadas a cada versão de veículo, como por exemplo, para o chicote principal se o motor A possui câmbio manual e automático, só para a versão de veículo com motorização A, o chicote deve conter duas variantes (Manual e Automática); se este mesmo motor (A), possui versões com combustível Flex Fuel e ainda apenas Gasolina, para ambas possibilidades de câmbio, essa variante passa a ser dobrada, sendo que apenas para a motorização A, quatro variantes seriam necessárias, mas normalmente há mais opções de motores envolvidas por projetos. Por este fato, o chicote principal é sempre o chicote de maior diversidade borda de linha.

Baseado na empresa automobilista de referência deste estudo, para informar o sistema de produção a versão de veículo a ser produzida, é disponibilizado por versões uma ficha de conjunto de códigos de identificação, ou seja, cada *feature* tem cadastrado no sistema um código padrão de referência. Por exemplo, o motor do tipo A é referenciado por um código em todas as unidades fabris. Neste sentido, todos os projetos que utilizam este tipo de motor deverão referênciá-lo por este código. A formação do código se dá por meio de números e letras, podendo ser por exemplo “MOT01”, sendo todo código sempre constituído pela mesma quantidade de caracteres, onde os dígitos iniciais são os identificadores do *feature* e os dígitos finais o valor correspondente ao tipo ou versão do *feature* em questão, conforme ilustrado pela Figura 2.6.

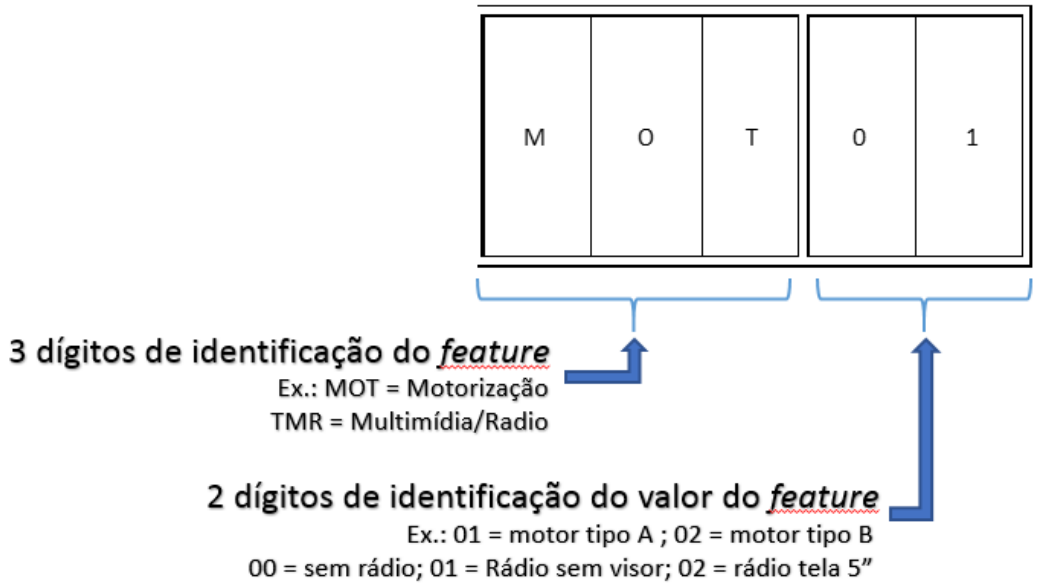

Figura 2.6: Código de identificação das características técnicas.

Sendo assim, uma determinada versão de veículo poderia ser descrita da seguinte forma:

a MOT01; CMBGS; CXCMA; ... => Motorização tipo A, com tipo de combustível apenas gasolina, câmbio manual, etc...

b MOT02; CMBFF; CXCAT; ... ... => Motorização tipo B, com tipo de combustível Flex Fuel, câmbio automático, etc...

Nos Apêndices A e B é possível verificar as Tabelas A.1 e B.1 contendo os dados sintéticos representando diversos desses códigos indústriais.

2.3 Identificação das diversidades de chicotes

Conforme os dados já apresentados, a partir de um modelo padronizado de planilha no Excel®, o engenheiro de chicotes de cada projeto vai preenchendo manualmente e avaliando que variante ou diversidade é necessário considerar criação e quais módulos de características técnicas são necessários amarrar a cada versão destas variantes, processo manual, trabalhoso e delicado, para que abranja todas as versões necessárias sem que faltem ou sobrem circuitos, durante a montagem de alguma versão. Lembrando que, por exemplo, ao entregar chicotes com módulos de circuitos de funções extras sem utilidades

naquela versão, são custos em metragens de fios e conectores desnecessários agregados ao mesmo e ainda haveriam mais custos para neutralização do mesmo, pois solto (o conector não haveria onde conectar) geraria ruídos dentro do veículo.

Como resultado, o engenheiro tem que chegar em uma tabela próxima ao representado pela Tabela 2.1, com um exemplo do resultado de diversidade para o chicote principal, temos uma representação genérica com variantes de A à F, ou seja, 6 variantes, onde a coluna "versão de veículo" poderia ser correlacionada ao exemplo do subtópico anterior, onde "123" poderia estar sendo representado pela sequência apresentada no subtópico 2.2.a, e "132" estar sendo representado pela sequência apresentada no subtópico 2.2.b, onde cada um dos módulos remarcados com "x" representaria cada uma das características técnicas da sequência de código de cada versão de veículo. Sendo assim, conforme representado no subtópico 2.2.b, a versão "132" estaria com um "x" no módulo de circuitos da motorização tipo B (MOT02), outro em Flex Fuel (CMBFF) e um outro "x" no módulo de circuitos da caixa de câmbio automática (CXCAT), por exemplo.

	Versão de veículo	Mód.1	Mód.2	Mód.3	...	Mód.38	Mód.39
Chicote principal A	123	x		x	...	x	
Chicote principal B	132	x			...		x
Chicote principal C	213	x		x	...		x
Chicote principal D	231		x		...	x	
Chicote principal E	321		x	x	...	x	x
Chicote principal F	312		x		...		x

Tabela 2.1: Tabela Manual de correlação de diversidade de chicotes principal

A fim de se chegar nesse resultado de forma rápida e robusta, este trabalho visa desenvolver um algoritmo capaz de chegar nos mesmos resultados, identificando a quantidade necessária de variantes, e quais os módulos de circuitos de características técnicas farão parte de cada uma destas variantes. Tendo os dados de entrada para início das negociações com fornecedores logo que todos os dados necessários para obtenção destas variantes estejam disponíveis, e ainda, em caso de mudanças de escopo, rapidamente gerenciar o mesmo, aplicando os novos dados no algoritmo e obtendo os novos resultados, partir dos novos dados de entrada.

2.4 Explicabilidade e Interpretabilidade

Antes de apresentar as metodologias elegídas para a solução da problemática descrita, é importante apresentar os conceitos que fundamentaram a decisão pelos métodos. Con-

forme apresentado por [7], a sociedade tem se despertado para o problema da lacuna de explicações acerca do comportamento de sistemas inteligentes, sendo alguns destes sinais: a formação de um consórcio de parceria em IA, em novembro de 2016, para beneficiar as pessoas e a sociedade (HERN, 2016 *apud* [7]) por parte de algumas das maiores empresas de tecnologia do mundo (Google, Facebook, Amazon, IBM, Microsoft); e a legislação relativa à *General Data Protection Regulation* (GDPR - Regulamento Geral sobre a Proteção de Dados), que entrou em vigor em 2018 no âmbito da União Européia, induzindo um direito à explicação por parte do cidadão que seja "afetado significativamente" por decisões automatizadas tomadas por algoritmos preditivos no nível (individual) de um usuário (GOODMAN, 2016 *apud* [7]).

A compreensão de um sistema pode levar à correção de suas deficiências. Ao desenvolver um modelo, [8] destaca que, a consideração da interpretabilidade como um driver de design adicional pode melhorar sua implementação;[9] também coloca que a explicabilidade pode servir para muitos propósitos, e cita os 3 principais pontos de assistência:

1. Informar e ajudar o sujeito a entender por que uma determinada decisão foi tomada;
2. fornecer fundamentos para contestar decisões adversas; e
3. entender o que podem ser alterados para receber um resultado desejado no futuro, com base no modelo de tomada de decisão atual.

As explicações que suportam a saída de um modelo são cruciais, por exemplo, na medicina de precisão, onde os especialistas exigem muito mais informações do modelo do que uma simples previsão binária para apoiar seu diagnóstico (TJOA E. , GUAN C., 2019 *apud* [8]). Outros exemplos incluem veículos autônomos em transporte, segurança e finanças, entre outros. Conforme relatado por [10], explicações devem ser formuladas sob restrições pragmáticas, que incluem considerar os objetivos, crenças, contexto, de acordo com os usuários. Segundo [11] como a viabilização desses benefícios vêm com o aumento de custo de tempo e energia no desenvolvimento, alguns optam por ignorar estes fatores sociais inerentes aos modelos, o que podem gerar sérias implicações, incluindo questões judiciais, de responsabilidade e desalinhamento com os valores humanos (Char et al., 2018; Zafar et al., 2017; Leike et al., 2018 *apud* [11]) conforme previsto nos artigos da GDPR.

Conforme destacado por [8], uma das questões que dificultam o estabelecimento de bases comuns de entendimento sobre o que o termo explicabilidade significa no contexto de sistemas inteligentes, é o uso indevido intercambiável de interpretabilidade e explicabilidade na literatura. Assim sendo, abaixo o esclarecimento de algumas terminologias:

- Inteligibilidade: Capacidade de um modelo permitir que um humano compreenda sua função, sem a necessidade de explicar sua estrutura interna ou o processamento algorítmico;
- Compreensibilidade: Capacidade de um modelo representar o conhecimento aprendido de uma forma compreensível por humanos;
- Interpretabilidade: Capacidade de um modelo de explicar ou providenciar o significado em termos compreensíveis para humanos;
- Transparência: Capacidade de um modelo ser intrinsecamente compreensível;
- Explicabilidade: Capacidade de oferecer explicações como uma interface entre humanos e o modelo.

Entender um modelo induzido por computador, segundo [12], geralmente é um pré-requisito para que os usuários confiem nas previsões do modelo e sigam as recomendações associadas a essas previsões. A compreensibilidade do modelo, é importante para a aceitação do modelo pelos usuários em diversas aplicações, podendo por exemplo, aumentar as chances de que os usuários invistam muito tempo e dinheiro na execução de experimentos biológicos sofisticados para tentar confirmar as previsões de modelo de bioinformática. Indiscutivelmente, o valor final das previsões de um modelo para bioinformática é determinado pelo sucesso cumulativo dos experimentos inspirados por essas previsões (Jiang, T., and Keating, A.E., 2005 *apud* [12]).

Seguindo nas contextualizações, [8] destaca os objetivos da explicabilidade, sendo elas:

- Confiabilidade: Permitir avaliar o grau com que o modelo toma decisões conforme o esperado;
- Causalidade: Permitir encontrar a causalidade entre as variáveis de entrada e o resultado do modelo;
- Transferibilidade: Permitir descobrir as restrições do modelo para aplicá-los em outros contextos;
- Informatividade: Tornar acessível o conhecimento interno do modelo;
- Justiça: Permitir avaliar se o modelo é enviesado, se está em conformidade com padrões éticos, etc;

- Privacidade: Permitir avaliar se o modelo contém dados sensíveis que podem ser recuperados e vazados.

Doran et al. (2017) *apud* [7], identificam três classes de sistema possíveis, sendo eles: opacos, interpretáveis e compreensíveis. Os sistemas opacos seriam como caixas pretas, i.e., seus mecanismos não são inspecionáveis por usuários; Os sistemas interpretáveis, por outro lado, permitiriam inspeção, estudo e compreensão dos seus processos e mecanismos internos, mesmo que essas tarefas demandem certo conhecimento técnico especializado. Já os sistemas compreensíveis seriam aqueles que, além de oferecerem um resultado, também oferecem símbolos que são inteligíveis para os usuários, permitindo compreender por que uma certa saída está associada a uma certa entrada.

Apartir destes conceitos, visto que a problemática tem como proposta aplicação de solução em setor empresarial na qual visa-se a confiabilidade, assim como, transparência, buscou-se modelos que fossem interpretáveis e facilmente compreensíveis, e que atendessem os objetivos da próblemática, para que além da aceitação do modelo pelos usuários, futuramente, outros colaboradores pudessem propor melhorias ou mesmo manutenções de acordo com as necessidades da empresa.

Capítulo 3

Metodologia

Para chegar-se na resolução do problema proposto dessa etapa do desenvolvimento de chicotes que deseja-se obter a modelagem computacional, foram necessários dentro da mineração de dados a exploração de duas metodologias: uma de regras de associação, baseado no algoritmo *FPGrowth*, e de Análise de Agrupamentos.

3.1 Mineração de dados

O termo mineração de dados (MD), é de fato como alusão ao processo de mineração propriamente dito, uma vez que se explora uma base de dados (mina) usando algoritmos (ferramentas) adequados para obter conhecimento (minerais preciosos). Os dados são símbolos ou signos não estruturados, sem significado, como valores em uma tabela, e a informação está contida nas descrições, agregando significado e utilidade aos dados, como o valor da temperatura do ar [13]. De acordo com Elmasri e Navathe *apud* [14], mineração de dados é um termo que se refere à mineração ou descoberta de novas informações em uma grande quantidade de dados, usando padrões ou regras. Por fim, conforme colocado por [13], o conhecimento é algo que permite uma tomada de decisão para a agregação de valor, como por exemplo, saber que vai chover no fim de semana pode influenciar sua decisão de viajar, ou não, para a praia. Veja o exemplo da Figura 3.1.

Para ressaltar a importância da praticidade e confiabilidade da Mineração de Dados, cita Goldshmidt (2011) *apud* [15], que com a grande quantia de informação que se aglomera em grandes bases de dados faz-se necessário o estimulo sobre o estudo da Mineração de Dados e suas técnicas, buscando conhecimento que auxiliarão na competitividade de serviços de empresas e micro empresas podendo assim melhorar seus negócios, traduzindo-se em diferenciais mercadológicos trazendo lucros. Reforçando esta ideia Gomes (2008)

apud [15], cita que com a crescente informatização no mundo em todos os tipos de negócio, o volume de dados armazenados é enorme, e tornam-se cada vez mais difíceis de serem trabalhados com formas e métodos tradicionais, sendo assim, para contornar tais problemas tem-se os conceitos de Mineração de Dados, onde faz parte do processo de Descoberta de Conhecimento em Bases de Dados (DCBD) / Knowledge Discovery in Database (KDD) que tem a finalidade de auxiliar na busca e descoberta de conhecimento e informações úteis em grandes volumes de dados.

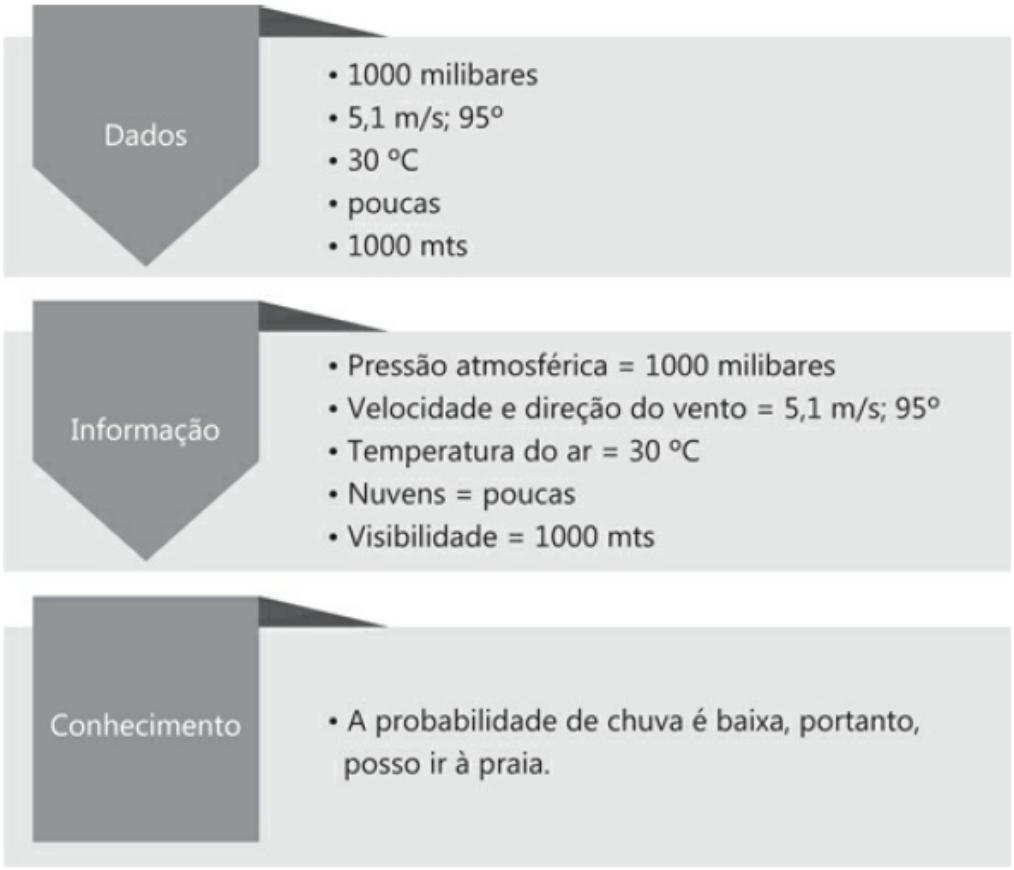

Figura 3.1: Exemplo da diferença entre dados, informação e conhecimento. Fonte: [13]

Para o auxílio e explicação da função do KDD Usama (1996) *apud* [15], confirma que o KDD é o conjunto de métodos desenvolvidos para buscar nos dados informações úteis e que façam sentido, sendo sua utilidade principal mapear os dados tornando-os mais compactos e mais fáceis de trabalhar, aplicando assim a Mineração de Dados a fim de buscarem dados relevantes.

Em [16], conforme colocado por WITTEN e FRANK, KDD pode ser definido como sendo o processo de identificação de padrões válidos, novos, potencialmente úteis e com-

preensíveis presentes nos dados sendo analisados e tratados. A Figura 3.2 apresenta resumidamente as etapas envolvidas neste processo.

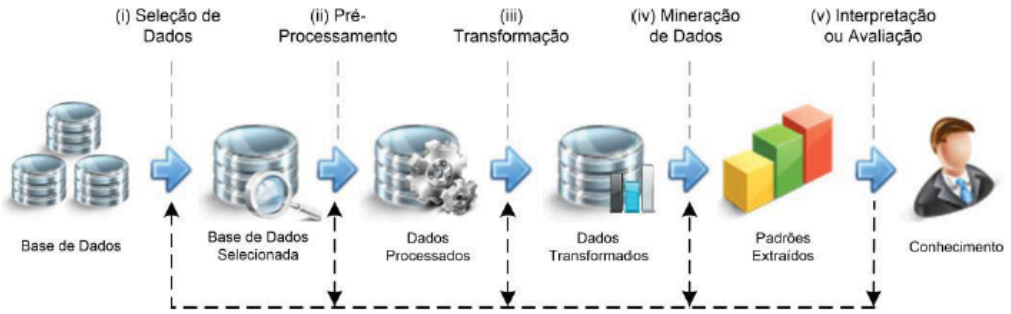

Figura 3.2: Etapas do processo KDD. Fonte: [16]

3.2 Agrupamento de Dados

Agrupamento (*clustering*) é o nome dado ao processo de separar (particionar ou segmentar) um conjunto de objetos em grupos (do inglês *clusters*) de objetos similares. Diferentemente da tarefa de classificação, o agrupamento de dados considera dados de entrada não rotulados, ou seja, o grupo (classe) ao qual cada dado de entrada (objeto) pertence não é conhecido inicialmente. O processo de agrupamento (ou clusterização) normalmente é utilizado para identificar tais grupos e, portanto, cada grupo formado pode ser visto como uma classe de objetos [13].

O Agrupamento é uma técnica mais "primitiva" na qual nenhuma suposição é feita a respeito dos grupos. Ao contrário da classificação, a Agrupamento não conta com classes predefinidas e exemplos de treinamento de classes rotuladas, sendo assim realiza uma forma de aprendizado não supervisionado [17]. De acordo com VERMA et al. *apud* [18], *Clustering* pode ser considerada a mais importante técnica de aprendizagem não supervisionada dentre as demais técnicas desse tipo, onde o agrupamento é o processo de organização de objetos em grupos cujos membros são semelhantes de alguma forma.

Em um processo de agrupamento, os objetos são agrupados com o objetivo de maximizar a distância interclasse e minimizar a distância intraclasse, ou, dito de outra forma, maximizar a similaridade intraclasse e minimizar a similaridade interclasse. Portanto, um cluster pode ser definido como uma coleção de objetos similares uns aos outros e dissimilares aos objetos pertencentes a outros clusters [13].

HRUSCHKA EBECKEN (2001) *apud* [17], apresenta uma definição formal do problema de Agrupamento, na qual, considerando um conjunto de n objetos $X = \{X_1, X_2, \ldots, X_n\}$ onde cada $X_i \in R^p$ é um valor de p medidas reais que dimensionam as características do objeto, estes devem ser agrupados em k clusters disjuntos $C = \{C_1, C_2, \ldots, C_k\}$, de forma que tenhamos as seguintes condições respeitadas:

1. $C_1 \cup C_2 \cup ... \cup C_k = X$;
2. $C_i \neq \varnothing, \forall i, 1 \leq i \leq k$; e
3. $C_i \cap C_j = \varnothing, \forall i \neq j, 1 \leq i \leq k, 1 \leq j \leq k$.

Logo essas condições enfatizam que um objeto não pode pertencer a mais de um cluster (grupos disjuntos) e que cada cluster tem que ter ao menos um objeto. COLE (1998) *apud* [17] ainda acrescenta que o valor de k geralmente é desconhecido, e caso k seja conhecido, o problema é referido como o problema de k-Clusterização.

Identificar corretamente o número de clusters é um dos principais problemas que continua atrair muita atenção, pois conforme apresentado por [19], muitos algoritmos atuais exigem isso como um parâmetro especificado pelo usuário, o que é difícil de decidir sem um conhecimento prévio. Uma estimativa incorreta do número de clusters impedirá que os investigadores aprendam a real estrutura de agrupamento, por exemplo, para aplicações biomédicas, foco da pesquisa de [19], a subestimação pode fazer com que os subtipos de câncer sejam perdidos. Assim, se essa informação não for conhecida ou confiável a priori, o número de clusters deve ser determinado pelo próprio algoritmo com base nas propriedades dos dados.

Conforme apresentado por [17], o agrupamento pode ser empregado quando objetivo é reduzir o número de objetos, para um número de subgrupos característicos, levando as observações a serem consideradas como membros de um grupo e perfiladas segundo características gerais que rotulam distintamente este grupo, ou também quando o pesquisador deseja formular hipóteses sobre a natureza dos dados ou examinar hipóteses pré-estabelecidas.

Para ilustrar uma tarefa de agrupamento, considere o problema de segmentar uma base de dados descrevendo frutas, na qual cada fruta está descrita por um conjunto de atributos, como forma, cor e textura. Suponha que haja maçãs e bananas nessa base de dados e que o algoritmo precisa segmentá-los sem ter conhecimento algum sobre a classe da fruta, recebendo apenas informações dos atributos. Como a forma, cor e textura das

bananas são substancialmente diferentes da forma, cor e textura das maçãs, durante o agrupamento o algoritmo deverá, naturalmente, colocar bananas em um grupo e maçãs em outro [13].

Em [17], lista-se alguns outros exemplos de aplicações como às relativas a de reconhecimento de padrões, análise de dados, processamento de imagens, pesquisa de mercado, padrão de compra, especificações físicas e químicas de petróleos, dentre muitas outras, na qual as aplicações de clusterização são utilizadas para cumprir pelo menos um dos seguintes objetivos principais:

- *Identificação da estrutura subjacente:* para obter *'insights'* sobre os dados, gerar hipóteses, detectar anomalias, e identificar características marcantes.
- *Classificação Natural:* identificar o grau de semelhança entre as formas ou organismos (filogenética).
- *Compressão:* como um método para a organização dos dados e resumindo-o através de protótipos do cluster.

Para aplicação inicial aos dados de entrada do problema proposto por esta dissertação, o objetivo é a compressão dos dados, assim como identificação das estruturas subjacentes, para preparação dos dados para aplicação do próximo algoritmo a ser apresentado, o *FPGrowth*. Maiores detalhes serão explicados no subtópico 3.4.1.

3.3 Regras de Associação

A análise por associação, também conhecida por mineração de regras de associação, corresponde à descoberta de regras de associação que apresentam valores de atributos que ocorrem concomitantemente em uma base de dados. Há dois aspectos centrais na mineração de regras de associação: a proposição ou construção eficiente das regras de associação e a quantificação da significância das regras propostas [13].

Conforme apresentado por [1], em mineração de dados e aprendizado de máquina de IA, existem alguns algoritmos que permitem classificar de forma rápida e fácil, dados não categorizados com vários tipos de estruturas, em um conjunto de transações, realizando aprendizado de regras de associação (ARL - *Association Rules Learning*). Dentre esses

algoritmos, inclui-se um famoso algoritmo de "força bruta" (ou "busca exaustiva"), que gera todas as combinações possíveis de itens, para que na sequência identifique-se as possíveis regras, e a partir dele, uma série de modificações otimizadas formuladas sobre o mesmo. O algoritmo pioneiro é denominado Apriori, e como suas variações, temos como exemplo o ECLAT e o *FPGrowth*. Estes algoritmos pertencem a uma família inteira de aprendizagem de regras de associação (ARL).

A mineração de regras de associação, conforme descrito por [13], é uma técnica usada na construção de relações sob a forma de regras entre itens de uma base de dados transacional. Diferentemente do agrupamento, que busca relações de similaridade entre objetos, as regras de associação buscam relações entre os atributos dos objetos, ou seja, os itens que compõem a base. O objetivo é encontrar regras fortes de acordo com alguma medida do grau de interesse da regra. Os algoritmos operam com duas fases centrais, sendo a primeira geração dos conjuntos de itens frequentes e a segunda construção das regras a partir daqueles conjuntos que satisfazem critérios explícitos de avaliação de qualidade, podando regras pouco interessantes.

Conforme destacado por [1], a aprendizagem de regras de associação (ARL) é uma das estratégias mais essenciais do processo de mineração de dados, visto que se torna muito útil e eficiente sempre que precisamos processar grandes quantidades de dados representados em um formato de *string* (texto) e nenhuma outra informação é fornecida. Outro benefício de envolver a abordagem de mineração de regras de associação é seu bom desempenho em realizar um processamento otimizado de fluxos de dados dinâmicos normalmente enormes, obtendo percepções, tendências e conhecimento sobre os dados sendo analisados quase em tempo real.

A partir dos dois passos centrais para o processo de mineração das regras de associação, que geralmente envolvem a geração dos conjuntos de itens frequentes e extração das regras, seguem alguns conceitos básicos necessários para entendimento dos algoritmos:

- *Conjunto de itens:* sequência de itens possíveis, valores *string*, que representem todos os itens específicos;
- *Transações:* toda transação é um conjunto de itens de transações confirmadas e/ou processadas por algum sistema (dados de entrada), na qual certa quantidade de itens possíveis é associada.
- *Identificador da transação:* cada transação está associada à um identificador, denominado TID. Veja o exemplo da Tabela 3.1.

- *Conjunto de transações:* são os dados de entrada, fragmento do fluxo de dados ou conjunto de todas as transações confirmadas. Esses conjuntos de transações podem conter similaridade nos conjuntos de itens de outras transições. Veja o exemplo da Tabela 3.1.

- *Suporte mínimo:* Contagem do número de transações necessárias para que o conjunto de itens satisfaça o suporte mínimo (*minsup*), sendo este estipulado durante a implementação.

- *Conjunto de itens frequentes:* conjunto formado por itens com suporte maior que o mínimo (*minsup*). Cada elemento desse conjunto possui um contador de suporte que satisfaz o suporte mínimo, ou seja, a frequência de ocorrência do item dentre os conjuntos de itens das transações é maior ou igual *minsup*;

  – Como pode ser observado na Tabela 3.1, há itens que suas associações ocorrem com mais ou menos frequências, ou ainda, pelo menos um item que pode ocorrer na maioria das transações do conjunto.

TID	Transações
1	Engine, Vehicle, Steering, CPU, Brakes, Motherboard, Machines, Appliances
2	CPU, Wheels, Tank, Motherboard, Seats, Camera, Cooler, Computer, Machines, Appliances
3	Motherboard, Steering, Monitor, Brakes, Engine, Vehicle, Machines, Appliances
4	CPU, Steering, Brakes, Cooler, Monitor, GPU, Computer, Machines, Appliances
5	Vehicle, Cooler, CPU, Steering, Brakes, Wheels, Tank, Engine, Machines, Appliances
6	Computer, CPU, Motherboard, GPU, Cooler, Engine, Machines, Appliances
7	Brakes, Vehicle, Tank, Engine, CPU, Steering, Machines, Appliances
8	Pens, Ink, Pencils, Desk, Typewriter, Brushes, Office, Appliances
9	Ink, Typewriter, Pencils, Desk, Office, Appliances

Tabela 3.1: Exemplo de conjunto de transações. Fonte: [1]

Para a problemática abordada neste trabalho não se descartar dados com frequencias baixas, diferentemente de vários exemplos de aplicações de regras de associação, como por exemplo, dados de transações de caixas de mercados ou de vendas/tendências de compras dos clientes. Nestas aplicações, normalmente, há um grande número de transações coletadas e, neste sentido, transações pouco frequentes podem ser tratadas como pontos fora do padrão. No problema abordado isso não ocorre devido o banco de dados de entrada de um projeto que trata a definição do novo produto, todos os dados devem ser levados

em conta, pois são os dados de entrada necessários para chegarmos à resolução desta problemática. Neste caso, para a aplicação a ser detalhada mais a frente, o suporte *minsup* sempre será igual a 1, levando em consideração sempre, 100% dos dados de entrada.

Conforme apresentado por [1], para ser capaz de realizar o aprendizado de regras de associação (ARL), todo o conjunto de dados da transação deve estar em conformidade com pelo menos três requisitos, são eles:

1. Um conjunto de itens de cada transação deve conter em uma certa quantidade de itens pertencentes a uma "classe" específica, juntamente com outra quantidade menor de itens de uma ou múltiplas classes, diferentes da classe específica;
2. Na totalidade de itens de cada conjunto deve conter aqueles itens que representem uma "classe" ou "categoria" específica;
3. Deve haver um item que representa a classe "enraizada" superior, incluída na maioria dos conjuntos de itens de transação.

Para solução do problema proposto desta dissertação, como já antecipado na introdução deste capítulo, foi utilizado a variante optimizada do algoritmo pioneiro *Apriori*, denominado *FPGrowth*. Ambas as metodologias serão apresentadas nas próximas subseções, abordando as características que justicam a escolha pelo *FPGrowth*.

3.3.1 Algoritmo Apriori

Conforme apresentado por [13], os trabalhos de Agrawal, e Agrawal e Srikant foram pioneiros na proposição de um algoritmo de mineração de regras de associação, denominado Apriori, cujo objetivo era descobrir associações de produtos em grandes bases de dados transacionais obtidas em supermercados.

O algoritmo Apriori é o método mais conhecido para a mineração de regras de associação e emprega busca em profundidade e gera conjuntos de itens candidatos de k elementos a partir de conjuntos de itens com $k-1$ elementos [13]. Os itens candidatos não frequentes são eliminados, no teste dos *itemset*, a fim de filtrar aqueles que são de interesse a partir da verificação daqueles que atendem à frequência mínima pré-estabelecida (*minsup*) [20], veja o esquema de funcionamento na Figura 3.3. Toda a base de dados é rastreada e os conjuntos de itens frequentes obtidos a partir dos conjuntos de itens candidatos [13].

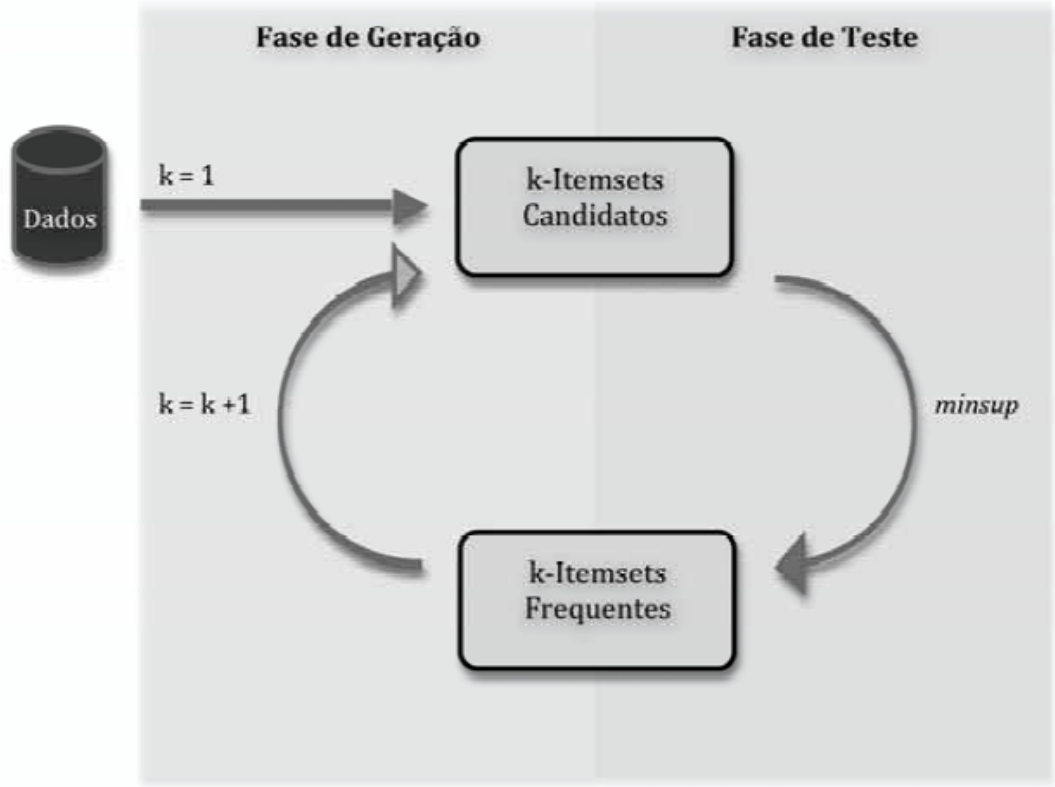

Figura 3.3: Esquema de funcionamento do algoritmo Apriori. Fonte: [20]

O procedimento Apriori formulado por Agrawal e Srikant, conforme apresentado por [1], basicamente realiza sempre uma busca completa em todos os conjuntos de itens, gerando novas regras e computando os valores de suporte e confiança apenas para aqueles itens candidatos recuperados, para os quais os valores das métricas de suporte e confiança excedem o limite mínimo. Especificamente, se dois candidatos a regra têm um valor suficiente de medida de suporte e confiança, uma nova regra produzida tem exatamente o mesmo valor dessas métricas ou mais. A abordagem a seguir permite reduzir drasticamente a quantidade de cálculos e memória do sistema consumidos e no final do processo de mineração de regras, basicamente acabaremos com um conjunto denominado "truncado" (no "limite") para novas regras sendo gerado, e que não precisará mais gerar variantes possíveis de candidatos a regra. O pseudocódigo apresentado por [1] fornece uma compreensão geral do algoritmo Apriori:

$$I = \{\forall i_n | \forall i_n \in t_s, t_s \in T\}$$

$$R_1 \leftarrow I, k \leftarrow 2$$

$$\text{enquanto} \quad R_{k-1} \neq \varnothing \quad \text{faça}$$

$$R_k \leftarrow \{r_i \cup \{r_j\} | (r_i, r_j) \in R_k, r_i \notin r_j\}$$

$$\text{para cada transação} \quad t_s \in T \quad \text{faça}$$

$$R_t \leftarrow \{r_z | r_z \in R_k, r_z \subseteq t_s\}$$

$$R_k \leftarrow \{r_s | r_s \in R_t, conf(r_s) > min\_conf\}$$

$$k \leftarrow k + 1$$

$$retorne \bigcup_k R_k$$

Onde I é o conjunto de todos possíveis itens, únicos e exclusivos pertencente aos grupos (T) de itens das transições (t) e k o tamanho de cada grupo de itens destas transições para cada interação, sendo R_k o conjunto de candidatos a regra de tamanho k sendo produzidos durante a iteração k, R_t o subconjunto de novos candidatos à regra sendo gerado satisfazendo a condição especificada ($\forall r_z \subseteq t_s$), r_z o grupo de k itens de R_k contidos na transação t_s e r_s o grupo de k itens de R_t que satisfazem a condição específica ($conf(\forall r_z) > min\_conf$).

A mineração de regras de associação pelo uso do algoritmo Apriori pode apresentar bons resultados, mas quando o tamanho do banco de dados de entrada aumenta, o desempenho cai [21]. A imagem abaixo, apresentado por [13], ilustra o pseudocódigo descrito por [1].

Pelo fato do algoritmo Apriori utilizar-se de cálculos para geração de possíveis candidatos, e não partir diretamente para implementação a partir de pequenos padrões, ou seja, pequenas regras prévias estabelecidas já pelos dados de entradas, no caso da problemática apresentada ele não apresenta uma resolução adequada pois necessita-se da consideração de 100% dos dados, pois nenhum dado é irrelevante. Sendo assim, na próxima subsessão será apresentado o algoritmo *FPGrowth*, na qual traz uma solução mais próxima do que é necessário para a resolução.

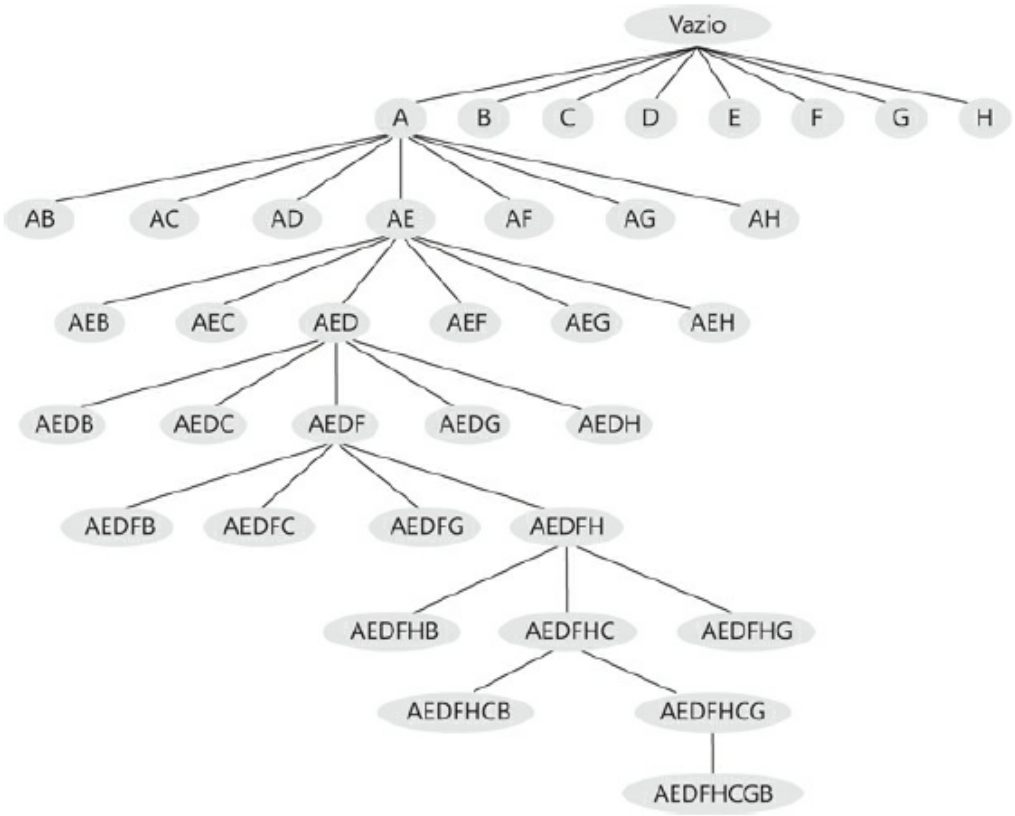

Figura 3.4: Ilustração do algoritmo Apriori. Fonte: [13]

3.3.2 Algoritmo FPGrowth

De modo geral, o algoritmo PFGrowth (*Frequent Patterns Growth* - Crescimento de Padrões Frequentes), conforme apresentado por [2], em vez de realizar a travessia de largura em primeiro lugar para produzir todas as variantes possíveis de candidatos a regras de associação, conforme apresentado anteriormente pelo Apriori, o seguinte algoritmo permite construir uma estrutura de dados compactada chamada FP-Tree (*Frequent Pattern Tree* - Árvore de Padrão Frequente) e extrair as regras de associação mais interessantes e significativas diretamente da árvore, tornando todo o processo de mineração de regras um pouco mais inteligente. Todo o processo de mineração de regras de associação formulado com o algoritmo *FPGrowth* é basicamente realizado em duas fases principais.

De acordo com Han, Pei e Yin *apud* [13], as dificuldades anteriormente apresentadas pelo algoritmo Apriori, sendo elas, dificuldade para tratar uma grande quantidade de conjuntos candidatos e execução de repetidas passagens pela base de dados, se devem, principalmente, ao método de geração e teste dos conjuntos candidatos a itens frequentes.

Sendo assim, os autores propuseram esse novo algoritmo *FPGrowth*, baseado na estrutura em árvore de prefixos para padrões frequentes, o qual armazena de forma comprimida todos os padrões frequentes.

Similar ao Apriori, no pré-processamento dos dados de cada transação no FP-Growth, há uma verificação de todos itens e determinação de suas frequências. Sendo assim, todos itens não frequentes - que são, todos os itens que aparecem em poucas transações e consequentemente sua frequência é inferior ao valor mínimo especificado pelo usuário - são descartados das transações [22].

A FP-Tree, árvore de itens frequentes, conforme apresentada por [13] e [23], é uma estrutura em árvore que possui um nó raiz nulo, com um conjunto de subárvores de itens prefixos como filhos da raiz (*children*) e uma tabela de cabeçalho dos itens frequentes. Cada nó na subárvore de itens possui quatro campos: nome_do_item, contagem (número de transações representadas pela porção do caminho que chega àquele nó), ponteiro_ao_itemOrigem (nó do item de origem ou a raiz, caso não haja outro nó) e ponteiro_conj_nós_filhos (conectando-o aos próximos nós da árvore, caso estes não sejam folhas da árvoré). Por fim, na tabela de cabeçalho de itens frequentes, cada valor possui dois campos: nome_do_item e cabeçalho da ligação_do_nó, o qual aponta para o primeiro nó na FP-Tree que possui o mesmo nome_do_item.

Para ter a capacidade de realizar mineração de regras de associação nos dados usando o algoritmo *FPGrowth*, [2] ressalta que primeiro devemos transformar um fluxo de dados brutos em um banco de dados canônico, ou seja, organizada e/ou estruturada da seguinte maneira:

Supondo-se que os dados de entrada seja um conjunto de transações $T = t_1, t_2, t_3, ..., t_{(n-1)}, t_n$, obtido de um fluxo de dados bruto como o da Tabela 3.1 e um conjunto de classes predefinidas $C = c_1, c_2, c_3, ..., c_{(m-1)}, c_m$, como as listadas pela Tabela 3.2. Todo o conjunto de transações deve ser convertido em um banco de dados canônico, possuindo o formato apresentado pela Tabela 3.3.

ID	Classes
1	Appliances
2	Machines
3	Vehicle
4	Computer
5	Office

Tabela 3.2: Exemplo de conjunto de classes proposto. Fonte: [2]

TID	Transações
1	Appliances, Machines, Vehicle, CPU, Brakes, Engine, Motherboard, Steering
2	Appliances, Machines, Computer, CPU, Motherboard, Cooler, GPU, Tank, Wheels, Camera, Seats
3	Appliances, Machines, Vehicle, Brakes, Engine, Motherboard, Steering, Monitor
4	Appliances, Machines, Computer, CPU, Brakes, Motherboard, Steering, Cooler, GPU, Monitor
5	Appliances, Machines, Vehicle, CPU, Brakes, Engine, Steering, Cooler, Tank, Wheels
6	Appliances, Machines, Computer, CPU, Engine, Motherboard, Cooler, GPU
7	Appliances, Machines, Vehicle, CPU, Brakes, Engine, Steering, Tank
8	Appliances, Office, Desk, Ink, Pencils, Typewriter, Brushes, Pens
9	Appliances, Office, Desk, Ink, Pencils, Typewriter

Tabela 3.3: Exemplo de conjunto de transações. Fonte: [2]

Conforme pode ser observado na Tabela 3.3, num banco de dados canônico os itens específicos no conjunto de itens de cada transação são realmente classificados em ordem decrescente por sua frequência de ocorrência. Nesse caso, a frequência de ocorrência é estimada pelo número de vezes em que um determinado item aparece em cada transação, abaixo o passa-a-passo para a verificação dos itens nos conjuntos de transações apresentado por [2] que detalha bem esse processo:

a. Identificar o conjunto de itens possíveis que são encontrados pela análise de cada conjunto de itens nas transações $I = \{\forall i_k | i_k \in t_s, t_s \in T\}$;

b. Calcular a frequência de ocorrência de cada um dos itens $(\forall i_k)$ do conjunto de itens $I = \{(\forall i_k, contagem(\forall i_k) | i_k \in t_s, t_s \in T\}$;

c. Classificar todos os itens $\forall i_k \in I$ por sua frequência de ocorrência em ordem decrescente;

d. Para cada "classe" $\forall c_j \in C$, encontrar todos os itens em I, representando a "classe" e adicionar ao conjunto de itens $I_c = \{(\forall i_k, contagem(\forall i_k)) | i_k = c_j, c_j \in C\}$;

e. Remover todos os itens em I, representando uma "classe" $\forall i_k \in I_c \in I$;

f. Classificar todos os itens que representam uma "classe" $\forall i_k \in I_c$ pela frequência de ocorrência em ordem decrescente;

g. Anexar o conjunto de itens I_c, contendo classes, ao topo do conjunto de itens $I (I \leftarrow I_c)$;

h. Para cada conjunto de itens $t_s \in T$ da transação, organizar todos os itens pela ordem decrescente de frequência em I.

Para [13], na determinação da lista de itens frequentes é considerado um dado de entrada *minsup*, na qual faz-se a leitura de dados para listagem dos itens frequentes e ordenação, desconsiderando os itens que não contabilizam a frequência que atenda o *minsup*. Tendo todas as transações organizadas conforme as informações de frequência de todos os itens dos dados de entrada, o próximo passo de fato é a construção da FP-Tree para extração das regras de associações direto desta árvore.

Para a criação da FP-Tree, primeiramente, inicia-se a árvore com a criação da raiz representada por um nó nulo; Recupera-se cada item da primeira transação para criação de um nó especifico para cada um destes itens e, a partir da raiz, acrescenta-se cada um deles, realizando-se referência do nó de item anterior (ou seja, "pai") aos nós de item subsequentes (ou seja, "adjacente"), conforme pode ser observado na Figura 3.5. Além disso, conforme já relatado anteriormente, cada novo nó adicionado à árvore terá um valor específico de contagem suporte, que começará inicialmente com seu valor inicial igual a 1. Veja na Figura 3.6 o fluxograma ilustrativo desta etapa.

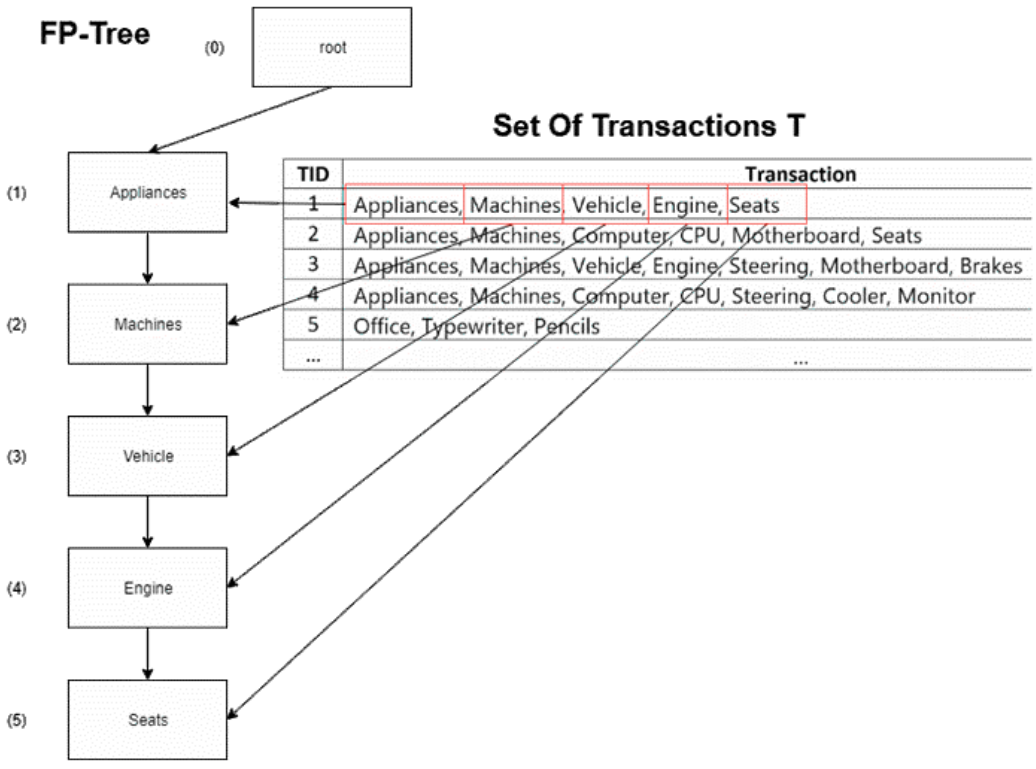

Figura 3.5: Construção da FP-Tree. Fonte: [2]

Tendo toda a primeira transação acrescentada, a partir da seguinte já é verificado se o item da transação atual já foi acrescentado previamente na árvore, pois caso seja encontrado o mesmo item, é realizado o incremento da contagem suporte apenas; e em

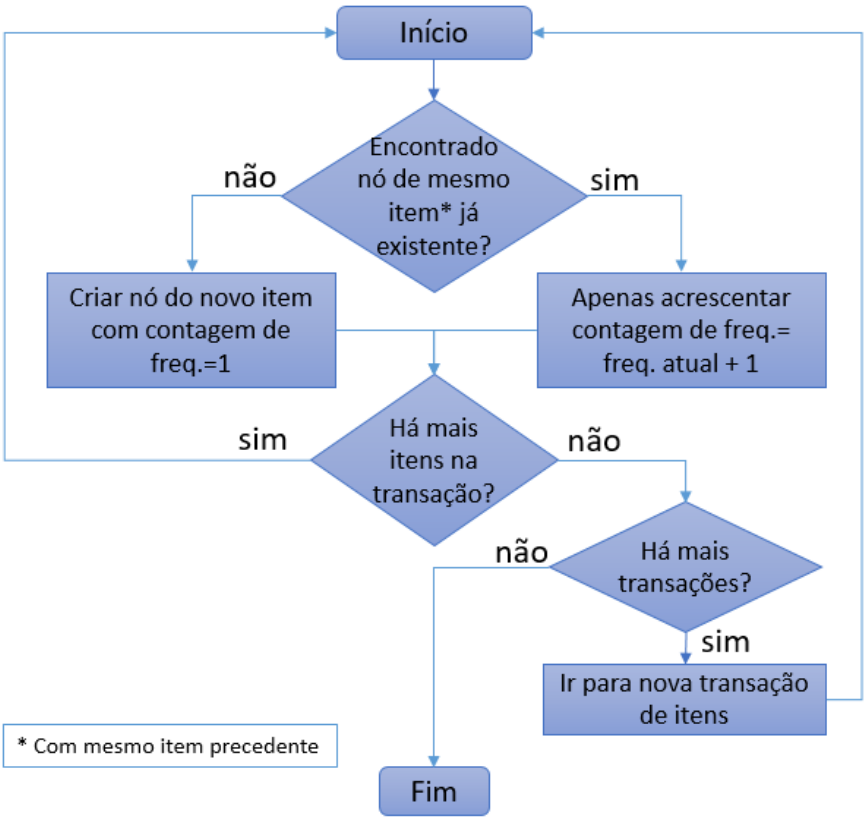

Figura 3.6: Fluxograma de construção da FP-Tree.

caso, de identificação de novos itens, a partir de seu nó de origem ("pai"), cria-se o novo respectivo nó à árvore, conforme pode ser observado no exemplo da Figura 3.7 e Tabela 3.4.

ID	Item	Pai	Adjacente	Contagem Suporte
1	root	0	{2}	0
2	Appliances	1	{3}	2
3	Machines	2	{4,5}	2
...	...	...	...	...

Tabela 3.4: Exemplo um fluxo de dados brutos. Fonte: [2]

Realizado todo esse processo por todas as transações, ao final da construção da FP-Tree, os conjuntos de dados obtidos como resultado da realização do percurso, das folhas à raiz, ou seja, "de baixo para cima" na árvore, poderia já ser o resultado parcial esperado. Conforme pode ser visto no pequeno exemplo ilustrativo da Figura 3.8 e Tabela 3.5, onde o total de folhas já poderia contabilizar o total de variantes, e o percurso de todos os

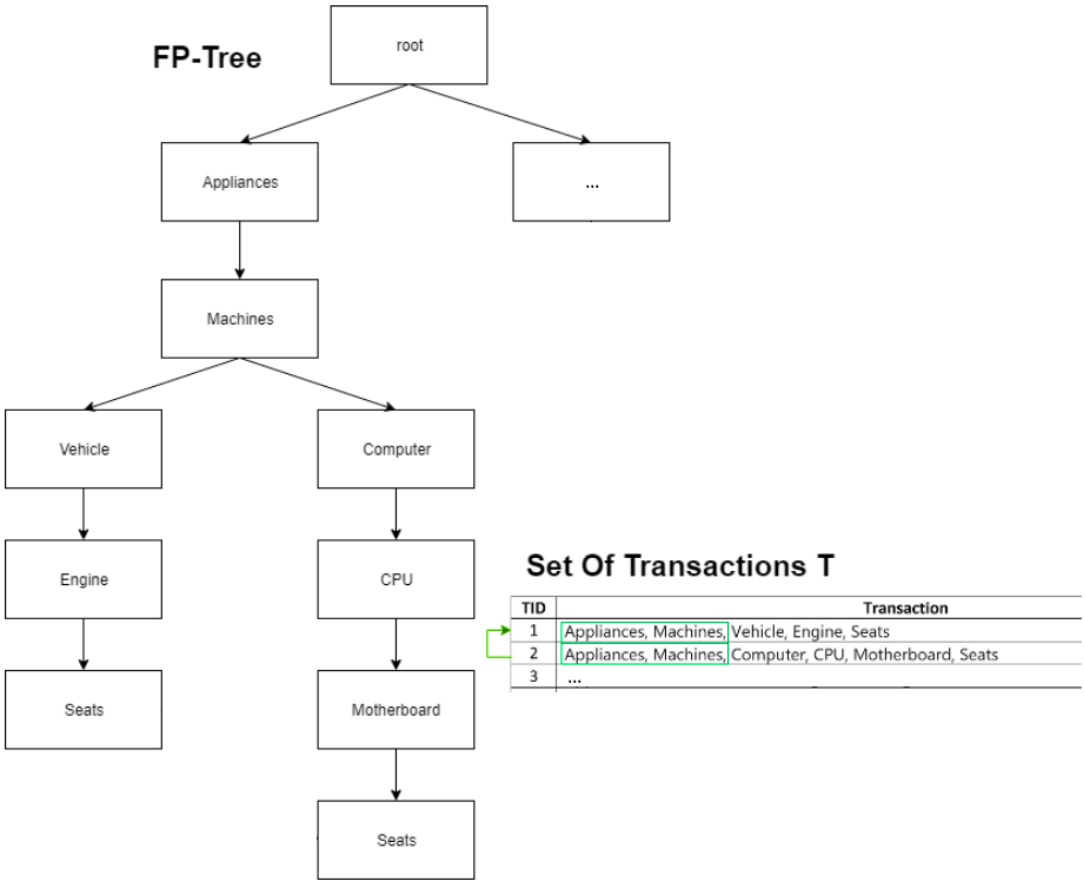

Figura 3.7: Exemplo de FP-Tree após tratativas de 2 transações. Fonte: adaptado de [2].

conjuntos de módulos amarrados a ela.

ID	Classes
Variante 1	Nivel 1, Motor 1, Câmbio 1,...
Variante 2	Nivel 1, Motor 2, Câmbio 1,...
Variante 3	Nivel 2, Motor 3, Câmbio 1,...
Variante 4	Nivel 2, Motor 3, Câmbio 2,...
Variante 5	Nivel 2, Motor 4, Câmbio 2,...

Tabela 3.5: Exemplo variantes gerados pela leitura de uma suposta FP-Tree.

Conforme Tabela 3.5, o exemplo apresentado a partir da representação de uma inicial construção de FPTree, chega-se à necessidade de 5 variantes ou diversidades de chicote borda de linha para atender os *features* inicialmente destacados.

Conforme apresentado por [24] e [25], algoritmo Apriori utiliza-se das gerações de candidatos, propriedade apriori e une, propriedades puras para mineração de padrões frequentes e o *FPGrowth* constrói um padrão condicional livre e uma base de padrão

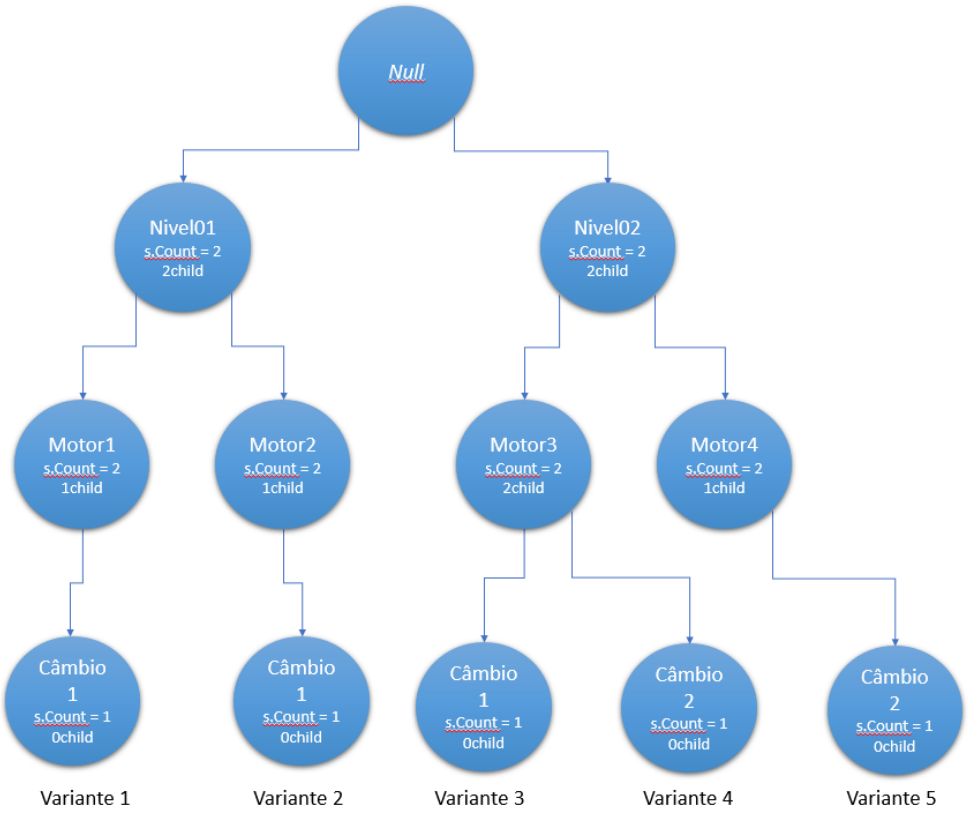

Figura 3.8: Exemplo de construção de uma FP-Tree com dados sintéticos de veículo.

condicional a partir do banco de dados que satisfaça o suporte mínimo. Sendo assim, como precisamos do padrão condicional gerada a partir dos próprios dados do projeto, o algoritmo *FPGrowth* é o que melhor atende a necessidade para resolução da proposta.

3.4 Pseudoprojeto Veículo

Para desenvolvimento do algoritmo e execução do código, foi elaborado uma tabela com pseudocódigos de *features*, correlacionando seus impactos por famílias de chicotes, que pode ser consultado no Apêndice A, caracterizando uma pseudoestrutura de arquitetura veicular para um possível projeto veículo, um dos dados necessários para desenvolvimento desta etapa. Desta mesma maneira, foi elaborado uma pseudoficha de definição de um projeto veículo que pode ser consultada no Apêndice B, representando este segundo dado de entrada necessário para realização desta etapa de identificação de variantes (ou diversidade) de chicotes do projeto.

ID	1	2	3	4	5
1	NVL01	MOT01	CMBGS	CXCMA	
2	NVL01	MOT02	CMBFF	CXCMA	
3	NVL02	MOT02	CMBFF	CXCAT	
4	NVL03	MOT03	CMBGS	CXCAT	
5	NVL03	MOT03	CMBFF	CXCAT	
6	NVL03	SPP01			
7	NVL02	CHA01			
8	NVL03	CHA01			
9	NVL01	RBG01			
10	NVL02	RBG01			
11	NVL03	RBG02			
12	NVL01	SCS01			
13	NVL02	SCS02			
14	NVL03	SCS02			
15	NVL03	SBA01			
16	NVL01	MOT01	CMBGS	CXCMA	OFR01
17	NVL03	TAL02			
18	NVL01	TAL01			
19	NVL02	TAL01			
20	NVL02	TRM01			
21	NVL03	TRM01			
22	NVL03	SES01			
23	NVL01	TMR00			
24	NVL02	TMR01			
25	NVL03	TMR02			
26	NVL02	MIC01			
27	NVL03	MIC01			
28	NVL01	DAF01			
29	NVL02	DAF02			
30	NVL02	TAN01			
31	NVL03	TAN02			
32	NVL01	CAC01			
33	NVL02	CAC02			
34	NVL03	CAC02			
35	NVL01	TIG01	FFH00		
36	NVL02	TIG01	FFH00		
37	NVL03	TIG02	FFH01		
38	NVL01	TFA01			
39	NVL02	TFA02			
40	NVL03	TFA03			
41	NVL02	FNB01			
42	NVL03	FNB02			
43	NVL01	TLC01			
44	NVL02	TLC01			
45	NVL03	TLC02			

Tabela 3.6: Dados de entrada gerados a partir de dados sintéticos de projeto veículo.

O engenheiro tendo disponível todos os dados de entrada do projeto, visto que ainda não há um sistema integrado para disponibilização das informações do novo projeto, apenas planilhas de informações que não há uma padronização fixa para os dados, uma das premissas levadas em consideração para desenvolvimento do algoritmo, é de que o próprio engenheiro terá que fazer os *inputs* necessários ao iniciar o algoritmo. Como a ferramenta utilizada para elaboração da problemática foi o Microsoft Excel®, ferramenta familiarizada pela maioria dos profissionais e disponível na maioria das empresas, no qual dispõe da ferramenta de desenvolvimento Microsoft *Visual Basic for Applications* (VBA), ambiente utilizado para implementação do algoritmo, foi proposto formulários de preenchimentos iniciais assim que o arquivo é aberto antes de rodar o algoritmo, as figuras dos mesmos podem ser consultadas no Apêndice E. Logo, todo *input* do engenheiro será feito através da interface de caixas de diálogo (*InputBox*), sendo essa interação o momento da geração dos dados brutos, a partir dos dados de entrada apresentados inicialmente pelos apêndices A e B, na qual origina-se os dados sintéticos de entradas conforme apresentado na Tabela 3.6.

A partir dos dados de entrada apresentados na Tabela 3.6, o algoritmo irá iniciar sua leitura primeiramente fazendo a varredura total nos dados obtidos para identificar o tamanho dos dados de entrada, que podem variar de acordo com a complexidade de cada projeto, e a partir desta análise inical, agregar todos os dados de entrada em uma matriz para início da mineração dos dados.

No caso apresentado, de acordo com a Tabela 3.6 seria uma matriz de 45 linhas e 5 colunas. Lembrando que, são dados sintéticos estes dados de entrada, normalmente essa baixa quantidade de dados de entrada não definem um veículo completo, é apenas didático e com condições suficientes para testarmos todas as funcionalidades necessárias para o algoritmo proposto. Nesta sessão não vamos detalhar como foi realizado a criação das caixas de diálogo (*InputBox*) do Excel®, pois vamos enfatizar a solução do algoritmo proposto e não a ferramenta utilizada, mas toda implementação poderá ser consultada no Apêndice C.

O Fluxograma da Figura 3.9, descreve bem toda esta aplicação da metologia aplicada para solução da problemática. Nesta seção é descrito todo o processo envolvido na etapa 1, sendo a sessão 3.4.1 seguinte detalhando a etapa 2, seguido da sessão 3.4.2 descrevendo as etapas 3 e 4, por fim a sessão 3.4.3 descrevendo a etapa 5 que consequentemente irá finalizar para chegar-se ao resultado (item 6).

Figura 3.9: Fluxograma da implementação da metodologia no Pseudoprojeto.

3.4.1 Tratamento dos dados de entrada e Agrupamento

Após o algoritmo obter todos os dados de entrada, o primeiro passo é checar todas as premissas inerentes aos dados e adaptar qualquer informação necessária nestes dados existentes e realizar o agrupamento dos dados de entrada, que necessitem de tratativas particulares sem poluir os *inputs* que farão parte do algoritmo *FPGrowth*. São necessários conferir 4 aspectos iniciais em cada transação dos dados de entrada, e realizar suas respectivas tratativas apartir do agrupamento, para isso, utilizou-se de 4 diferentes vetores, sendo eles para:

- Realizar o agrupamento de todas as transações que possuem apenas 2 atributos de *features* associados;
  - Costuma ocorrer devido a associação de alguns *features* ser diretamente ligado ao nível veículo apenas. Para isso, o algoritmo checa sempre se a terceira posição de cada transação (linhas) de dados está vazia, que é o primeiro critério para separar os dados que serão agrupados e adiciona os mesmos à um vetor auxiliar (*ArrayAux1*).
- Realizar o agrupamento de todas as transações que não possuem atributos correlacionados a atributo de motorização (*MOTxx*);
  - Costuma ocorrer devido a associação de alguns *features* ser diretamente ligado ao nível veículo, mas também por consequência, carrega outros *features* apartir

da sua existência, e não apenas um (caso anterior), exemplo: o *feature* da função *hands-free*, de acesso remoto, está condicionado a ser também ignição por botão para dar a partida no motor e aplicação da chave remota (sem plip). Mas de fato, o que direciona sua aplicação principal ao veículo, é o seu atributo de nível. (*ArrayAux3*).

- Realizar o agrupamento de todas as transações de casos especiais, sendo aqueles que não iniciam com correlação à atributos de nível veículo e sejam com mais de dois *features* associados;

  – Estes casos podem ocorrer, por exemplo, devido à alguma especificidade do processo fornecedor de chicotes, ou ainda alguma versão especial de veículo, na qual faz-se necessários módulos específicos de circuitos associados à uma combinação de caracrísticas específicas, e não à uma definição usual de projetos.(*ArrayAux2*).

- Por fim, é realizado também o agrupamento temporário de todas as transações que possua atributo de motorização (*MOTxx*) mas que não estão amarrados todas as possibilidades de atributos de câmbio(*CMBxx*) e combustível(*CMBxx*) para cada um deles.

  – Costuma ocorrer devido a associação de algumas características ser diretamente ao motor e aplicável a todas suas variantes, seja câmbio manual ou automático por exemplo, sendo assim, inicialmente quando o algoritmo inicia-se, os primeiros dados de características solicitados para cadastro e armazenamento, são os de variantes de motores, para realização dessa conferência de acordo com cada dado de entrada dos projetos.

  – Esses dados serão armazenados em um vetor provisóriamente e devolvidos aos dados de entradas com os valores completados corretamente, de acordo com os dados de entrada de possíveis variantes de cada motor. (*ArrayAux4*).

Na Tabela 3.6 é possível verificar os exemplos de *features* diretamente ligados ao nível veículo, utilizando-se as linhas 9 à 14 como referência. Verificando-se os códigos no Apêndice B, pode-se observar que estas linhas associam os códigos de níveis da coluna 1 “NVLxx”, aos códigos de Airbag “RBGxx” e sensor do cinto de segurança “SCSxx”, na qual podemos ler que, para esse pseudo-projeto, no nível 1, tem-se apenas os airbags

frontais regulamentares e sensor de cinto de segurança apenas para o motorista; no nível 2, tem-se apenas os airbags frontais regulamentares e mas já tem sensor de cinto de segurança tanto para o motorista quanto para o passageiro frontal; e no nível 3, tem-se os airbags regulamentares e também os laterais, assim como, os sensores de cinto de segurança tanto para o motorista quanto para o passageiro frontal. Nenhum destes, como pode ser observado, tem associação com qualquer outro *feature*, como por exemplo, tipo de motorização ou combustíveis associados.

Identificado todos os itens para agrupamento, iniciando pelo vetor de transações de itens de 2 atributos de *features* associados apenas, já é realizado neste mesmo *looping* a análise dos principais atributos que se repetem em cada transação e sua contagem suporte, para identificação dos atributos mestres, ou atributos de classe, e que formam a quantidade dos agrupamentos (ou *Clusters*). Para esta etapa, utiliza-se mais dois vetores auxiliares, um vetor para identificação dos atributos de classe (*ArrayIDAux*) e um vetor para contagem suporte da frequencia do mesmo (*ArrayIDCoef2*).

Como a quantidade de transações a serem agrupadas são desconhecidas, o vetor auxiliar (*ArrayAux1*) é criado inicialmente com a mesma quantidade de linhas da matriz de entrada, pois a única premissa que sabemos, é de que será suficiente pois a quantidade de linhas a ser utitilizada para identificação prévia dos dados a serem agrupados será menor que a entrada, e o vetor de identificação dos atributos de classe é criado com um tamanho mínimo e redmensionado dentro do próprio *looping*, de acordo com a identificação dos dados.

Na Figura 3.10 e 3.13, um pseucódigo representando o trecho da identificação dos dados a serem agrupados. Todos os dados identificados e coletados para o processo de agrupamento são removidos da matriz de entrada de dados.

Analisando inicialmente os dados presentes no vetor auxiliar *ArrayAux1*, com todos os dados a serem agrupados identificados, o algoritmo consegue obter a quantidade de classes identificadas pela quantidade de dados estocados nos *ArrayIDs* e cria-se um novo vetor *ArrayCluster*, conforme a quantidade de classes, e mais uma coluna para estocar os dados que serão agrupados na classe identificada, conforme a varredura de todos os dados estocados no vetor auxiliar.

Para a varredura do *ArrayAux1* de acordo com as classes identificadas em cada trasação, agrupa-se todas informações numa variável *collection* que agrupa todas as entradas *string* identificadas numa única variável e ao final, registrasse no vetor *ArrayCluster* a

```
Considerando:
        [Matriz de textos] DadosEntrada (x,y) = dados de brutos de entrada
        [inteiro] NumLinhas = total do número de linhas(x) dos dados de entrada
Declaração de matrix ArrayAux1()
Declaração de vetores ArrayIDAux(); ArrayIDcoef2()
Declaração de variáveis n, n1, n2, aux, VarID
PARA n = 1 À NumLinhas
  SE DadosEntrada(n, 3) = vazio ENTÃO
    ArrayAux1(n1, 1) RECEBE DadosEntrada(n, 1)
        DadosEntrada(n, 1) DELETADO
    ArrayAux1(n1, 2) RECEBE DadosEntrada(n, 2)
        DadosEntrada(n, 2) DELETADO
    SE ArrayIDAux(1) = vazio ENTÃO
      ArrayIDAux(n2) RECEBE ArrayAux1(n1, 1)
      ArrayIDcoef2(n2) RECEBE 1
      n2 INCREMENTA + 1
    SENÃO
      aux RECEBE 1
      ENQUANTO ArrayIDAux(aux) ≠ vazio FAÇA
        SE ArrayIDAux(aux) = ArrayAux1(n1, 1) ENTÃO
          ArrayIDcoef2(aux) INCREMENTA + 1
          VarID RECEBE "Encontrado"
          FINALIZAR ENQUANTO
        SENÃO
        aux INCREMENTA + 1
        VarID RECEBE "NãoEncontrado"
        FIM SE
      CICLO ENQUANTO
      SE VarID = "NãoEncontrado" ENTÃO
      ArrayIDAux(aux) RECEBE ArrayAux1(n1, 1)
      ArrayIDcoef2(aux) RECEBE 1
      n2 INCREMENTA + 1
      REDIMENSIONE ArrayIDAux PRESERVANDO VALORES CONFORME TAMANHO (n2)
      REDIMENSIONE ArrayIDcoef2 PRESERVANDO VALORES CONFORME TAMANHO (n2)
      FIM SE
    FIM SE
    n1 INCREMENTA + 1
  FIM SE
PRÓXIMO
```

Figura 3.10: Pseudocódigo - Identificação dos dados de entrada para o agrupamento 1

classe e na sequência o valor único de toda coleção (variável *collection*) de *string* dos itens das transações.

Veja na Figura 3.11 todo o processo de agrupamento descrito, e na sequencia, a eliminação do vetor *ArrayAux1*, obtendo todos os dados agrupados no vetor *ArrayCluster*, que será adicionado posteriormente ao final do resultado do algoritmo *FPGrowth*. Conforme pode ser observado pela janela de variáveis locais do VBA, na Figura 3.12, é possível identificar os valores de Airbag "RBGxx" e sensor do cinto de segurança "SCSxx", conforme exemplificado anteriormente.

Para os demais casos, Figura 3.13, todos os dados identificados pelo vetor auxiliar *ArrayAux3*, conforme realizado para o agrupamento do *ArrayAux1*, será analisado em cada uma das transações as classes existentes e será agrupado todas as entradas *string*

```
Considerando:
        [Matriz de textos] ArrayAux1 (x, y) = dados identificados para agrupamento
        [inteiro] NumCol = total do número de colunas(y) dos dados de ArrayAux1
        [inteiros] n = total do número de linhas(x) dos dados a serem agrupados; aux = 1; n1 = 1; n2 = 1; e
        [texto] VarID = vazio
Declaração de matrix ArrayCluster(n,2)

Sendo Cluster uma Classe definida como:
        Declaração de variável texto ValorID
        Declaração de variável coleção Agrupados

PARA n1 = 1 À n
  DECLARANDO x COMO UM NOVO Cluster
    x.ValorID RECEBE ArrayIDAux(n1)
    x.Agrupados DEFINIDO COMO UMA NOVA coleção
    ArrayCluster (n1, 1) RECEBE x.ValueID
  PARA n2 = 1 À NumCol
    SE ArrayAux1(n2, 1) = ArrayIDAux(n1) ENTÃO
        ADICIONE ArrayAux1(n2, 2) + "; " EM x.Agrupados
        aux RECEBE aux + 1
    FIM SE
  PRÓXIMO
  PARA CADA VarID PRESENTE EM x.ArrayCL
    out RECEBE out + VarID
  PRÓXIMO
  ArrayCluster (n1, 2) RECEBE out
  LIMPE VALORES DE out
PRÓXIMO
DELETE ArrayAux1
```

Figura 3.11: Pseudocódigo - Agrupamento

Expressão	Valor	Tipo
ArrayCluster		String(1 to 3, 1 to 2)
ArrayCluster(1)		String(1 to 2)
ArrayCluster(	"NVL03"	String
ArrayCluster(	"SPP01; CHA01; RBG02; SCS02; SBA01; TAL02; TRM01; SES01; TMR02; MIC01; TAN02; CAC02; TFA03; FNB02; TLC02"	String
ArrayCluster(2)		String(1 to 2)
ArrayCluster(	"NVL02"	String
ArrayCluster(	"CHA01; RBG01; SCS02; TAL01; TRM01; TMR01; MIC01; DAF02; TAN01; CAC02; TFA02; FNB01; TLC01"	String
ArrayCluster(3)		String(1 to 2)
ArrayCluster(	"NVL01"	String
ArrayCluster(	"RBG01; SCS01; TAL01; TMR00; DAF01; CAC01; TFA01; TLC01"	String

Figura 3.12: Valor de variáveis locais VBA - ArrayCluster

no registro do vetor *ArrayCluster* de acordo com cada atributo de classe, e na sequência, também ocorrerá a eliminação do vetor *ArrayAux3*. Para os dados obtidos no *ArrayAux4*, que foram trabalhados acrecentando dados faltantes, os mesmos serão devolvidos para os dados de entrada e farão parte dos dados de entrada para o algoritmo *FPGrowth*, e na sequência, o vetor *ArrayAux4* também poderá ser eliminado. Para os dados obtidos no *ArrayAux2*, oriundos dos casos especiais, estes serão os únicos que terão tratativas apenas no final do algoritmo.

```
Considerando:
        [Matriz de textos] DadosEntrada (x, y) = já ajustados, sem os dados do primeiro agrupamento
        [Matriz de textos] ArrayMOT (z, w) = dados de entrada das variáveis possíveis de motorização
        [inteiro] NumLinhas2 = total do número de linhas(x) dos dados de entrada
        [inteiro] NumColunas = total do número de colunas(y) dos dados de entrada
Declaração de matrix ArrayAux2(), ArrayAux3() e ArrayAux4()
Declaração de variáveis n; aux = 1; nct = 1; i = 1; k = 1; VARID
PARA n1 = 1 À NumLinhas2
 PARA n2 = 1 À NumColunas
  SE DadosEntrada (n1, n2) ≠ vazio ENTÃO
    VARID RECEBE DadosEntrada (n1, n2)
     SE VARID ≠ "FEATURE DE NVL" E n2 = 1 ENTÃO
        n = IDENTIFICADO CASOS ESPECIAIS
     SENÃO
        SE DadosEntrada (n1, n2 + x) = "SEQUÊNCIA COMPLETA DE FEATURES DE MOTOR" (Para x de 1 a 3 -- FEATURES DE
"MOT"; "CMB"; e "CXC") ENTÃO
          n = NADA PRECISA SER FEITO, INFORMAÇÃO COMPLETA
        SE DadosEntrada (n1, n2 + x) = "CONTÉM APENAS ATRIBUTO MOTOR" (Para x de 1 a 3 -- FEATURES DE "MOT" apenas,
faltando "CMB"; e "CXC")
          n = INFORMAÇÃO INCOMPLETA, AJUSTAR
        SENÃO
          n = MAIS DE 3 ATRIBUTOS E AMARRADOS APENAS A NÍVEL
     FIMSE
     SE n = IDENTIFICADO CASOS ESPECIAIS ENTÃO
        ArrayAux2(nct, n2) RECEBE DadosEntrada (n1, n2)
        DadosEntrada (n1, n2) DELETADO
     SENÃO SE n = INFORMAÇÃO INCOMPLETA, AJUSTAR ENTÃO
        ArrayAux4(aux, n2) RECEBE DadosEntrada (n1, n2)
        DadosEntrada (n1, n2) DELETADO
     SENÃO SE n = MAIS DE 3 ATRIBUTOS E AMARRADOS APENAS A NÍVEL ENTÃO
        ArrayAux3(i, n2) RECEBE DadosEntrada (n1, n2)
        DadosEntrada (n1, n2) DELETADO
     SENÃO
        n2 RECEBE NumColunas
     FIMSE
 PRÓXIMO
  SE n = INFORMAÇÃO INCOMPLETA, AJUSTAR ENTÃO
        CHECA ArrayMOT (z, w)
        ArrayAux4(aux, n2) ACRESCENTA INFORMAÇÕES FALTANTES CONFORME INFORMAÇÕES DE ArrayMOT (z, w)
  FIMSE
     SE n = IDENTIFICADO CASOS ESPECIAIS ENTÃO
        nct RECEBE nct +1
     SENÃO SE n = INFORMAÇÃO INCOMPLETA, AJUSTAR ENTÃO
        aux RECEBE aux+1
     SENÃO SE n = MAIS DE 3 ATRIBUTOS E AMARRADOS APENAS A NÍVEL ENTÃO
        i RECEBE i +1
     FIMSE
   n RECEBE 0
PRÓXIMO
```

Figura 3.13: Pseudocódigo - Identificação dos dados de entrada para o agrupamento 2

3.4.2 Adaptação do algoritmo FPGrowth

Após o agrupamento, vários vazios ficaram presentes no array de entrada, usando a Tabela 3.6 como referência, seriam por exemplo, trechos de linhas de 6 à 15. Logo, antes de iniciar-se a tratativa dos dados para aplicação do algoritmo *FPGrowth*, foi realizado o ajuste dos dados de entrada no *ArrayIN*, para reagrupamento e redução dos vazios. E Para identificar a nova quantidade de linhas que atualmente possuem valores de entrada, foi criado uma nova variável *NbLines*2 que chama uma nova sub-rotina *CheckArray*.

Para inicialização da primeira etapa do algoritmo *FPGrowth*, identificação de todos os itens e suas frequências, para reorganização dos dados de entrada e construção da FPTree, utiliza-se dois vetores auxiliares *ArrayID* e *ArrayIDcoef*, na qual ao fazer a varredura da matriz de dados *ArrayIN*, o *ArrayID* irá estocar cada um dos itens existentes e o *ArrayIDcoef* irá contabilizar a frequência de cada um deles, conforme pode ser observado pelo trecho da Figura 3.14.

```
PARA n1 = 1 À NbLines2
  PARA n2 = 1 À NbColun
    SE n = 1 ENTÃO
    ArrayID(n) RECEBE ArrayIN(n1, n2)
    ArrayIDcoef(n) RECEBE 1
    SENÃO
      aux RECEBE1
      FAÇA ENQUANTO  ArrayID(aux) ≠ VAZIO
        SE ArrayID(aux) = ArrayIN (n1, n2) ENTÃO
          ArrayIDcoef(aux) RECEBE ArrayIDcoef(aux) + 1
          VarID RECEBE "Found"
          SAIR DO FAÇA
        SENÃO
        aux RECEBE aux + 1
        VarID RECEBE "NotFound"
        FIMSE
      CICLO ENQUANTO
      SE VarID = "NotFound" ENTÃO
      ArrayID(aux) RECEBE ArrayIN (n1, n2)
      ArrayIDcoef(aux) RECEBE 1
      FIMSE
    FIMSE
    n RECEBE n + 1
  PRÓXIMO
PRÓXIMO
'
aux RECEBE aux - 1
'
REDIMENSIONE ArrayID () PRESERVANDO VALORES CONFORME TAMANHO aux
REDIMENSIONE ArrayIDcoef () PRESERVANDO VALORES CONFORME TAMANHO aux
```

Figura 3.14: Pseudocódigo - Obtenção de todos itens existentes da matriz de dados de entrada e suas frequências

Realizado esse processo, é recuperado a frequência dos itens identificados como itens

de classe que foram inicialmente agrupados e suprimidos, para que seus valores sejam devidamente considerados e o processo do *FPGrowth* não seja prejudicado pelo agrupamento. Com isso, temos todos os dados devidamente preparados para o processo de ordenação dos dados de entrada conforme suas frequências e obtenção de um banco de dados canônico para construção da FPTree.

Para ordenação dos dados de entrada, primeiramente ordenou-se os vetores de itens que serão a base desse processo. Para isso, utilizou-se do método *Quick Sort* — algoritmo de ordenação interna mais rápido que se conhece para uma ampla variedade de situações segundo [26] — sobre o vetor *ArrayIDcoef* ordenando consequentemente em paralelo o vetor *ArrayID* de acordo com a frequência de cada um dos itens. Baseado na técnica de divisão e conquista, neste algoritmo de ordenação o vetor é particionado em dois por meio de um procedimento recursivo. Essa divisão ocorre até que o vetor fique com apenas um elemento, enquanto os demais ficam ordenados à medida que ocorre o particionamento [27].

Tendo o vetor de itens ordenado, criou-se uma nova matriz *ArrayINord* que será construída a partir da análise da matriz *ArrayIN* de acordo com a ordenação dos itens presentes no *ArrayID*, conforme pode ser observado na Figura 3.15, podendo assim eliminar a antiga matriz *ArrayIN*, possuindo o banco de dados canônico para construção da FPTree no *ArrayINord*.

```
aux RECEBE 1
PARA n = f À 1 PASSO -1
  VarID RECEBE ArrayID(n)
  PARA n1 = 1 À NbLines2
    PARA n2 = 1 À NbColun
      SE ArrayIN (n1, n2) = ArrayID(n) ENTÃO
        FAÇA ENQUANTO ArrayINord(n1, aux) ≠ VAZIO
          aux RECEBE aux + 1
        CICLO ENQUANTO
        ArrayINord (n1, aux) RECEBE ArrayID(n)
        aux RECEBE 1
      FIMSE
    PRÓXIMO
  PRÓXIMO
PRÓXIMO
'
DELETE ArrayIN
```

Figura 3.15: Pseudocódigo - Ordenação da matriz de dados de entrada

Para inicialização da FPTree, foram criados os módulos de classe para estruturação

dos nós (classe *TreeNode*) e da árvore, com a implementação da função inserir recursiva para construção dos nós na árvore, dentro do módulo de classe *FPTree*, conforme pode ser observado nas Figuras 3.16, 3.20 e 3.21.

```
MÓDULO DE CLASSE TREENODE
Declaração de variável Value como String
Declaração de variável ValueCount como Long
Declaração de variável Pather como TreeNode
Declaração de variável Child como Collection
```

Figura 3.16: Pseudocódigo - Módulo de classe para definição do nó

Com as definições de módulos de classes disponíveis, o módulo do programa principal é inicializado com a criação de todas as variáveis necessárias para início da construção da FPTree, incluindo a criação do nó raiz (*root*), a lista *collection* denominada *PrefixSup* para agregar todos os itens de determinada transação para a chamada da função inserir (*insert1*) e também uma lista *collection* denominado *NodeRg* que foi criada para já ir coletando durante o processo de inserção dos nós, todas as folhas identificadas por transação. Conforme pode ser observado no trecho da Figura 3.17, cada transação de informação de entrada será agregada à lista *PrefixSup* e sempre realizado uma nova definição do *root* com a chamada da função *FPT.insert1*.

```
Declarações: FPT como FPTree; DEFINA root como TreeNode; DEFINA supNode como TreeNode; DEFINA PrefixSup como
Collection; DEFINA NodeRg como Collection; DEFINA Outputs como Collection; DEFINA OutPutFPTree () como String;
REDIMENSIONE OutPutFPTree() CONFORME TAMANHO (NbLines2, NbColun + 1)
'
DEFINA root como Novo TreeNode
DEFINIÇÕES root
  .Value RECEBE "VAZIO"
  .ValueCount RECEBE -1
FIM
'
aux RECEBE 1
PARA n1 = 1 À NbLines2
  PARA n2 = 1 À NbColun
    PrefixSup.ADICIONA_ITEM: ArrayINord (n1, n2)
    aux RECEBE aux + 1
  PRÓXIMO
    SE PrefixSup.Item(5) ≠ VAZIO ENTÃO
      PrefixSup.ADICIONA_ITEM: VAZIO
    FIMSE
    DEFINA root COMO FPT.insert1(root, PrefixSup, NodeRg)
    LIMPAR PrefixSup
PRÓXIMO
```

Figura 3.17: Pseudocódigo - implementação da construção FP-Tree.

Como a lista *NodeRg* sempre carrega a folha existente em cada transação de inserção, sempre haverá uma folha que durante o processo deixará de ser uma folha e passará a

ter filhos. Com isso, foi criado uma triagem, conforme Figura 3.18, para varrer a lista e eliminar os nós que não fossem mais folhas da árvore, para então verificar os resultados para impressão.

```
PARA nct = 1 À NodeRg.NumItens
  SE nct <= NodeRg.NumItens ENTÃO
    DEFINA supNode como NodeRg.Item(nct)
  FIMSE
  SE supNode.Child.NumItens ≠ 0 ENTÃO
    NodeRg.Remove (nct)
  FIMSE
PRÓXIMO nct
'
nct RECEBE VAZIO
VarID RECEBE VAZIO
```

Figura 3.18: Pseudocódigo - Eliminação dos nós "não folhas" do vetor.

Garantido o vetor com apenas folhas, é realizado a impressão dos dados a partir de cada uma delas, buscando o ponteiro de seu pai (sempre único) até chegar ao nó raiz (*root*) a agregando os resultados na matriz denominada *OutPutFPTree*, conforme demonstrado no trecho da Figura 3.19.

```
PARA aux = 1 À NodeRg.NumItens
  DEFINA supNode como NodeRg.Item(aux)
  VarID RECEBE supNode.Valor
  DEFINA Outputs como NOVO Collection
    Outputs.ADICIONA_ITEM: VarID
  Call PrintOut (supNode, Outputs, VarID)
  PARA n = 1 À Outputs.NumItens
    nct RECEBE Outputs.NumItens - n + 1
    VarID RECEBE Outputs.Item(nct)
    OutPutFPTree (aux, n) RECEBE VarID
  PRÓXIMO n
PRÓXIMO aux
'
n RECEBE aux – 1

FUNÇÃO PrintOut (Noderef como TreeNode, Outputs como Collection, VarID como String)
  SE Noderef.Pather (NÃO É VAZIO) ENTÃO VarID RECEBE Noderef.Pather.Valor
  SE VarID ≠ "VAZIO" ENTÃO Outputs.ADICIONA_ITEM: VarID
  SE Noderef.Pather (NÃO É VAZIO) ENTÃO PrintOut (Noderef.Pather, Outputs, VarID)
FIM FUNÇÃO
```

Figura 3.19: Pseudocódigo - Impressão dos resultados da FP-TREE

Uma premissa que deve ser avaliada nos resultados da FPTree, antes da impressão, é checar todas as folhas do último nó identificado como pai, ou seja, avaliar todos os nós folhas do penúltimo nível da árvore, pois, se as *strings* das folhas iniciarem com diferentes identificadores de *features*, exemplo1: ABCxx e DEFxx, e não termos diversidade como,

```
FUNÇÃO insert1(TN como TreeNode, PrefixSup como Collection, ctr como Collection) como TreeNode
  DEFINA Nd como TreeNode: DEFINA Item como TreeNode
  DEFINA i = 1: k = 0, f = 0, ck = 0
 SE TN.ValueCount = -1 ENTÃO
  ' Checar filhos no Nó inicial
    SE TN.Child (É VAZIO) ENTÃO
      DEFINA TN.Child como Novo Collection
      DEFINA Nd como Novo TreeNode
      DEFINA Nd COMO insert1(Nd, PrefixSup, ctr)
      DEFINA Nd.Pather COMO TN
      TN.Child.ADICIONA_ITEM: Nd
    SENÃO
      PARA cada Item em TN.Child
        DEFINA Nd como TN.Child.Item(i)
        SE PrefixSup.Item(1) = Nd.Value ENTÃO
          PrefixSup.Remove (1)
          Nd.ValueCount RECEBE Nd.ValueCount + 1
          SE PrefixSup.Item(1) ≠ VAZIO ENTÃO
            DEFINA Nd como insert1(Nd, PrefixSup, ctr)
          FIMSE
          k RECEBE 1
        FIMSE
        i RECEBE i + 1
      PRÓXIMO
      SE k = 0 And PrefixSup.Item(1) ≠ VAZIO ENTÃO
        DEFINA Nd como NOVO TreeNode
        DEFINA Nd como insert1(Nd, PrefixSup, ctr)
        DEFINA Nd.Pather como TN
        TN.Child.ADICIONA_ITEM: Nd
      FIMSE
    FIMSE
  SENÃO
```

Figura 3.20: Pseudocódigo - Módulo de classe FPTree - implementação da função inserir - parte 1

exemplo2: DAB00 e DAB01; todos que são *features* exclusivos deverão ser impressos juntos neste ramo da árvore, e criar apenas impressões distintas para folhas de mesmo identificador de atributos, como do exemplo2. Sendo assim, para um nó pai que possua os dados dos exemplos, esse ramo da árvore deveria ser impresso em direção à raiz apenas pela diversidade DAB00 e DAB01 e as demais folhas ABCxx e DEFxx aplicados às duas diversidade, exemplo:

$$varianteImpresso_1 = (...); DAB00; ABCxx; DEFxx$$

$$varianteImpresso_2 = (...); DAB01; ABCxx; DEFxx$$

.

Realizando-se a devida conferência, conforme descrito, avaliando todas as folhas de cada nó pai do penúltimo nível da árvore, toda impressão, com os resultados do *FPGrowth*, são registrados na matriz *OutPutFPTree*.

```
SENÃO
  SE TN.Child (É VAZIO) ENTÃO
    TN.Value RECEBE PrefixSup.Item(1)
    TN.ValueCount RECEBE 1
    DEFINA TN.Child como NOVO Collection
    PrefixSup.Remove (1)
    SE PrefixSup.Item(1) ≠ VAZIO ENTÃO
      DEFINA Nd como NOVO TreeNode
      DEFINA Nd como insert1(Nd, PrefixSup, ctr)
      DEFINA Nd.Pather como TN
      TN.Child.ADICIONA_ITEM: Nd
    FIMSE
  SENÃO
    PARA cada Item em TN.Child
      DEFINA Nd como TN.Child.Item(i)
      SE PrefixSup.Item(1) = Nd.Value ENTÃO
        PrefixSup.Remove (1)
        Nd.ValueCount RECEBE Nd.ValueCount + 1
        SE PrefixSup.Item(1) ≠ VAZIO ENTÃO
          DEFINA Nd como insert1(Nd, PrefixSup, ctr)
        FIMSE
        k RECEBE1
      FIMSE
      i RECEBE i + 1
    PRÓXIMO
    SE k = 0 And PrefixSup.Item(1) ≠ VAZIO ENTÃO
      DEFINA Nd como NOVO TreeNode
      DEFINA Nd como insert1(Nd, PrefixSup, ctr)
      DEFINA Nd.Pather como TN
      TN.Child.ADICIONA_ITEM: Nd
    FIMSE
  FIMSE
FIMSE
'
SE TN.Child.NumItens = 0 ENTÃO
  ctr.ADICIONA_ITEM: TN
FIMSE
'
DEFINA insert1 como TN
FIM FUNÇÃO
```

Figura 3.21: Pseudocódigo - Módulo de classe FPTree - implementação da função inserir - parte 2

3.4.3 Adequação dos resultados agregando os dados do agrupamento

Para chegada ao resultado final esperado e identificação de todas as variantes de chicote necessárias para atender determinado projeto, com a descrição detalhada de todos os *features* associados a cada um deles, conforme representado pela Figura 3.22, é utilizado uma lista *collection* denominada *Outputs* para recuperar todos os dados de cada um dos resultados presentes na matriz *OutPutFPTree*, coletando sempre em um único item *string* cada um dos resultados.

Ao acrescentar cada um destes dados, é identificado o atributo de nível veículo, sendo assim, após a conclusão da recuperação dos dados presentes na matriz é associado o resultado do agrupamento, agregando os dados agrupados pertencentes a esta classe de

nível logo na sequência, no mesmo item de resultados desta lista *Outputs*.

```
out RECEBE Vazio
DEFINA Outputs como NOVO Collection
PARA n1 = 1 À NbLines2
  VarID RECEBE OutPutFPTree (n1, 1)
  n2 RECEBE1
    ENQUANTO OutPutFPTree (n1, n2) ≠ VAZIO
      out RECEBE out + "; " + OutPutFPTree (n1, n2)
      n2 RECEBE n2 + 1
    Wend
  PARA n2 = 1 À nct
    SE VarID = ArrayCluster (n2, 1) ENTÃO
      out RECEBE out + "; " + ArrayCluster (n2, 2)
    FIMSE
  PRÓXIMO
  SE out ≠ VAZIO ENTÃO out RECEBE Right (out, Len(out) - 2)
  SE out ≠ VAZIO ENTÃO Outputs.ADICIONA_ITEM: out
  out RECEBE Vazio
PRÓXIMO
'
DELETE ArrayCluster
DELETE OutPutFPTree
```

Figura 3.22: Pseudocódigo - União dos resultados da FP-TREE e Agrupamento

Sendo assim, chega-se nos resultados de variantes de chicotes necessários pela lista *Outputs*, onde cada um dos itens representa uma variante de chicote, contendo todos os *features* associados a cada uma dessas variantes.

Capítulo 4

Resultados

Com o exemplo do pseudoprojeto veículo explorado para desenvolvimento do algoritmo, considerando os dados de entrada dos Apêndice A e Apêndice B, imprimindo os resultados obtidos na lista *Outputs* através do comando *Debug.Print*, para vizualização dos resultados na janela de Verificação Imediata do Visual Basic, foram obtidos os resultados de variantes e seus respectivos *features* associados, conforme a Figura 4.1.

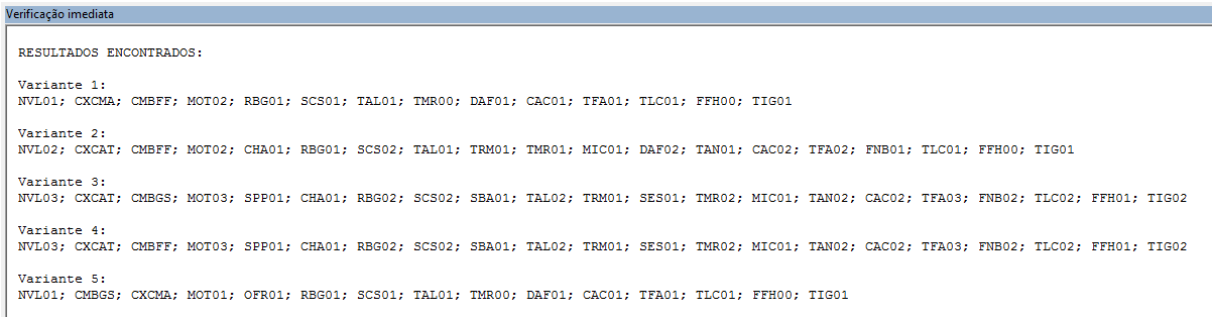

Figura 4.1: Janela de verificação imediata com a impressão dos resultados finais

Ou seja, nesse pseudoexemplo, identificou-se cinco variantes, sendo duas diversidades previstas para o nível 1, uma diversidade prevista para nível 2 e duas diversidades para o nível 3. Dessa forma, tendo correlacionados a identificação de todos os *features* associados, garante-se assim a realização da amarração correta de todos os planos de esquemas elétricos dos componentes para cada uma das variantes.

Seguindo os agrupamentos descritos na sessão 3.4.1, criando-se um novo input para conferência de todas as tratativas de agrupamentos, veja os testes que seguem:

1. Supondo que haja uma requisição do fornecedor, devido ao seu processo de produção por exemplo, que seja destacado a necessidade de um circuito de *ground* (GND ou

aterramento) todas as vezes que se tratar de uma versão de chicote que tenha a caixa de câmbio manual (CXCMA) associada à Optmização de Freio (OFR01); Considerando o mesmo banco de dados da tabela 3.6, incluíndo mais um input que seria: $ID : 46 = CXCMA; OFR01; GND01$, Pode-se observar através da variável *ArrayAux2*, neste caso, a tratativa de agrupamento e posterior adição desse impacto do caso especial ao resultado final da resolução do algoritmo, conforme a Figura 4.2, onde tem-se destacado em azul os *features* que formam os pré-requisitos da regra especial e em verde a adição do atributo de requisição especial.

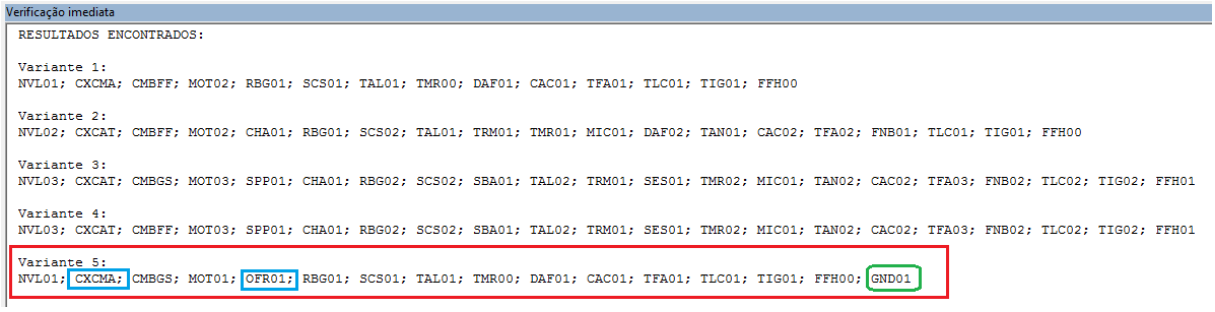

Figura 4.2: Resultado com adição de casos especiais nos dados de entrada

b. No caso agrupado pelo *ArrayAux4*, considerando o atributo de motorização (MOTxx) mas que não estão amarrados à todas as possibilidades de atributos de câmbio(CXCxx) e combustível(CMBxx) para cada um deles, pode-se considerar por exemplo a necessidade de um sensor de temperatura (ex.:TMP01) para todo motor tipo 3 (MOT03); Considerando o mesmo banco de dados da tabela 3.6, incluíndo mais um input que seria: $ID : 47 = NVL03; MOT03; TMP01$, neste caso, tem-se a tratativa de agrupamento e adaptação das informações no *ArrayAux4* e posterior adição desses valores novamente no banco de dados de entrada para rodagem do algoritmo *FPGrowth*. O resultado final da resolução do algoritmo pode ser observado na Figura 4.3.

Com esses dados adicionais foi possível testar todas as outras possíveis situações que gerariam os demais agrupamentos descritos que foram levantados sobre hipóteses quando avaliados com o engenheiro de desenvolvimento deste produto da empresa em questão. Todos os dados foram satisfatórios, seguindo uma implementação manual com estes dados sintéticos de entrada, o engenheiro chegaria nos mesmo valores similares, que seriam os dados descritos na tabela 4.1, disponibilizados como exemplo pelo engenheiro consultado. Tabela precisou ser adaptada e reduzida para conseguirmos adiciona-lá para melhor vizualização, mas sua totalidade pode ser consultado no Apêndice D.

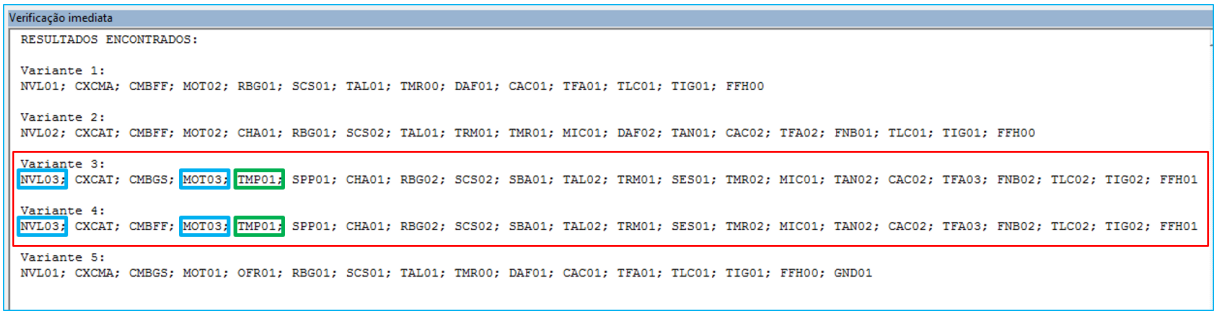

Figura 4.3: Resultado com adição de dados correlacionado a motorização com correlação incompleta nos dados de entrada.

	Versão de veículo	CXCMA	CXCAT	CMBFF	...	TIG01	FFH01
Variante A	NVL01; CXCMA; CMBFF; MOT02;[...]; TIG01; FFH00	x		x	...	x	
Variante B	NVL02; CXCAT; CMBFF; MOT02; [...]; TIG01; FFH00		x	x	...	x	
Variante C	NVL03; CXCAT; CMBGS; MOT03; [...]; TIG02; FFH01		x		...		x
Variante D	NVL03; CXCAT; CMBFF; MOT03; [...]; TIG02; FFH01		x	x	...		x
Variante E	NVL01; CXCMA; CMBGS; MOT01; [...]; TIG01; FFH00	x			...	x	

Tabela 4.1: Tabela resumida de diversidade de chicotes construída manualmente

Todo o processo para chegar-se a esses resultados, partindo do tempo para execução e resolução da problemática, lembrando que estes dados sintéticos são apartir de um exemplo bem mais simples e didático com menos complexidade que um projeto de carro real, obteve-se um tempo gasto para manuseio das informações e resultado manual de 50min. Em contrapartida, considerando os inputs agregados à ferramenta proposta, levando-se em consideração todo o processo de preenchimento dos inputbox para geração dos resultados, obteve-se um tempo total de 10min. Ou seja, foi otimizado 80% do tempo gasto para a obtenção dos resultados de variantes do chicote principal.

Considerando o caso real da empresa utilizada como referência no estudo, no qual o prazo padrão para conclusão dessa etapa do desenvolvimento são de 15 dias, sabe-se que o chicote principal é o responsável pelo impacto de 80% no tempo total deste processo. Sendo assim, levando em consideração o ganho de 80% sobre esse valor, chega-se a um total de aproximadamente 64% de ganho neste tempo, reduzindo de 15 para aproximadamente 5 dias esse processo, o que em dias úteis significam 3 semanas reduzidas em 1 semana para fechamento desta etapa.

Capítulo 5

Conclusões e Trabalhos Futuros

Apartir dos resultados obtidos, consegue-se atingir os objetivos previamente especificados por este estudo, com a automatização do processo e rápida obtenção das variantes (ou diversidade) de chicote principal de acordo com cada entrada de projetos utilizando-se da ferramenta Excel®. Visto que ainda não há na empresa de estudo um sistema centralizado que disponibiliza todas as informações por extrações padronizadas de acordo com cada projeto, e os inputs ainda são por planilhas que acabam sendo recebidas com diferentes formatações para os dados, a utilização de (*InputBox*), um recurso disponível para ser programado no Excel®, com premissas no algoritmo para gerar alertas evitando o preenchimento incorreto, atendendeu bem os dados de entrada para os algoritmos que solucionam a problemática.

Considerando os resultados satisfatórios, levando sem consideração que o prazo padrão em casos reais seria 15 dias para obtenção dos dados de diversidade, sendo este projetos complexos e reais o que leva a um banco de dados muito maior, podemos chegar a equivalência de resultados. Sendo assim, podemos concluir que, para um ganho de 70% passamos de um total de 15 dias, para apenas 5 para chegarmos ao resultado final, podendo este ainda ser melhorado conforme a familiaridade com a ferramenta for aumentando.

Os algoritmos propostos são sistemas interpretáveis e facilmente compreensíveis, sendo assim, passa uma maior confiabilidade devido sua transparência, fazendo com que seja possível a compreensão do conhecimento interno do modelo, facilitando a aplicação futura desse modelo dentro de um sistema novo que a empresa em estudo venha à desenvolver, permitindo também que, se necessário futuramente alguma adaptação ou mesmo manutenção, não dificulte os terceiros a atuarem no que for necessário. Sem a utilização de algoritmos de estrutura *Black Boxes* (ou sistemas opacos), conseguiu-se uma solução

tradicionalmente explicável que não precisa de algoritmos auxiliares para verificação.

Como oportunidades futuras, fica a adaptação para agregar as demais famílias de chicotes que são menores e com menos *features* associados, gerando uma única vez todo o conjunto de resultado por famílias de chicotes, tendo referenciado bem no algoritmo o tipo de arquitetura que amarrá quais *features* impactam qual família. Como o principal é o que carrega a maior diversidade e *features* associados, poderia-se partir já desses resultados como novos dados de entrada agregando os demais *features* para rodar novamente os algoritmos e obter as informações para os demais chicotes.

Referências

[1] RATZ, A. V. Association rules learning (arl): Part 1 - apriori algorithm. *The Code Project Open License (CPOL)* (2019). Published in Jully. Canada.

[2] RATZ, A. V. Association rules learning (arl): Part 2 - fpgrowth algorithm. *The Code Project Open License (CPOL)* (2019). Published in Jully. Canada.

[3] NORONHA, J. C. E. A. Opções reais aplicadas à gestão do processo de desenvolvimento de produtos em uma indústria de autopeças. UNIFEI - Universidade Federal de Itajubá. (2014) Minas Gerais, Brasil - Gest. Prod., São Carlos, v. 21, n. 1.

[4] SILVA, L. C. Aplicação da teoria das opções reais para tomada de decisão em adotar política pública: O caso inovar-auto. Dissertação de Mestrado, UNESP - Universidade Estadual Paulista – Júlio de Mesquita Filho, Guaratinguetá -SP, Brasil, 2016.

[5] KUHN, M.; NGUYEN, H. The future of harness development and manufacturing. University of Erlangen-Nürnberg - Germany. Case Study - January, 2019.

[6] KING, G. Bentley bentayga wiring harness is weirdly beautiful. From: Jul 12, 2016 at 3:00pm ET. <https://www.motor1.com/news/65202/ bentley-bentayga-wiring-harness-is-weirdly-beautiful/>.

[7] CARBONERA, JOEL; GONÇALVES, B. S. C. O problema da explicação em inteligência artificial: considerações a partir da semiótica. *Teccogs: Revista Digital de Tecnologias Cognitivas* (2018). PUC-SP, São Paulo.

[8] ARRIETA, A. B. E. A. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *TECNALIA. P. Tecnologico, Ed. 700. 48170 Derio (Bizkaia), Spain.* (2019). Partnership of 8 institutions from France and Spain.

[9] WACHTER, SANDRA; MITTELSTADT, B. R. C. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harvard Journal of Law Technology* (2018). Volume 31, Number 2 Spring.

[10] BRANDÃO, R. E. A. Mediation challenges and socio-technical gaps for explainable deep learning applications. IBM Research, UFRGS e PUC-Rio (2018).

[11] MATHEWSON, W. KORY; PILARSKI, M. P. A brief guide to designing and evaluating human-centered interactive machine learning. *Published at ML Evaluation Standards Workshop at ICLR* (2022). DeepMind.

[12] FREITAS, A. A. Comprehensible classification models – a position paper. *SIGKDD Explorations* (2014). School of Computing, University of Kent, Canterbury, CT2 7NF, UK.

[13] CASTRO, Leandro N.; FERRARI, D. G. *Introdução a Mineração De Dados – Conceitos Básicos, Algoritmos e Aplicações*. Saraiva, São Paulo, SP, 2016. 1ª edição.

[14] MARIANO, M. A. Comparação de algoritmos paralelos para a extração de regras de associação no modelo de memória distribuída. Dissertação de Mestrado, Universidade Federal de Mato Grosso do Sul (FACOM-UFMS), Campo Grande, Mato Grosso do Sul, Brasil, 2011.

[15] SOCZEK, Felipe C.; ORLOVSKI, R. Mineração de dados: Conceitos e aplicação de algoritmos em uma base de dados na Área da saúde. Faculdade Guairacá. Tecnologia em Análise e Desenvolvimento de Sistemas – Guarapuava, PR. Novembro, 2019.

[16] RIGO, S. J. e. a. Aplicações de mineração de dados educacionais e learning analytics com foco na evasão escolar: oportunidades e desafios. *Revista Brasileira de Informática na Educação, Volume 22, Número 1* (2014). Universidade do Vale do Rio dos Sinos - Unisinos (Programa Interdisciplinar de Pós-Graduação em Computação Aplicada) e Universidade Federal de Ciências da Saúde de Porto Alegre (Departamento de Educação e Informação em Saúde).

[17] CASSIANO, K. M. *Análise de Séries Temporais usando Análise Espectral Singular (SSA) e Clusterização de suas Componentes baseada em Densidade*. PhD thesis, PUC - Pontifícia Universidade Católica do Rio de Janeiro, 2015. Rio de Janeiro-RJ, Brasil.

[18] GARCIA, Cássio Alan; FROZZA, R. Sistema de recomendação de produtos utilizando mineração de dados. *TECNO-LÓGICA, Revista do depto. de Química e Física, do depto. de Engenharia, Arquitetura e Ciências Agrárias e do Mestrado em Tecnologia Ambiental* (2013). Universidade de Santa Cruz do Sul (Departamento de Informática) e Universidade de Santa Cruz do Sul (Programa de Pós-Graduação em Sistemas e Processos Industriais [Mestrado]).

[19] Al-jabery K. K., W. I. C. D. Clustering algorithms. Computational Learning Approaches to Data Analytics in Biomedical Applications, 2020.

[20] OYAMA, F. T. Mineração multirrelacional de regras de associação em grandes bases de dados. Dissertação de Mestrado, UNESP - Universidade Estadual Paulista – Júlio de Mesquita Filho, São José do Rio Preto -SP, Brasil, 2010.

[21] MEERA, Narvekara; SHAFAQUE, F. S. An optimized algorithm for association rule mining using fp tree. *International Conference on Advanced Computing Technologies and Applications (ICACTA)* (2015). College of Engineering, Mumbai and 400056, India.

[22] BORGELT, C. An implementation of the fpgrowth algorithm. School of Computer Science, Department of Knowledge Processing and Language Engineering.

[23] HAN, Jiawei; PEI, J. Y. Y. Mining frequent patterns without candidate generation. *Simon Fraser University* (2000). School of Computing Science. Canada.

[24] HUNYADI, D. Performance comparison of apriori and fp-growth algorithms in generating association rules. *Proceedings of the European Computing Conference* (2011). University of Sibiu, Romania (Department of Computer Science).

[25] KAVITHA, Mrs. M.; SELVI, M. Comparative study on apriori algorithm and fp growth algorithm with pros and cons. *International Journal of Computer Science Trends and Technology (IJCST) – Volume 4 Issue 4, Jul - Aug* (2016). Department of Computer Science - Tiruppur, Kumaran College for Women and Tamil Nadu, India.

[26] ZIVIANI, N. *Projeto de algoritmos com implementações Pascal e C.* Pioneira, São Paulo, SP, 1999. 4ª edição.

[27] ASCENVIO, Ana Fernanda G.; ARAÚJO, G. S. d. *Estrutura de Dados – Algoritmos, análise da complexidade e implementações em JAVA e C/C++.* Pearson, São Paulo, SP, 2010.

APÊNDICE A – Exemplo de pseudocódigo de features e impactos por famílias de chicotes

		FEATURE CODES	Principal	Painel	Portas D.	Portas T.	BAT +	BAT -	Motor	Para Choq	Pre Aquec
1	POWERTRAIN										
	Motor	MOTxx	x						x		
	Tipo de Câmbio	CXCxx	x					x	x		
	Alternador	ALTxx							x		
	Motor de Partida	MPAxx							x		
	Controle nivel Bateria	CBTxx	x					x			
	Tipo de combustível	CMBxx	x								x
2	SECURITY										
	Tipos de Freio	TFRxx	x								
	Sensor pressão pneu	SPPxx	x								
	Controle Hill Assist	CHAxx	x								
	Tipos de Airbag	RBGxx	x	x							
	Sensor cinto segurança	SCSxx	x								
	Sensor no banco	SBAxx	x								
	Otimização de Freio	OFRxx	x				x				
3	ADAS (*Adv. driver assist. system*)										
	Limitador/Reg. Velocidade	LRVxx		x							
	Tipo de alarme	TALxx	x	x							
	Tipo de trancas (nas portas,..)	TTRxx	x	x	x	x					
	Tipo de trancas porta malas	TRMx	x	x							
	Tipo de alerta ao condutor	ACOxx		x							
	Sensor de estacionamento	SESxx	x	x						x	
	Camera de ré panoramica	CRPxx	x	x							
4	INFOTAINMENT										
	Tipo de multimídea	TMRxx		x							
	Microfone - acessório	MICxx	x	x							
	Diversidade de Alto Falantes	DAFxx	x	x	x	x					
	Tipos de antena	TANxx	x	x							
	Tipos de painel de instrumentos	TPIxx		x							
5	COMFORT										
	Tipo de direção	TDAxx	x								
	Tipo de ar condicionado	TACxx	x	x							
	Compressor do AC	CACxx					x				
	Tipo de levanta vidros	TLVxx	x	x	x	x					
	Tipo de ignição (chave)	TIGxx	x	x							
	Função Free Hands (acesso remoto)	FFHxx	x	x							
	Tipos de lava vidros	DLVxx	x	x							
	Tipo de retrovisores	TRDxx/TRExx		x	x						
	Desembaçador de vidro traseiro	DVTxx	x	x							
	Tipo de acessórios (USB, etc)	USB/TUSxx	x	x							
	Tipo de ajuste dos bancos	TABxx	x								
6	EXTERIOR										
	Tipos de Farois	TFAxx	x	x							
	Farois neblina	FNBxx	x	x							
	Tipo de indicação de seta	TISxx	x	x							
	Tipo de lanternas traseiras	TLTxx	x	x							
	Brake light	TBLxx	x								
	Sensor de temperatura EXT	STPxx		x	x						
7	INTERIOR										
	Botão ECO	BECxx		x							
	Tipos de luz de cortesia	TLCxx	x								
	Luz porta-malas	LPMxx	x								
	Tipos de controle no volante	TCVxx	x	x							
	Quantidade de passageiros	TABxx	x								
	Prise 12v Socket	TTVxx		x							
8	PARTICULARIDADES										
	Conexão p/ Instrumentação	TIVxx	x	x							
	Outros controle no volante	OCVxx	x	x							
	Níveis Veículo	NVVxx	x	x	x	x					
	Tipo de país a ser comerc.	PCVxx	x	x	x						
	Tipo de família veículo (projeto)	FVPxx	x	x	x	x	x	x	x	x	x

Tabela A.1: Exemplo de pseudocódigo de features e impactos por famílias de chicotes

APÊNDICE B – Exemplo de pseudoficha de definição de um projeto veículo

	NVL01 (Nível 1)	NVL02 (Nível 2)	NVL03 (Nível 3)	
Motor tipo 1 - gasolina	x			MOT01
Motor tipo 2 - Flex Fuel	x	x		MOT02
Motor tipo 3 - FlexFuel/gasolina			x	MOT03
Câmbio Manual	x			CXCMA
Câmbio Automático		x	x	CXCAT
Combustível gasolina	x		x	CMBGS
Combustível Flex Fuel	x	x	x	CMBFF
Sensor de pressão pneu presente			x	SPP01
Controle Hill Assist presente		x	x	CHA01
Airbag regulamentar (frontais)	x	x		RBG02
Airbag completo (laterais+frontais)			x	RBG02
Sensor cinto segurança Motorista apenas	x			SCS01
Sensor cinto segurança Motorista/Passageiro.		x	x	SCS02
Sensor no banco frontal passageiro presente			x	SBA01
Otimização de Freio	MOT01			OFR01
Alarme Volumétrico			x	TAL02
Alarme Perimétrico	x	x		TAL01
Tranca Porta Malas Manual		x	x	TRM01
Tranca Porta Malas Elétrica		x	x	TRM01
Sensor Estacionamento presente			x	SES01
Tipo de multimídea A	x			TMR00
Tipo de multimídea B		x		TMR01
Tipo de multimídea C			x	TMR02
Microfone - acessório presente		x	x	MIC01
Diversidade de Alto Falantes - qtd.: 2	x			DAF01
Diversidade de Alto Falantes - qtd.: 4		x	x	DAF02
Tipos de antena passiva		x		TAN01
Tipos de antena ativa			x	TAN02
AC controle Manual	x			CAC01
AC controle Auto		x	x	CAC02
Ignição c/ chave mecânica	x	x		TIG01
Ignição por botão (chave remota)			x	TIG02
Função Free Hands ausente (s/ acesso remoto)	x	x		FFH00
Função Free Hands presente (c/ acesso remoto)			x	FFH01
Farol Lampada + DRL Lampada	x			TFA01
Farol Lampada + DRL LED		x		TFA02
Farol LED + DRL LED			x	TFA03
Farois neblina Lâmpada		x		FNB01
Farois neblina LED			x	FNB02
Luz de cortesia Lâmpada	x	x		TLC01
Luz de cortesia LED			x	TLC02

Tabela B.1: Exemplo de pseudoficha de definição de um projeto veículo

APÊNDICE C – VBA utilizado para a aplicação

Para acessar todo o código de aplicação desenvolvido no VBA com todos os códigos do *forms* para os *inputbox* de inicialização do código, assim como para os *inputs* do banco de dados pelo engenheiro responsável, você pode enviar uma solicitação ao link disponibilizado. O nome dado para a ferramenta desenvolvida foi HVD code - Harness Variant Discovery, na qual todo código VBA pode ser acessado pelo link: `https://drive.google.com/drive/folders/1xuAHupux2fcLYa-RMW1YsdeQ2LlyrsnH?usp=sharing`

APÊNDICE D – Tabela completa de diversidade de chicotes construída manualmente

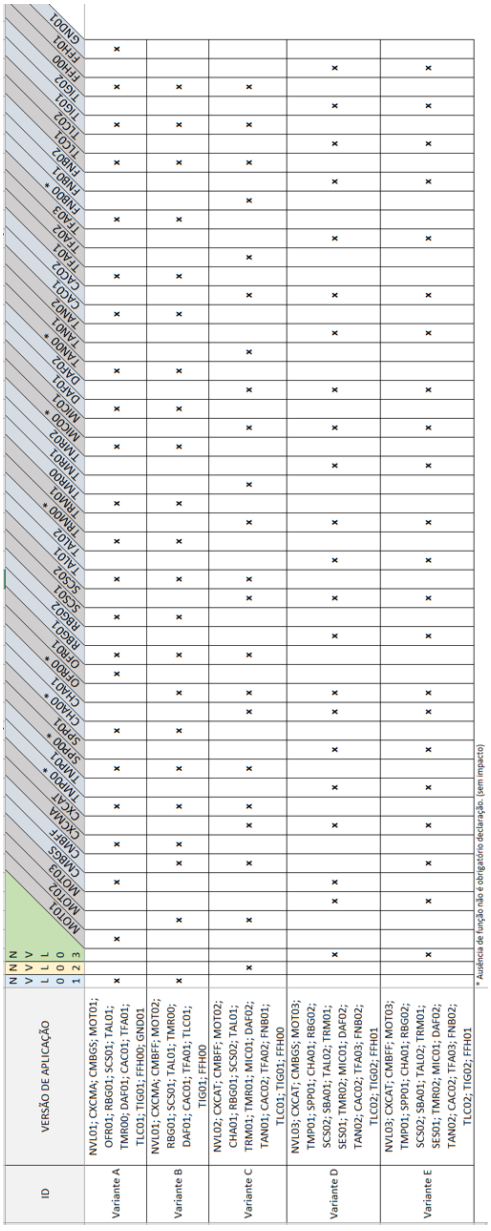

Figura D.1: Tabela completa de diversidade de chicotes construída manualmente

APÊNDICE E – Todos Inputbox utilizados para inicialização da ferramenta proposta

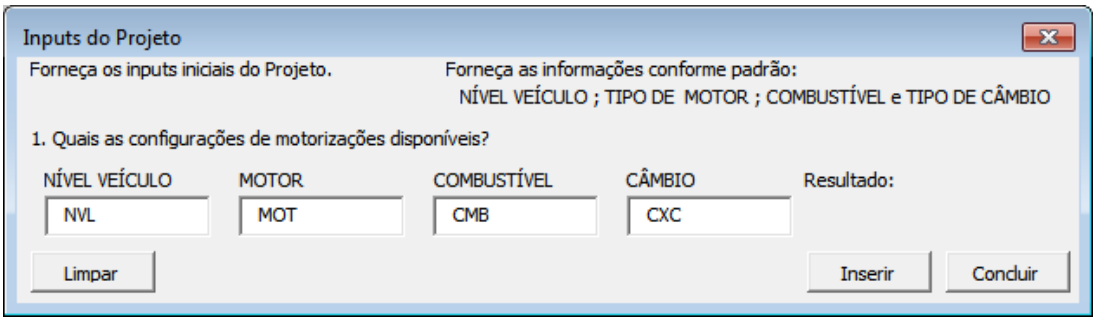

Figura E.1: Inputbox1

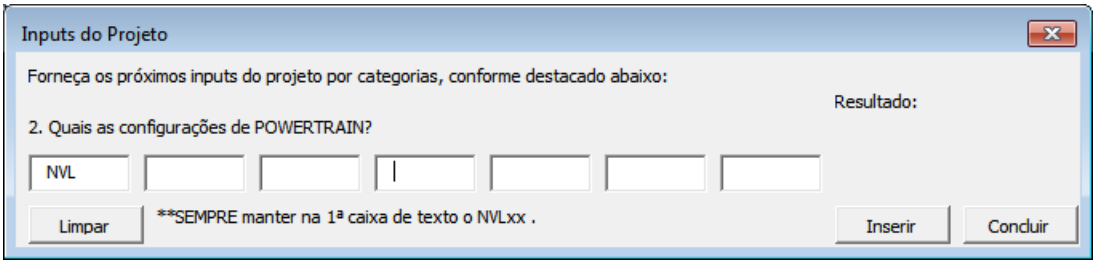

Figura E.2: Inputbox2

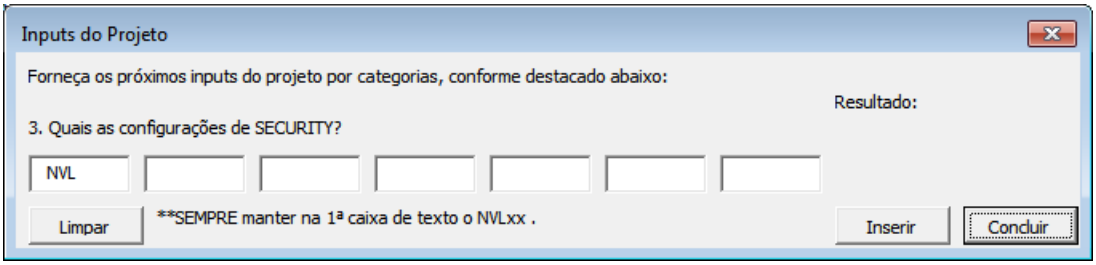

Figura E.3: Inputbox3

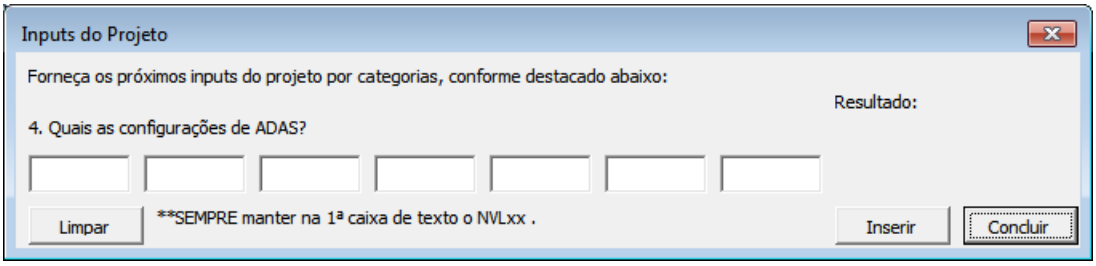

Figura E.4: Inputbox4

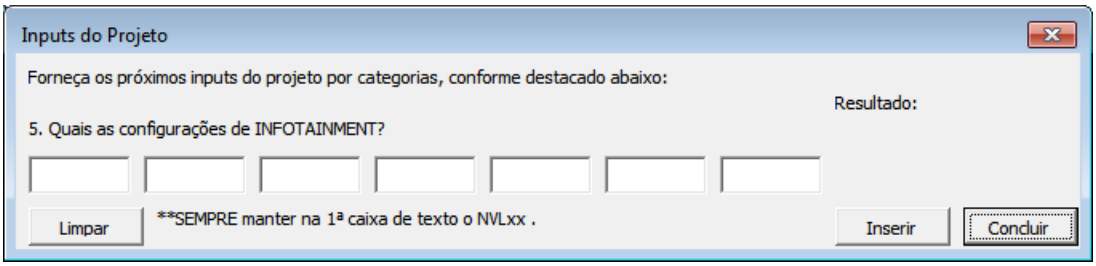

Figura E.5: Inputbox5

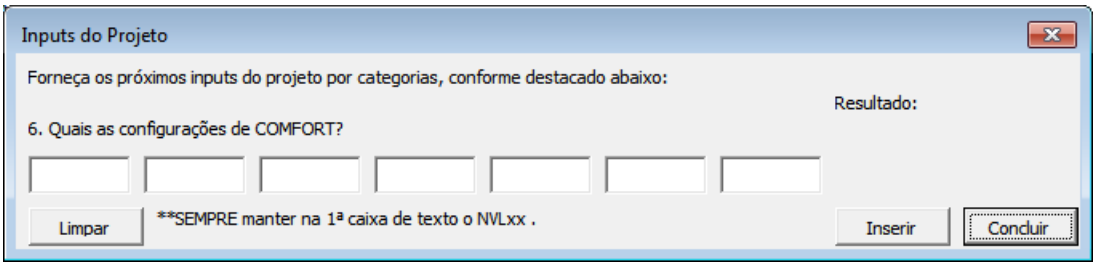

Figura E.6: Inputbox6

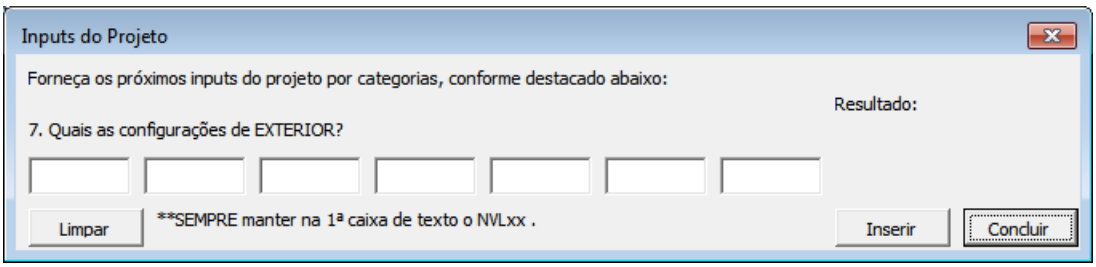

Figura E.7: Inputbox7

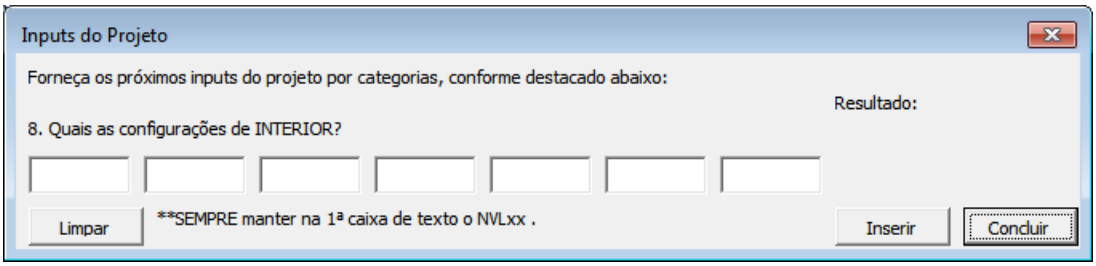

Figura E.8: Inputbox8

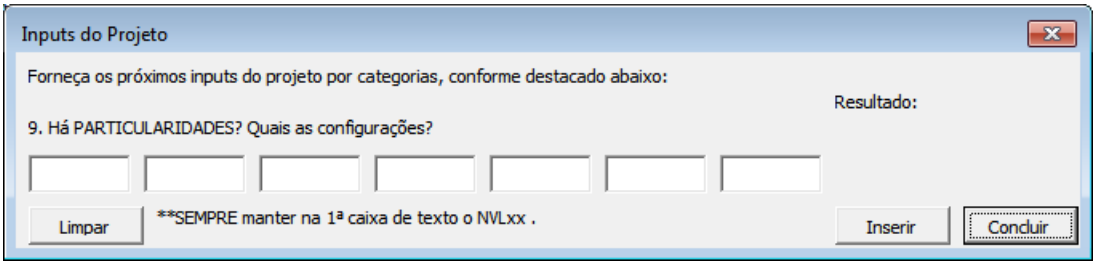

Figura E.9: Inputbox9