Uma Abordagem para Detecção de *Outliers* em Dados Categóricos

Flávio Roberto Silva

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Uma Abordagem para Detecção de *Outliers* em Dados Categóricos

Flávio Roberto Silva[1]

27 de fevereiro de 2004

Banca Examinadora:

- Prof. Dr. Geovane Cayres Magalhães
  Instituto de Computação, UNICAMP (Orientador)
- Profa. Dra. Cristina Dutra de Aguiar Ciferri
  Departamento de Informática, Universidade Estadual de Maringá
- Prof. Dr. Neucimar Jerônimo Leite
  Instituto de Computação, UNICAMP

[1]**Suporte financeiro da CAPES e apoio parcial do SAI/PRONEX-MCT.**

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Silva, Flávio Roberto
Si38a Uma abordagem para detecção de outliers em dados categóricos / Flávio Roberto Silva – Campinas, [S.P. :s.n.], 2004.

Orientador: Geovane Cayres Magalhães
Dissertação(Mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Banco de dados. 2. Sistemas de recuperação de informação. 4. Modelos log-lineares. I. Magalhães, Geovane Cayres. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Uma Abordagem para Detecção de *Outliers* em Dados Categóricos

Este exemplar corresponde à redação final da Dissertação devidaménte corrigida e defendida por Flávio Roberto Silva e aprovada pela Banca Examinadora.

Campinas, 27 de fevereiro de 2004.

Prof. Dr. Geovane Cayres Magalhães
Instituto de Computação, UNICAMP
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 27 de fevereiro de 2004, pela Banca examinadora composta pelos Professores Doutores:

Profa. Dra. Cristina Dutra de Aguiar Ciferri
UEM

Prof. Dr. Neucimar Jerônimo Leite
IC - UNICAMP

Prof. Dr. Geovane Cayres Magalhães
IC – UNICAMP

À toda minha família, com carinho.

The formulation of a problem is often more essential than its solution,
which may be merely a matter of mathematical or experimental skill.
To raise new questions, new possibilities, to regard old problems from a new angle,
requires creative imagination and marks real advance in science.
Albert Einstein.

Resumo

Outliers são elementos que não obedecem a um padrão do conjunto de dados ao qual eles pertencem. A detecção de *outliers* pode trazer informações não esperadas e importantes para algumas aplicações, como por exemplo: descoberta de fraudes em sistemas telefônicos e de cartão de crédito e sistemas de detecção de intrusão.

Esta dissertação apresenta uma nova abordagem para detecção de *outliers* em bancos de dados com atributos categóricos. A abordagem proposta usa modelos log-lineares como um padrão para o conjunto de dados, o que torna mais fácil a tarefa de interpretação dos resultados pelo usuário.

Também é apresentado o FOCaD (*Finding Outliers in Categorical Data*), protótipo de um sistema de análise de dados categóricos. Ele ajusta e seleciona modelos, faz testes estatísticos e detecta *outliers*.

Abstract

An outlier is an element that does not conform to a given pattern to a set. Outlier detection can lead to unexpected and usefull information to some applications, e.g., discovery of fraud in telephonic and credit card systems, intrusion detection systems.

This Master Thesis presents a new approach for outlier detection in databases with categorical attributes. The proposed approach uses log-linear models as a pattern for the dataset, which makes easier the task of interpreting results by the user.

It is also presented FOCaD (Finding Outliers in Categorical Data), a prototype of a categorical data analysis system. It adjusts and selects models, performs statistic tests, and outlier detection.

Agradecimentos

Em primeiro lugar devo agradecer a Deus por me proporcionar esta grande oportunidade de cursar o mestrado em uma ótima instituição de ensino.

Meu pai José e minha mãe Maria, obrigado por todo apoio, carinho e amor que me deram durante toda minha vida. Obrigado Núbia e Ângelo, meus queridos irmãos que sempre estiveram comigo. Um especial agradecimento aos meus tios Valdivinho e Ireni pelo acolhimento com muito carinho durante toda minha graduação.

Ao professor Geovane, meu orientador, que me ajudou neste caminho até aqui sempre com paciência e compreensão.

À professora Claudia pela condução do grupo de banco de dados do IC e os preciosos conselhos.

Foram muitos os amigos que percorreram e cruzaram esta jornada comigo. Desde já peço desculpas por não citar o nome de todos que merecem. Aos companheiros da primeira república Flavão e Pára. Agradecimento ao Celso (Blues Boy), Fernando (Ninja), Chenca (Chenca), Marcelo (Baboo), Marcelo (Chosen One), Lasáro (do Contra), Quintão (Parede) e o argentino Aníbal (argentino), companheiros da atual república "Os Malditos" pelas risadas e partidas de pembolim. Obrigado pelo apoio quando cheguei a Campinas Glauber e William, amigos desde a graduação na UFU. Foram bons momentos no LIS e nas reuniões semanais; obrigado pelas dicas, sugestões e principalmente pelo companheirismo Daniel, Fileto, João Guilherme, Juliano, Ricardo Torres e Randall. Aos amigos que ingressaram e trabalharam comigo Henrique, Silvânia, Gustavo Kasprzak e Nielsen. Valeu pelas valiosas dicas no trabalho e pela amizade Wesley. Como a lista está ficando grande, agradeço os amigos que vieram no sul, norte, nordeste, centro-oeste e da minha região sudeste pelas horas de lazer, pelo futebol e pelas festas.

Aos colegas de trabalho pelo grande incentivo. Especial agradecimento a Adriana e ao Alonso, que me apoiaram e foram compreensivos nestes últimos momentos.

Ao CPqD, especialmente à Celly e Cleida, por me fornecerem dados e ajuda para validar este trabalho.

À CAPES e ao projeto SAI-PRONEX/MCT pelo apoio financeiro. Ao IC-UNICAMP pela infra-estrutura fornecida indispensável ao desenvolvimento deste trabalho.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

One person's noise is another person's signal.
Anônimo.

Com a informatização de vários setores da sociedade, uma enorme quantidade de dados é gerada e armazenada em bancos de dados. Um exemplo encontrado no dia-a-dia é de uma pessoa ligando para sua administradora de cartão de crédito, pedindo o cancelamento de uma dívida na sua fatura referente à assinatura de uma revista. Neste exemplo existem os registros armazenados pelas companhias telefônicas envolvidas para completar a ligação; o registro do atendimento efetuado pelo *call center* da administradora do cartão; o registro do cancelamento do cartão pois ele pode ter sido "clonado". Nota-se que um simples fato ocorrido pode gerar muitos dados que serão armazenados por vários atores diferentes.

Individualmente, cada dado armazenado possui uma informação associada. O registro do *call center* da administradora do cartão significa que o cliente ligou e pediu o cancelamento do cartão por indício de clonagem. Contudo, o conjunto dos registros armazenados devem ser estudados para se obter informação. Analisando as faturas de seu cliente, a administradora do cartão poderia notar que ele nunca havia feito a assinatura de nenhuma revista. Assim, no momento que ela tomasse conhecimento do fato, ela poderia ligar para seu cliente e pedir uma confirmação, evitando futuros transtornos.

Como são muitos dados, é impossível analisá-los "manualmente". É necessário então a utilização de ferramentas semi-automáticas para auxiliar o usuário nesta análise. O usuário deve possuir sumários dos dados e ferramentas que indiquem subconjuntos nos quais seja mais provável que ele possa encontrar alguma informação interessante.

Devido a estas necessidades surgiram as ferramentas de mineração de dados ou KDD (Knowledge Discovery in Databases). Fayyad [FPSS96] definiu KDD como "um processo

não trivial de identificar padrões válidos, novos, potencialmente úteis e compreensíveis". Em geral, o analista procura por padrões nos dados, como grupos de consumidores com o mesmo comportamento para que uma campanha de *marketing* possa ser bem planejada. Este é o problema de descoberta de *clusters*, conhecido como *clustering* [HKT01].

Existem dois fatores principais que podem ser usados para caracterizar métodos de mineração de dados. O primeiro é a capacidade de lidar com grandes conjuntos de dados. Isto, em geral, é obtido através de técnicas desenvolvidas pela área de banco de dados. O segundo fator é o fato de exigir pouca interação do usuário, alcançado com a utilização de técnicas de estatística, inteligência artificial e reconhecimento de padrões.

Em mineração de dados tradicionalmente busca-se por padrões para o conjunto de dados. Recentemente, Knorr *et. al* [KN97] notou que pode ser interessante para algumas aplicações a busca por *outliers*, ou seja, encontrar elementos em um conjunto que não obedeçam um padrão do conjunto ao qual pertencem. Uma interessante citação de [CTF03] é "*outliers* são semanticamente mais significantes (eles são *outliers*)".

Em organizações, estima-se que 5% dos registros de banco de dados são erros [Orr98]. Com isto estas organizações têm prejuízos anuais enormes para detectá-los. Para corrigí-los, existe um processo conhecido como limpeza de dados (*data cleansing*). Maletic e Marcus [MM00] apresentam uma revisão de métodos para o problema de limpeza de dados, sob a perspectiva de identificação de erros. Um dos métodos explorados pelos autores é a detecção de *outliers*.

Outra possível aplicação consiste na identificação de erros em textos. Miller e Myers [MM01] acoplaram ao sistema LAPIS (*Lightweigh Architecture for Processing Information Structure* – um editor de textos desenvolvido para navegação e edição de textos semi-estruturados) um algoritmo para busca de *outliers*. Este algoritmo procura por *matches* não usuais para o padrão do texto. Ele é útil para chamar a atenção em situações nas quais o julgamento humano é necessário.

Uma aplicação de detecção de *outliers* espaço-temporais é apresentada por Shekhar *et.al* [SLZ01]. Foi construído um banco de dados que armazena dados de sensores de um complexo de rodovias. Foram encontrados vários *outliers*, dentre eles estações com volume de tráfego muito diferente de estações vizinhas e faixas do dia nas quais a diferença de volume de tráfego é grande. Sabendo-se destas diferenças, é possível efetuar o planejamento do tráfego específico para cada área.

A maioria das propostas para detecção de *outliers*, em mineração de dados, é baseada na existência de uma função de distância entre os elementos do conjunto [Kno02, BKNS99, RRS00]. Entretanto, os atuais bancos de dados armazenam dados categóricos que não possuem uma função de distância implícita definida. Após testes realizados neste trabalho com dados reais, notou-se que em algumas aplicações o uso de métodos de detecção de *outliers* baseados em uma função de distância pode ser inviável.

O objetivo desta dissertação é desenvolver um método para detecção de *outliers* em conjunto de dados com atributos categóricos.

A abordagem proposta consiste na utilização de modelos log-lineares como um padrão para os dados. É construída uma tabela de contingência (pode ser vista como um sumário dos dados) a partir do conjunto de dados de entrada. Para esta tabela são gerados modelos com uma probabilidade associada e o modelo com maior probabilidade é selecionado. É possível então verificar quais elementos diferem significantemente da tabela gerada pelo modelo selecionado. Como a probabilidade associada ao modelo deve ser alta, somente poucos elementos não irão satisfazê-lo. Estes elementos serão considerados *outliers*.

Para validar esta proposta foram realizados testes com dados reais da área de telecomunicações, gentilmente cedidos pela empresa CPqD Telecom & IT Solutions.

As principais contribuições desta dissertação são:

1. Proposta de uma nova abordagem, no âmbito de mineração de dados, para detecção de *outliers* em conjuntos de dados com atributos categóricos, que:
   - provê um modelo para os dados reais;
   - permite a interação com o usuário; e
   - retorna uma medida para os *outliers* e para o modelo selecionado.
2. Implementação de um protótipo de sistema de análise de dados categóricos, permitindo ao usuário:
   - selecionar e ajustar um modelo aos dados;
   - fazer testes estatísticos entre modelos; e
   - detectar *outliers*.

O restante desta dissertação está organizado da seguinte forma. O capítulo 2 apresenta os conceitos básicos ao entendimento da dissertação e os principais trabalhos relacionados. O capítulo 3 mostra o problema e detalha a proposta desta dissertação. O capítulo 4 descreve o protótipo FOCAD - *Finding Outliers in Categorical Data*, apresentando os principais algoritmos para sua implementação. No capítulo 5 são apresentados os resultados dos testes realizados. Por fim, o capítulo 6 apresenta as conclusões desta dissertação e propõe alguns trabalhos futuros.

Capítulo 2

Conceitos Básicos e Trabalhos Relacionados

Este capítulo apresenta os conceitos básicos para o entendimento do restante da dissertação e os trabalhos relacionados a ela. A seção 2.1 aborda conceitos de mineração de dados. Na seção 2.3 é dada a definição de *outliers* e também é feita uma pequena discussão sobre o assunto. A seção 2.4 traz definições básicas sobre grafos, necessárias para o entendimento do capítulo 3. A seção 2.5 apresenta os principais trabalhos relacionados a esta dissertação, e, por fim, a seção 2.6 traz o resumo deste capítulo.

2.1 Dado e Informação

As definições de dado e informação são muito importantes para se entender mineração de dados. O principal objetivo é mostrar as diferenças entre dado e informação para que se possa definir mineração de dados da melhor forma possível. A seguir são apresentadas as definições de dado e informação baseadas no artigo de Setzer [Set01].

- **Dado:** qualquer pessoa está diretamente envolvida com dados, seja através de uma planilha eletrônica, de seu extrato bancário ou até mesmo por meio de um jornal. Setzer define *dado* como uma *seqüência de símbolos quantificados ou quantificáveis.* De acordo com esta definição, qualquer texto ou número é um dado; imagem, som e vídeo também são dados, pois é possível quantificá-los. Pode-se dizer então que dado é uma entidade matemática e puramente sintática, podendo ser manipulada por uma máquina.

- **Informação:** Setzer caracteriza *informação* como uma *abstração informal* (ou seja, não pode ser formalizada por meio de uma teoria lógica ou matemática) que está na

mente de alguém, representando algo significativo para a pessoa. Como o próprio Setzer aponta, esta é apenas uma caracterização e não uma definição, pois as palavras "algo" e "significativo" não estão bem definidas. Uma distinção fundamental entre dado e informação é que o primeiro é puramente sintático e o segundo necessariamente contém *semântica*. Informação não pode ser armazenada e nem manipulada diretamente em uma máquina, apenas sua representação sob a forma de dados. Um texto em chinês não transmite nenhuma informação para alguém que não entenda este idioma, mas ele é um dado. Para quem é fluente em chinês, o mesmo texto irá transmitir informação.

2.2 Mineração de Dados

Os termos mineração de dados (do inglês *data mining*) e KDD (*Knowledge Discovery in Databases*) são usados como sinônimos nesta dissertação. O objetivo de um algoritmo para mineração de dados é transformar um grande conjunto de dados em outro conjunto mais simples, de forma a facilitar o entendimento do usuário sobre o conjunto, permitindo que ele obtenha informação. A informação adquirida deve ser válida, preferencialmente nova e útil. Fayyad [FPSS96] definiu KDD como "um processo não trivial de identificar padrões válidos, novos, potencialmente úteis e compreensíveis". Ele é um processo interativo e iterativo, consistindo dos seguintes passos:

Entendimento do domínio do problema: é importante que o usuário entenda o domínio do problema; a origem dos dados, ou seja, como e porque foram gerados. Todo o restante do processo irá depender deste conhecimento inicial.

Pré-processamento: consiste em selecionar os dados, manipular campos nulos e em branco, aplicar transformações e normalizações no conjunto.

Escolha e execução de um método de mineração de dados: consiste em decidir qual o melhor método para tratar o problema. Exemplos de métodos são: regras de associação [AMS+95], *clustering* [ZRL96, NH94, EKSX96], classificação e detecção de *outliers*.

Interpretação dos resultados: nesta etapa pode ser notado que seja necessário repetir todo o processo, desde o estudo do domínio do problema até a escolha e métodos de mineração de dados.

Consolidação do processo: colocar os resultados em uso prático.

A saída da maioria dos métodos de mineração de dados são padrões para o usuário interpretar, ou seja, o usuário deve ser capaz de obter informação. Porém, a geração de um número grande de padrões pode dificultar sua interpretação. Portanto, um bom método deve prover medidas de interesse dos padrões gerados ao usuário: simplicidade, certeza, utilidade e novidade [HK01]. A simplicidade de um padrão leva a um entendimento mais fácil pelo usuário, o que possibilita sua validação e uso mais rápidos. Medidas de certeza devem ser associadas a cada padrão descoberto. Medidas de utilidade definem o interesse do usuário por um novo padrão. Um padrão deve ser novo para ser interessante; um método que retorne vários padrões com o mesmo significado semântico somente dificulta o processo. Todas estas medidas devem ser parametrizadas, permitindo ao usuário interagir com o sistema.

Métodos de mineração de dados devem ser eficientes e escaláveis, pois a quantidade de dados é enorme. Outra característica importante é a possibilidade de representação dos padrões encontrados sob várias formas: regras, árvores, tabelas, gráficos, etc.

2.3 *Outliers*

Outliers nem sempre são bem definidos e várias definições ou caracterizações podem parecer vagas em um primeiro momento. Nesta dissertação tentou-se definir *outliers* (definição 2.1) de uma forma precisa e que abrangesse várias outras definições encontradas na literatura.

Definição 2.1 *Um* outlier *é* um *elemento que desvia de* um *padrão do conjunto de dados ao qual ele pertence.*

Apesar da definição 2.1 ser simples, algumas considerações devem ser feitas. Em primeiro lugar, um *outlier* é sempre um elemento de um conjunto, podendo existir outros *outliers* no mesmo conjunto. Um elemento é dito *outlier* sempre em relação a um *padrão.* Logo, um elemento pode ser chamado *outlier* em relação ao padrão $\mathcal{A}$ e não ser um *outlier* em relação ao padrão $\mathcal{B}$.

A figura 2.1 mostra três exemplos de conjuntos. Em cada um deles é possível buscar por um padrão para elementos do conjunto e então encontrar os *outliers*. A figura 2.1(a) mostra dois aglomerados de pontos e dois pontos distantes dos demais, caracterizando-os como *outliers* (veja a seta indicando). A figura 2.1(b) mostra outro conjunto formado pelos símbolos $\oplus$ e $\odot$ intercalados formando uma reta. O símbolo $\odot$ indicado na figura 2.1(b) por uma seta é um *outlier*, pois, conforme o padrão, em seu lugar deveria estar o símbolo $\oplus$. Já a figura 2.1(c) mostra um conjunto de pontos no qual não é possível definir, ou mesmo notar um padrão para os pontos. Logo, não é possível definir nenhum elemento como *outlier* do conjunto.

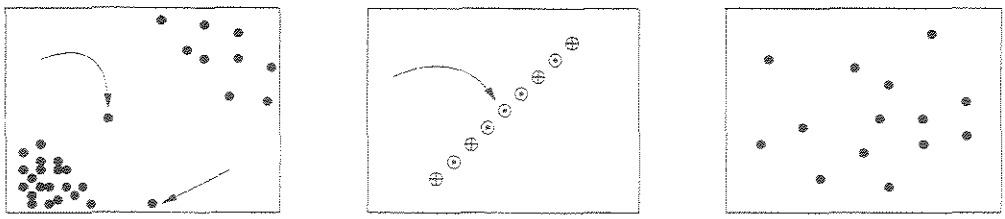


Figura 2.1: Exemplo de *outliers*.

Outliers sempre ocorrem em pequenas proporções no conjunto ao qual pertencem, pois se vários elementos fossem *outliers*, seria impossível caracterizar um padrão. Se um método qualquer para detecção de *outliers* retornar vários elementos, com certeza ele não é um bom método ou seus parâmetros não foram bem escolhidos.

A maioria dos métodos propostos para detecção de *outliers* leva em consideração a distância entre os objetos [Kno02, BKNS99, RRS00]. Talvez isto tenha ocorrido pois muitos deles foram baseados em métodos de *clustering*. Entretanto, além deste tipo de *outliers*, podem existir outros tipos, nos quais o padrão não seja a distância entre os objetos, mas a forma ou a disposição dos objetos. Regras de associação [AMS+95] (com confiança e suporte grandes) são outro exemplo de padrão, pois a maioria dos registros obedecem à regra. Logo, dada uma regra de associação, os registros que não a obedecem são considerados *outliers*. Como nota-se, o padrão do conjunto é muito importante e não pode ser esquecido quando pretende-se desenvolver um método para detecção de *outliers*.

A seguir é feita uma comparação da definição 2.1 com três importantes definições de *outliers* encontradas na literatura. Barnett e Lewis [BL84] definem um *outlier* em um conjunto de dados como "uma observação (ou um conjunto de observações) que parece ser inconsistente com o restante daquele conjunto de dados". Como os próprios Barnett e Lewis [BL84] destacaram, as palavras "parece ser inconsistente"são cruciais. Entende-se que o conjunto de dados possui um padrão e os elementos considerados *outliers* não obedecem ao padrão. Hawkins [Haw80] caracteriza um *outlier* de um modo intuitivo, como "um outlier é um objeto amostrado que desvia muito dos demais causando dúvidas quanto aos mecanismos de geração".

A definição de Barnett e Lewis [BL84] é subjetiva, principalmente pela palavra "parece". Já Knorr [Kno02] definiu *outlier* como "um objeto O em um conjunto de dados T é um $DB(p, D)$-outlier se uma fração mínima p de objetos em T possuem uma distância maior do que a distância D de O". Esta definição e o trabalho de Knorr serão mais detalhados na seção 2.5.2. Para entender melhor esta definição, veja a figura 2.2. Do total

de 10 elementos no conjunto T, somente 6 estão na vizinhança de O, formada pelo raio D. O elemento O é considerado um *outlier* somente se $p > 0.6$, ou seja, O está em uma região com poucos elementos por perto. Os objetos considerados *outliers* pela definição de Knorr são aqueles que estão em regiões menos densas (dados os parâmetros p e D). Logo, o padrão novamente é a aglomeração dos dados e os objetos considerados *outliers* são aqueles que não obedecem ao padrão.

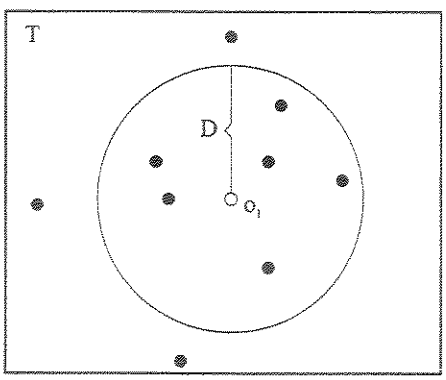


Figura 2.2: Exemplo de $DB(p, D)$-outlier

As causas de ocorrência de *outliers* foram agrupadas em três categorias por [BL84]:

1. Variedade inerente da população: os *outliers* são elementos que pertencem à população.

2. Erros de medição: ocorre na coleta dos dados. Pode ser causado por erros humanos, como a digitação de dados incorretos ou mesmo por erros de máquinas, como sensores.

3. Erros de execução: ocorrem quando os dados são adquiridos através de amostragem de mais de uma população.

Barnett e Lewis [BL84] adotaram uma abordagem estatística para agrupar as causas de ocorrência de *outliers* durante a amostragem dos dados. Contudo, em mineração de dados várias destas causas são interessantes. Quando os *outliers* pertencem à população, eles podem indicar novos horizontes, tanto para pesquisas científicas (por exemplo novos medicamentos ou tratamentos) como no meio comercial (por exemplo novos mercados ou a degradação de bons mercados). No caso da ocorrência de erros, eles podem ser identificados em processos humanos ou mesmo de *software*, possibilitando a correção do processo.

2.4 Grafos

Nesta seção serão apresentados os conceitos básicos sobre grafos necessários à compreensão de alguns tópicos desta dissertação. Eles serão referenciados no capítulo 3 e na seção 4.3.2.

Um grafo $G = (V, E)$ é formado pelos conjuntos $V = \{v_1, \ldots, v_n\}$ de vértices e $E = \{(u, v) | u, v \in V\}$ de arestas. Uma aresta partindo de um vértice para ele mesmo é chamada de *laço*. Um grafo *simples* não possui laços e não possui arestas idênticas. Um grafo *não direcionado* é um grafo $G = (V, E)$ onde E é um conjunto de pares *não ordenados*, ou seja, as arestas (u, v) e (v, u), $u, v \in V$ são consideradas a mesma.

Um *caminho* de comprimento k entre dois vértices $u, v \in V$ é uma seqüência $v_0, ..., v_k \in V$ com $\{u, v\} = \{v_0, v_k\}$ e $\{v_{i-1}, v_i\} \in E$ para $1 \leq i \leq k$. O *comprimento* do caminho é o número de arestas dele. O caminho contém os vértices $v_0, \ldots, v_k$ e as arestas $(v_0, v_1), \ldots, (v_{k-1}, v_k)$.

Dois subconjuntos $A, B \subseteq V$ são *separados* por $C \subseteq V$ se todo caminho de $a \in A$ para $b \in B$ contém um vértice $c \in C$. Um *ciclo* é um caminho formado pelos vértices $v_0, \ldots, v_k$ onde $v_0 = v_k$ e $k \geq 3$. O comprimento do ciclo é dado pelo comprimento do caminho.

Um grafo $G' = (V', E')$ é um *subgrafo* de $G = (V, E)$ (denotado por $G' \subseteq G$) se $V' \subseteq V$ e $E' \subseteq E$. Se além disso, E' possuir toda aresta (u, v) de E tal que u e $v \in V'$, então G' é o *subgrafo induzido* de G pelo subconjunto de vértices V'. Diz-se então que V' *induz* G' de G.

Um grafo é *completo* quando existe pelo menos uma aresta entre cada par de seus vértices. Um *clique* de G é um subconjunto $C \subseteq V$, tal que C induz um subgrafo completo de G. Um clique C é um *clique maximal* se não existe nenhum outro clique C' tal que $C \subset C'$.

Em um grafo não direcionado, se existe uma aresta $(u, v) \in E$ dize-se que os vértices u e v são *adjacentes* ou *vizinhos*, denotado por δ. Logo, $v \in \delta(u)$ e $u \in \delta(v)$. Um vértice é chamado *simplicial* se sua adjacência é um clique.

Definição 2.2 ***(Decomposição de um grafo)*** *Dado um grafo $G = (V, E)$, dois subconjuntos $A, B \subseteq V$ formam uma* decomposição *do grafo G, se $A \cup B = V$, $A \backslash B \neq \emptyset$, $B \backslash A \neq \emptyset$, $A \backslash B$ e $B \backslash A$ são separados por $A \cap B$ no grafo, e $A \cap B$ é um subconjunto completo.*

Em outras palavras, dado o grafo $G = (V, E)$, dois subgrafos de vértices A e B formam uma decomposição do grafo G se $A \cup B = V$ e a interseção entre A e B é um clique do grafo G. O grafo é então *decomposto* em dois subgrafos.

Definição 2.3 ***(Grafo decomponível)*** *Se um grafo pode ser decomposto recursivamente até que todos os subgrafos sejam completos, então o grafo é dito* decomponível.

Grafos *cordais* ou *triangulados* são grafos em que todos os ciclos de comprimento maior que 3 possuem uma aresta entre dois vértices não adjacentes, chamada de *corda*. Grafos decomponíveis são grafos cordais [Lau96].

Seja um grafo $G = (V, E)$ e $\pi = \{v_1, \ldots, v_n\}$ uma ordenação de seus vértices. Diz-se que π é uma *ordem de eliminação perfeita* se cada v_i é um vértice simplicial de um grafo induzido por $\{v_i, \ldots, v_n\}$ de G.

2.5 Trabalhos Relacionados

A detecção de *outliers* foi amplamente estudada pela comunidade estatística. Contudo, existem algumas limitações para se aplicá-la diretamente em mineração de dados. Knorr e Ng [KN97] propuseram o conceito de *outliers* baseados em distância, o que foi seguido por vários trabalhos nesta mesma linha, como [EH99, DSZ99, RRS00, BKNS00, Kno02]. Como nos bancos de dados atuais vários atributos são categóricos, é difícil o uso destes métodos, pois é necessária a existência de uma função de distância entre os elementos do conjunto de dados. Outros trabalhos, como [SAM98, FF01] lidam com cubos OLAP (*On-Line Analytical Processing*) procurando por células *anômalas* ou excepcionais. Alguns trabalhos de *clustering* de dados categóricos e recuperação de informação usam funções de similaridade para este tipo de dados.

As próximas seções detalham estes trabalhos.

2.5.1 Abordagem Estatística

Em estatística, existem duas formas de se tratar *outliers*. A primeira é a **acomodação** dos *outliers*, que consiste em criar métodos estatísticos que permitam inferências válidas da população mesmo que ocorram *outliers* na amostra dos dados. A segunda abordagem consiste na criação de um teste estatístico com o objetivo de identificar os *outliers*, conhecidos como **testes de discordância**.

Em [BL84] são apresentados vários testes de discordância. Cada teste foi desenvolvido para uma situação diferente, dependendo de: (1) distribuição dos dados; (2) se os parâmetros são conhecidos (média, variância); (3) número esperado de *outliers*; (4) tipos de *outliers* (superior, inferior).

Como exemplo de teste de discordância, considere uma amostra $x_{(1)}, \ldots, x_{(n)}$ seguindo a distribuição normal, com média μ e variância δ^2 desconhecidos. Um teste para um único *outlier* apresentado em [BL84] é:

$$T_{N1} = \frac{x_{(n)} - \overline{x}}{s}$$

na qual $\overline{x}$ e s são a média e o desvio padrão da amostra. T_{N1} é o valor padronizado para o elemento $x_{(n)}$. Logo, basta verificar a probabilidade de $x_{(n)}$ ocorrer na distribuição normal.

A maioria dos testes de discordância citados em [BL84] são para variáveis de uma dimensão e específicos para cada caso de *outliers*. Contudo, em mineração de dados, os parâmetros (número de *outliers*, distribuição dos dados, etc) não são conhecidos para muitas situações. Então, são necessários vários testes para encontrar uma distribuição de probabilidade que melhor se ajuste aos dados, tornando lento o processo. O tamanho do conjunto de dados em mineração de dados é outro problema, pois estes testes de discordância foram desenvolvidos para pequenos conjuntos e nem sempre são escaláveis.

Além dos testes para dados numéricos, existem também alguns trabalhos para tratar dados categóricos, organizados em tabelas de contingência. Barnett e Lewis [BL84] apresentam alguns métodos para tratar tabelas de contingência de duas dimensões, buscando por células excepcionais.

Em um modelo de independência pode-se encontrar variações nas frequências esperadas de uma célula para as frequências observadas. Se a frequência observada de uma célula exibe uma discrepância da sua frequência esperada, então a célula é considerada uma célula excepcional. O modelo de probabilidade é baseado em uma distribuição multinomial. Para cada célula é calculado o resíduo ajustado z_i. A célula que possuir o máximo $|z_i|$ é considerada uma célula *outlying*.

Os trabalhos apresentados em [BL84] são para tabelas de contingência de duas dimensões. Nesta dissertação trabalhou-se com um número de dimensões qualquer e foi usado outro teste para considerar uma célula *outlying*, apesar do princípio básico de ambos ser o mesmo: verificar a diferença entre as frequências esperadas em relação às observadas.

2.5.2 *Outliers* baseados em distância

A abordagem estatística tradicional encontra alguns limitantes para ser aplicada diretamente em mineração de dados, principalmente o tamanho dos conjuntos de dados. Em 1997, Knorr e Ng [KN97] propuseram o conceito de **DB(p, D)-outlier**[1] baseado na distância entre os elementos do conjunto de dados. Após este trabalho vários outros surgiram, usando o mesmo princípio da distância.

[1] Em [KN97], os *outliers* foram chamados de UO-Outlier (*unified outlier*) e em [KN98] o nome foi alterado para $DB(p, D)$-outlier (*distance-based outlier*).

Definição de Knorr *et.al*

Definição 2.4 *(DB(p, D)-outlier) Um objeto O em um conjunto de dados T é um $DB(p, D)$-outlier se uma fração mínima p de objetos em T possui uma distância maior do que a distância D de O.*

Os autores mostram que esta definição é equivalente a vários testes de discordância encontrados em [BL84] e ainda é aplicável a valores multidimensionais. A seção 2.3 possui um exemplo de DB-outliers. Knorr *et.al* [KN98] apresentam três algoritmos para detecção de *outliers*:

- O primeiro algoritmo usa uma estrutura de índice espacial, como R^*, para a pesquisa dos vizinhos de um objeto O dentro de um raio D. Seja M o número máximo de objetos dentro da D-vizinhança de um *outlier*. Assim, se $M + 1$ objetos forem encontrados na vizinhança de O, então O não é um *outlier*. A complexidade temporal de pior caso deste algoritmo é $O(kN^2)$, onde k é o número de dimensões e N o número de objetos.

- no primeiro algoritmo não foi considerado o tempo de construção do índice. O algoritmo *Nested Loop* foi proposto usando-se uma estrutura de dois laços aninhados: para cada objeto O, calcula-se sua distância aos demais objetos do conjunto. A complexidade temporal de pior caso continua $O(kN^2)$.

- O último algoritmo é baseado em uma estrutura de células. Cada elemento está em uma célula e o cálculo de distâncias é feito apenas para uma pequena quantidade de elementos de algumas células candidatas. O algoritmo tem complexidade $O(c^k + N)$, onde c é uma constante, k o número de dimensões dos objetos e N o número de objetos. A vantagem deste algoritmo é que ele não calcula a distância para todos os objetos. No entanto, para conjuntos de dados com $k \geq 5$, o algoritmo torna-se ineficiente.

Outliers fortes e fracos

Em [KN99] foi introduzido o conceito de *outliers* fortes e fracos. A principal necessidade desta diferenciação é poder "justificar" um *outlier*, ou seja, porque um elemento foi considerado um *outlier*. Assim, o usuário será capaz de: (i) avaliar a credibilidade dos *outliers* identificados; e (ii) prover conhecimento sobre os dados.

Definição 2.5 *Suponha que p é um* outlier *em algum espaço A_p. p é um* outlier ***não trivial*** *em A_p se p não é um* outlier *em um sub-espaço $\mathcal{B} \subset A_p$.*

Definição 2.6 *Seja $\mathcal{A}_p$ um espaço contendo um ou mais* outliers. *Se não existirem* outliers *em qualquer sub-espaço $\mathcal{B} \subset A_p$, então todos os* outliers *p em $\mathcal{A}_p$ são chamados* **outliers *fortes***.

Definição 2.7 *Suponha que p é um* outlier *não trivial em $\mathcal{A}_p$. Se p não é um* outlier *forte, então p é um* **outlier *fraco***.

Seja p um elemento que foi identificado como excepcional relativo a um espaço $\mathcal{A}_p$. A partir das definições acima é possível responder a seguinte questão: Qual é o menor conjunto de atributos que explicam por que p é um *outlier*?

Definição de Ramaswamy

Seja o_1 um elemento de um conjunto de dados. Seja $D^k(o_1)$ a distância de o_1 a seu k-ésimo vizinho mais próximo. Ramaswamy [RRS00] definiu *outliers* como:

Definição 2.8 *Dado um conjunto de dados com N elementos, os parâmetros n e k, um elemento o_1 é um D_n^k* outlier *se não existirem mais do que $n-1$ outros elementos o_1' tal que $D^k(o_1') > D^k(o_1)$.*

Em outras palavras, os elementos serão ordenados de acordo com $D^k(o_1)$ e os n maiores serão considerados *outliers*. A definição 2.8 é equivalente à definição 2.4 [RRS00]. A escolha dos parâmetos k e n deve ser feita considerando o tamanho do conjunto de dados. Em [RRS00], o autor cita que o parâmetro k não influi significantemente no tempo total do algoritmo. Já n deve ser apenas uma fração pequena do tamanho total do conjunto de dados.

Outliers baseados em densidade

As abordagens para detecção de *outliers* baseadas em distância não conseguem detectar todos os tipos de *outliers*. Outro tipo de *outlier* que pode ocorrer nos bancos de dados reais é a existência de *clusters* de dados em diferentes densidades, conforme apontado por Breuning *et.al* [BKNS00]. A figura 2.3 mostra os *clusters* C_1 e C_2, com diferentes densidades. Intuitivamente, os *outliers* são os pontos o_1 e o_2. Em uma abordagem baseada somente na distância dos objetos, o elemento o_2 é considerado um *outlier*, porém o_1 não será. Se o_1 for considerado um *outlier*, então todos os elementos do *cluster* C_2 também serão.

Para tratar este tipo de *outliers*, Breuning *et. al* [BKNS00] propuseram um método baseado na densidade da vizinhança de um elemento. É atribuído para cada elemento um fator chamado LOF (*Local Outlier Factor*), que irá medir o quão isolado um elemento

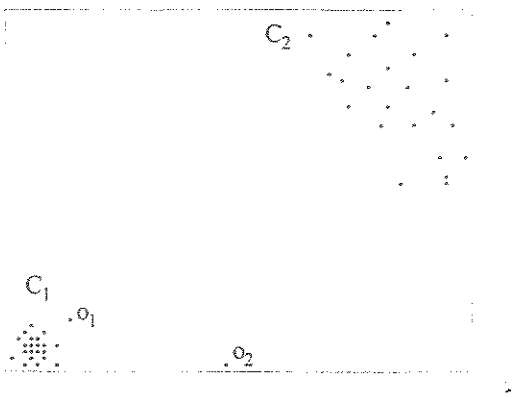


Figura 2.3: Exemplo de um conjunto mostrando a diferença de densidade de dois *clusters*, e dois *outliers*: o_1 e o_2 (retirado de [BKNS00]).

está de sua vizinhança. O tamanho vizinhança é especificado pelo usuário, através do parâmetro $MinPts$. Para elementos que estão dentro de algum *cluster*, o valor de LOF será 1. Para os elementos que estão nas bordas ou fora do *cluster*, o valor será maior que 1, limitados por uma expressão em função de $MinPts$.

Comentários

A definição de Knorr (definição 2.4) é intuitiva, simples e ainda aplicável a dados multi-dimensionais. Porém, é requerido do usuário especificar a distância D que pode ser difícil de determinar (os autores sugerem tentativa e erro, o que pode requerer várias iterações). Isto pode torná-la impraticável em casos práticos.

A definição de Ramaswamy (definição 2.8) requer do usuário a especificação dos parâmetros k e n, que é mais simples do que especificar o valor da distância D. O valor de n deve ser uma pequena fração de N, pois *outliers* constituem apenas um subconjunto pequeno dos dados. O valor de k é escolhido por tentativa e erro, mas o resultado final sofrerá pequenas variações.

Transformação Donoho-Stahel

Segundo Knorr *et. al* [KNZ01], o simples mapeamento de uma tupla $t = (a_1, \ldots, a_n)$ para um ponto espacial $p = (a_1, \ldots, a_n)$ não é uma boa transformação em virtude da diferença de escala, correlação, variabilidade e *outliers*. Eles propuseram o uso da transformação robusta Donoho-Stahel, que possui a seguinte propriedade: enquanto inapropriada no espaço original, a função de distância euclidiana torna-se razoável no espaço transformado.

Como exemplo, considere um conjunto com os seguintes atributos: pressão sangüínea (100-160mm, média $\mu = 120$mm); temperatura corporal ($\mu = 37°C$, com pequeno desvio padrão); e idade (20-50 anos). Existe diferença de escala e unidade dos atributos, diferença de variabilidade e os atributos podem ser correlacionados. A diferença de escala, unidade e a variabilidade podem ser tratadas com normalização ou padronização dos dados. Porém, a correlação entre os atributos também deve ser considerada.

A proposta de Knorr *et. al* [KNZ01] foi o uso da transformação Donoho-Stahel. Para tratar a correlação entre atributos categóricos, esta dissertação propôs o uso de modelos log-lineares, pois eles consideram as interações entre os atributos.

2.5.3 Outras abordagens em mineração de dados

Alguns algoritmos de *clustering*, tais como CLARANS [NH94], DBSCAN [EKSX96] e BIRCH [ZRL96] foram desenvolvidos com a capacidade de manipulação de *outliers* (exceções e ruídos). Como seu principal objetivo é encontrar *clusters*, sua noção de *outlier* é definida indiretamente ao longo da definição de *clusters*. Os algoritmos desenvolvidos possuem a finalidade de acomodação de *outliers*, ou seja, eles conseguem produzir *clusters* do conjunto mesmo quando existem *outliers*.

Arning *et.al* [AAR96] desenvolveram um método para detecção de desvio baseado no seguinte princípio: depois de uma série de dados similares, um elemento que cause distúrbio na série é considerado uma exceção. Uma função de similaridade irá dizer o quanto um novo item é "diferente" da série. Foi apresentado um algoritmo com tempo linear para o encontrar as exceções. O principal problema, apresentado pelos próprios autores, é que a qualidade do resultado irá depender da função de similaridade usada. A conclusão foi que é muito difícil, senão impossível, existir uma função de similaridade universal para qualquer banco de dados.

Nos testes realizados em [AAR96] existiam muitos atributos categóricos que foram tratados com uma função de similaridade. Tratar atributos categóricos com funções de similaridade é difícil, pois é necessária a criação de uma função que capture as interações entre os atributos.

2.5.4 Detecção de anomalias em cubos OLAP

Sarawagi *et.al* [SAM98] desenvolveram um método para buscar anomalias (exceções) em cubos OLAP. OLAP se caracteriza por operações de sintetização, consolidação e visualização de dados em múltiplas dimensões. Um cubo de dados consiste de atributos agrupados para formar uma chave. Exemplos de atributos são: nomes de produtos, sexo, períodos de tempo.

Em [SAM98] são apontadas duas formas de se navegar em um cubo. (1) Exploração dirigida por hipóteses: o analista interativamente explora o cubo, procurando por regiões anômalas, pois elas podem mostrar áreas com problemas ou novas oportunidades. A principal deficiência desta abordagem é o tempo necessário para se navegar pelo cubo que, em geral, possui várias dimensões com muitos níveis cada uma. Podem haver anomalias escondidas em níveis internos, na junção de quaisquer dimensões. A proposta de Sarawagi *et.al* [SAM98] para tratar esta deficiência é destacar para o usuário as células com comportamento anômalo, em qualquer nível do cubo. Para isto, são calculados valores baseados em um modelo estatístico para cada célula do cubo e é verificado se este valor é significantemente diferente do valor real da célula.

Para a escolha do modelo estatístico foram considerados os seguintes aspectos: (a) consideração da variação e padrões em todas as dimensões; (b) as exceções em todos os níveis, não somente nos mais detalhados; (c) o usuário deve ser capaz de interpretar os resultados sem a necessidade de um conhecimento estatístico avançado; (d) o processamento deve ser eficiente e escalável.

O modelo estatístico usado é um modelo log-linear, obtido do modelo saturado retirando-se o termo com todas as variáveis (veja a seção 3.4 para uma explicação sobre modelos log-lineares). O cálculo das exceções é feito com base nos resíduos padronizados, obtidos através da equação 2.1, onde $y_{i_1 i_2 \ldots i_n}$ é o valor real da célula, $\hat{y}_{i_1 i_2 \ldots i_n}$ é o valor esperado para a célula e $\sigma_{i_1 i_2 \ldots i_n}$ o valor esperado para o desvio padrão.

$$s_{i_1 i_2 \ldots i_n} = \frac{|y_{i_1 i_2 \ldots i_n} - \hat{y}_{i_1 i_2 \ldots i_n}|}{\sigma_{i_1 i_2 \ldots i_n}} \tag{2.1}$$

Nesta dissertação usou-se o modelo log-linear, porém não apenas um modelo específico. Com a restrição imposta em [SAM98], algumas interações entre variáveis talvez não existam e são consideradas no modelo, pois ele é fixo. O cálculo do desvio padrão σ nesta dissertação não foi necessário, pois foi utilizado outro teste estatístico.

Outros trabalhos foram propostos com o mesmo objetivo de encontrar anomalias em cubos OLAP, como os trabalhos de Fabris e Freitas [FF01] e Chen [CP99].

Chen e Petrounias [CP99] usou a mesma abordagem de Sarawagi *et.al* [SAM98]. Contudo, em seu trabalho, ele notou que o modelo log-linear não produz bons resultados quando o cubo é montado com a função `average()` e não com a função `count()`. Para tratar este tipo de dado, ele propôs o uso de um modelo linear. Os algoritmos permaneceram os mesmos, alterando apenas o cálculo de parâmetros do modelo.

2.6 Resumo

Este capítulo apresentou os conceitos básicos, iniciando com as definições de dado e informação e posteriormente mineração de dados. *Outliers* foram definidos nesta dissertação como "*um* elemento que desvia de *um* padrão do conjunto de dados ao qual ele pertence". Foram apresentados exemplos de *outliers* e feita uma comparação com duas importantes definições de *outliers* encontradas na literatura. Na seção 2.5 os trabalhos relacionados foram apresentados.

O próximo capítulo define o problema tratado nesta dissertação e a solução proposta.

Capítulo 3

Detecção de *Outliers*

Outliers são elementos que desviam de *um* padrão do conjunto de dados ao qual eles pertencem. Muito trabalho tem sido feito em mineração de dados para detecção de *outliers* em conjunto de dados numéricos, para os quais existem funções de distância bem definidas. Todavia, os bancos de dados, científicos e comerciais, possuem vários atributos categóricos ou discretos que também devem ser tratados. Muitas vezes o uso dos métodos tradicionais para detecção de *outliers* não podem ser utilizados, pois para este tipo de dado não existe uma função de distância definida.

Esta dissertação propõe o uso de modelos log-lineares para detecção de *outliers* em conjuntos de dados categóricos. O restante deste capítulo está dividido da seguinte forma: a seção 3.1 apresenta a descrição do problema tratado nesta dissertação; na seção 3.2 é descrito o método para tratar o problema; as seções 3.3, 3.4 e 3.5 detalham o método proposto nesta dissertação.

3.1 Problema

Esta dissertação trata do problema de detecção de *outliers* em bancos de dados com atributos categóricos. Dados categóricos são dados que não possuem ordenação e pertencem a um domínio finito. Dois exemplos simples e comuns nos bancos de dados atuais são sexo (masculino, feminino), estado da federação (SP, MG, ...), dentre outros. A seguir é dada uma descrição formal do conjunto de dados com o qual esta dissertação trabalha e a definição do problema..

Enunciado 3.1 *Seja* $\mathcal{R} = \{A_1, A_2, \ldots, A_n\}$ *um esquema relacional onde* $A_i \in D_i$, $1 \leq i \leq n$ *e o domínio* D_i *é finito e não ordenado.* $|D_i|$ *indica a cardinalidade do domínio* D_i. *Esta dissertação trata do problema de detecção de* outliers *em uma relação de esquema* $\mathcal{R}$.

A tabela 3.1 apresenta um exemplo de uma relação com dados categóricos. Os atributos "Produto", "Região" e "Sexo" claramente possuem um domínio finito, dada uma aplicação particular. Eles também não possuem uma ordenação implícita, ou seja, considerando apenas o atributo, sem nenhuma informação adicional, não pode-se dizer que existe uma ordenação {Cerveja, Refrigerante A} ou {Refrigerante A, Cerveja}. Para o atributo "Região", pode-se pensar em ordená-lo de acordo com sua posição geográfica, produto interno bruto, ou outra forma qualquer. Como o significado deste atributo é apenas dizer em qual região o produto foi vendido, o uso de qualquer ordenação é subjetiva.

O método a ser desenvolvido deve prover ao usuário alguma ferramenta pela qual ele possa entender o motivo de um dado do banco de dados ser considerado um *outlier*. Isto pode ser obtido através de modelos ou padrões para os dados, valores numéricos indicando o "quanto" cada objeto é um *outlier* ou mesmo através de gráficos. Apesar de implícito, visto que esta dissertação está no âmbito de mineração de dados, a quantidade de dados com a qual lida-se é grande. Assim, um método para detecção de *outliers* deve ser capaz de manipular uma grande quantidade de dados de forma eficiente.

Quantidade	Produto	Região	Sexo
223	Cerveja	Sul	masculino
113	Refrigerante A	Norte	feminino
⋮	⋮	⋮	⋮

Tabela 3.1: Exemplo de um conjunto de dados categóricos no qual é feita a detecção de *outliers*.

3.2 Método

Nos métodos apresentados na seção 2.5.2 para detecção de *outliers* em dados numéricos, verifica-se a existência de um padrão para os dados, mesmo que não seja explicitamente dito. Com isto notou-se a necessidade da criação de um modelo para dados categóricos e a partir deste modelo fazer detecção de *outliers*. Com esta idéia o método apresentado a seguir foi desenvolvido.

O método desenvolvido nesta dissertação está dividido em 3 etapas: importação dos dados, criação do modelo e detecção dos *outliers*. A figura 3.1 mostra um esquema do método e a seguir é dada uma descrição sucinta de cada uma de suas etapas:

- **Importação dos dados:** nesta fase é criada uma tabela de contingência a partir dos dados de entrada. A tabela de contingência é definida formalmente na seção 3.3,

mas agora pode-se dizer que ela é um sumário da tabela de entrada, com o número de dimensões igual ao número de atributos de $\mathcal{R}$. Cada célula conterá o número de ocorrências da categoria conjunta de suas dimensões.

- **Criação do modelo:** tendo a tabela de contingência em mãos, o método prossegue com a criação de um modelo para ela. Este modelo é um padrão para os dados categóricos, como os *clusters* existentes nos métodos para dados numéricos. Foi utilizado o modelo log-linear, pois com ele é possível calcular o valor esperado para cada célula da tabela de contingência, possibilitando a verificação da diferença entre os valores reais e os valores gerados pelo modelo. É possível ainda entender os relacionamentos entre os atributos, tendo assim uma forma de entender o "porquê" dos *outliers*.

- **Detecção dos *outliers*:** como foi dito, com o modelo é possível calcular o valor esperado para cada célula. Logo, é possível verificar quais células diferem *significantemente* deste valor, ou seja, quais células possuem um valor real muito diferente do valor esperado para ela.

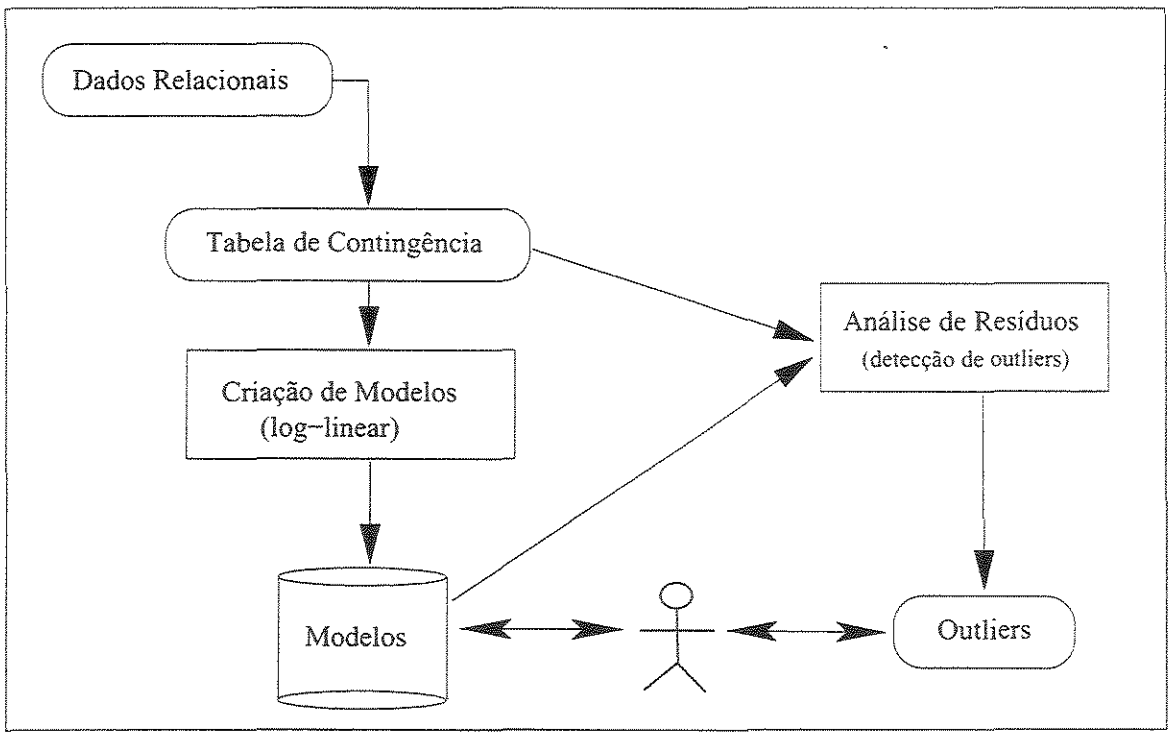


Figura 3.1: Método proposto para detecção de *outliers* em dados categóricos

Duas importantes características deste método são:

- Participação do usuário na criação dos modelos. Existe um método automático para a criação e seleção dos modelos, mas mesmo assim o usuário pode interferir neste processo. Deste modo é permitido a ele inserir seu conhecimento prévio sobre o domínio da aplicação no processo da criação do modelo log-linear.
- Com a criação de modelos e a posterior seleção de um modelo, o usuário poderá entender os relacionamentos entre cada atributo e assim entender os *outliers* retornados.

A próxima seção apresenta a definição de tabelas de contingência. Na seção 3.4 é apresentado o modelo log-linear e na seção 3.5 como será feita a detecção de *outliers*.

3.3 Tabela de Contingência

Nesta seção é apresentada a definição da tabela de contingência. No capítulo 4 serão apresentados os algoritmos para sua construção e acesso.

3.3.1 Definição e Notação

Uma tabela de contingência nada mais é do que uma forma de apresentar dados categóricos de forma organizada. Um exemplo é a tabela 3.2(b) que foi criada a partir da tabela 3.2(a). Cada célula da tabela 3.2(b) contém o número de ocorrências de um determinado registro da tabela 3.2(a). A seguir é dada a definição formal de tabela de contingência:

Definição 3.1 ***(Tabela de Contingência)*** *Sejam:*

- *um conjunto finito de variáveis* Δ*;*
- *um conjunto de índices* $i \in \mathcal{I} = \times_{\delta \in \Delta} \mathcal{I}_\delta$*, sendo* $\mathcal{I}_\delta$ *os níveis da variável* $\delta \in \Delta$*;*

Denota-se o valor contido na célula i *por* $N(i)$ *e por* N *o somatório de todas as células da tabela. A probabilidade de um objeto estar em uma célula* i *é denotado por* $p(i)$ *e seu estimador por* $\hat{p}(i)$*.*

Na tabela 3.2(b) o conjunto Δ é formado por {Produto, Região, Sexo} e os níveis da variável "Sexo" são {M, F}. O conjunto $\mathcal{I}$ é formado pelo produto cartesiano entre os níveis das três variáveis de Δ. A célula $i = \{Cerveja, Norte, M\}$ tem valor $N(i) = 501$ e $p(i) = 0.19$, pois $p(i) = N(i)/N$ e $N = 3359$. O estimador $\hat{p}(i)$ é determinado de acordo com o modelo *log-linear*.

Produto	Região	Sexo
Cerveja	Sul	masculino
Refrigerante	Norte	feminino
⋮	⋮	⋮

(a) Dados originais.

Produto	Sexo	Região			
		Norte	Sul	Leste	Oeste
Cerveja	M	501	210	201	240
	F	249	105	106	158
Refrigerante	M	376	160	149	179
	F	315	136	124	150
				Total	3359

(b) Tabela de Contingência

Tabela 3.2: Exemplo de uma tabela de contingência.

Definição 3.2 ***(Tabela marginal)*** *Seja $\mathcal{I}_\delta$ uma tabela de contingência de variáveis Δ; uma* ***tabela marginal*** *de $\mathcal{I}_\delta$ de variáveis $a \in \Delta$ é formada pela soma dos valores das variáveis em a sobre δ.*

A tabela 3.3 apresenta a tabela marginal formada a partir da tabela 3.2, sendo $a = \{\text{Produto, Sexo}\}$.

Produto	Sexo	
	M	F
Cerveja	1152	618
Refrigerante	864	725
	Total	3359

Tabela 3.3: Exemplo de uma tabela marginal.

3.4 Modelo Log-Linear

Nesta dissertação utilizou-se o modelo log-linear. Ele é uma expressão das freqüências esperadas das células da tabela de contingência em função de parâmetros representando o inter-relacionamento entre as variáveis categóricas. Neste modelo não se deseja conhecer o valor de uma variável a partir do conhecimento sobre os valores das demais variáveis, como em um modelo de regressão linear. O modelo log-linear apenas demonstra a interação entre as variáveis, por meio de dependências parciais e condicionais.

Seja o conjunto de variáveis $\Delta = (X_1, X_2, \ldots, X_k)$ e i um índice para uma célula da tabela de contingência formada por estas variáveis. l_a denota os níveis da variável a. A freqüência esperada (E) de uma célula é determinada pela combinação linear dos termos μ (média geométrica global) e λ (médias das freqüências marginais). Cada λ-termo representa os "efeitos" das variáveis no valor da célula, ou seja, o desvio da célula em relação à média μ. A forma geral do modelo log-linear é dada pela equação 3.1.

$$\log E_i(\Delta) = \mu + \sum_{a \in \Delta} \lambda_{l_a}(a) + \sum_{a,b \in \Delta} \lambda_{l_a l_b}(ab) + \sum_{a,b,c \in \Delta} \lambda_{l_a l_b l_c}(abc) + \ldots + \lambda_i(\Delta) \qquad (3.1)$$

Δ	Modelo Saturado
A, B	$\log E_{l_a l_b}(AB) = \mu + \lambda_{l_a}(A) + \lambda_{l_b}(B) + \lambda_{l_a l_b}(AB)$
A, B, C	$\log E_{l_a l_b l_c}(ABC) = \mu + \lambda_{l_a}(A) + \lambda_{l_b}(B) + \lambda_{l_c}(C) + \lambda_{l_a l_b}(AB) + \lambda_{l_a l_c}(AC) + \lambda_{l_b l_c}(BC) + \lambda_{l_a l_b l_c}(ABC)$

Tabela 3.4: Exemplos de modelos log-lineares de 2 e 3 variáveis.

A tabela 3.4 mostra a forma geral dos modelos log-lineares para duas e três variáveis. Pelas equações apresentadas, nota-se que as variáveis são combinadas em pares, triplas, até que se tenha todas no modelo. Para cada λ-termo, as variáveis estão entre parênteses e o subscrito indica os níveis delas, representando os "efeitos" das variáveis no valor da célula. Existe **interação** entre duas variáveis quando existe pelo menos um λ-termo que as contenha. Por exemplo, $\lambda_{1,2}(AB)$ indica a interação entre a variável A e B nos níveis 1 e 2 respectivamente.

Quando todos os λ-termos estão presentes no modelo, ele é chamado log-linear **saturado**. Este modelo possui um ajustamento perfeito aos dados, ou seja, com ele é possível reconstruir a tabela de contingência original. Porém, na prática ele é usado somente como modelo base para testar o ajustamento de outros modelos, já que ele se utiliza de todos os graus de liberdade possíveis. Comparando-se os λ-termos com vetores e matrizes, nota-se que o número de posições necessárias para o modelo log-linear saturado é igual ao número de células da tabela de contingência.

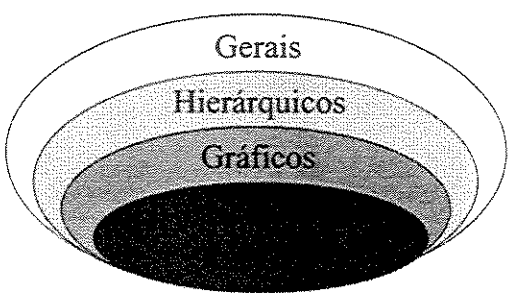


Figura 3.2: Classes de modelos log-lineares.

Para cada λ-termo igual a zero tem-se um modelo log-linear diferente. Em virtude do número de modelos ser exponencial, criou-se algumas classes de modelos. Cada classe é obtida na forma em restringir-se λ-termos iguais a zero. A figura 3.2 mostra as classes de modelos log-lineares que serão apresentadas. A classe "Gerais" na figura refere-se à classe de modelos log-lineares sem nenhuma imposição em restringir λ-termos a zero. A classe adotada nesta dissertação é a classe de modelos log-lineares decomponíveis. Apesar de ser mais restrita, ela possui várias características desejáveis em mineração de dados, como a possibilidade de interpretação do modelo através de dependência condicional e a fórmula direta para o cálculo das frequências esperadas. A seguir, cada uma destas classes será definida.

3.4.1 Modelos Log-Lineares Hierárquicos e Gráficos

Modelos log-lineares hierárquicos, ou simplesmente modelos hierárquicos, são uma classe de modelos log-lineares obtidos por impor uma condição em restringir λ-termos iguais a zero. Sua definição formal é:

Definição 3.3 ***(Modelos log-lineares hierárquicos)*** *Um modelo log-linear é* ***hierárquico*** *se, sempre que um λ-termo é zero então todos os λ-termos com mais variáveis contendo o mesmo conjunto de subscritos também são zero; ou seja, se $\lambda_a = 0$ então $\lambda_t = 0$ para todo $a \subseteq t$ [Whi90].*

Em outras palavras, sempre que um λ-termo aparece no modelo, todos os λ-termos contidos nele também devem pertencer ao modelo. Por exemplo, o modelo $\log E_{l_a l_b l_c} = \mu + \lambda_{l_a}(A) + \lambda_{l_b}(B) + \lambda_{l_a l_c}(AC) + \lambda_{l_a l_b}(AB)$ não é um modelo log-linear hierárquico pois o termo $\lambda_{l_a l_c}(AC)$ pertence ao modelo e o termo $\lambda_{l_c}(C)$ não pertence. Já o modelo apresentado a seguir é um modelo log-linear hierárquico.

$$\log E_{l_a l_b l_c} = \mu + \lambda_{l_a}(A) + \lambda_{l_b}(B) + \lambda_{l_c}(C) + \lambda_{l_a l_b}(AB) + \lambda_{l_a l_c}(AC) + \lambda_{l_b l_c}(BC) \quad (3.2)$$

Considerando a classe de modelos hierárquicos, os máximos λ-termos são chamados de **geradores** e o conjunto deles de **classe geradora** do modelo. É possível escrever um modelo log-linear hierárquico apenas com sua classe geradora. Como exemplo, o modelo da equação 3.2 pode ser reescrito de forma mais simples por sua classe geradora [AB][AC][BC]. Será utilizada esta notação até o fim desta dissertação.

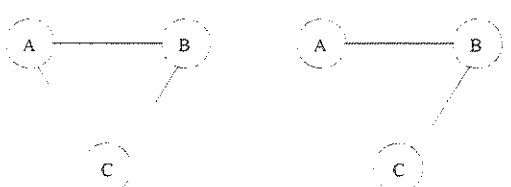
Outra forma de representação de um modelo log-linear é através de um grafo não direcionado $G = (V, E)$, onde o conjunto dos vértices V é o conjunto das variáveis Δ do modelo e as arestas são dadas pelas interações entre duas variáveis presentes nele. Este grafo é

chamado **grafo de interação**. A figura 3.3(a) mostra o grafo de interação do modelo [AB][AC][BC]. Conforme [Whi90], um grafo de interação é um grafo de **independência**.

O grafo de interação de um modelo hierárquico pode ser usado para ler as independências condicionais do modelo [Lau96]. Sempre que dois subconjuntos Δ_1 e Δ_2 são separados pelas variáveis de S neste grafo, então Δ_1 é independente condicionalmente de Δ_2, dadas as variáveis de S. Particularmente, diferentes componentes conexos do grafo de interação correspondem às variáveis mutuamente independentes. Porém, modelos hierárquicos podem ter o mesmo grafo de interação, como os modelos $\mathcal{M}_1 = [ABC]$ e $\mathcal{M}_2 = [AB][AC][BC]$.

Modelos log-lineares gráficos são uma sub-classe de modelos hierárquicos. De acordo com a proposição 7.3.1 de [Whi90], um modelo log-linear hierárquico é **gráfico** se e somente se seus geradores correspondem aos cliques do grafo de independência.

O modelo [AB][AC][BC] não é um modelo gráfico, pois, de acordo com seu grafo de independência, o termo [ABC] deveria pertencer ao modelo. Já o modelo [AB][BC], apresentado na figura 3.3(b), é um modelo gráfico. Uma grande vantagem desta classe é que os modelos dela possuem uma interpretação simples em termos de independência condicional.



(a) (b)

Figura 3.3: Exemplo de um grafo de independência.

3.4.2 Modelos Decomponíveis

Uma importante classe de modelos log-lineares são modelos gráficos cujo grafo de independência é um grafo decomponível, ou seja, um grafo cordal. Estes modelos são chamados modelos log-lineares **decomponíveis**.

Considerando-se os modelos log-lineares gerais, o espaço de busca é muito grande. Assim, o mais evidente benefício em se usar modelos decomponíveis é a redução do espaço de busca. Além disso, modelos decomponíveis podem ser interpretados em termos de independência condicional, o que nem sempre é possível com modelos hierárquicos genéricos.

Para qualquer modelo genérico, sempre existe um modelo mais geral que é decomponível. Este fato é simples de se provar, pois o modelo saturado é trivialmente decomponível.

Outra característica desta classe de modelos é que existe uma fórmula direta para o cálculo das freqüências esperadas para as células evitando o uso de métodos computacionalmente onerosos, como o *Interactive Proportional Fitting* [JP95].

Considere um modelo log-linear decomponível $\mathcal{M}$ com grafo de interação $G = (V, E)$. É possível obter uma ordenação de eliminação perfeita de seus vértices $\pi = \{X_1, \ldots, X_n\}$ e então o cálculo de $\hat{p}_\Delta(i)$ pode ser feito com a seguinte fórmula:

$$\hat{p}_\Delta(i) = \frac{1}{|\mathcal{I}_\Delta|} \times \prod_{k=1\ldots n} \frac{N(i_{\{X_k\} \cup (\partial_{X_k} \cap \{X_k, \ldots, X_n\})})}{N(i_{\partial_{X_k} \cap \{X_k, \ldots, X_n\}})} \tag{3.3}$$

A seção 4.3.3 apresenta o algoritmo para calcular as frequências esperadas de cada célula da tabela de contingência, baseado na equação 3.3.

3.4.3 Medida de Ajustamento

O objetivo do uso de modelos log-lineares nesta dissertação é encontrar um modelo que consiga refletir os inter-relacionamentos entre as variáveis, e que consiga gerar a tabela de contingência o mais próximo possível da original. Para isto é necessário medir o quanto um modelo $\mathcal{M}$ se ajusta aos dados. Uma maneira de se fazer isto é verificar a diferença entre os valores gerados pelo modelo e os valores reais. Existem diversas medidas para isto [Fre87], por exemplo o teste do qui-quadrado (χ^2). A medida utilizada nesta dissertação é a *deviance*, também conhecida como $2L$ e definida formalmente como:

$$dev(\mathcal{M}) = 2 \sum_{i \in \mathcal{I}} N(i) \log \frac{N(i)}{N \hat{p}_\Delta^{\mathcal{M}}(i)} \tag{3.4}$$

A *deviance* é um valor sempre positivo e, quanto mais próximo de zero ela estiver, melhor será o modelo. Se ela for igual a zero então o modelo se ajusta perfeitamente aos dados, como é o caso do modelo saturado. Além disto, a *deviance* segue assintoticamente a distribuição do χ^2 com graus de liberdade iguais ao número de termos iguais a zero [Whi90]. Esta é uma importante ferramenta na análise de modelos log-lineares, porém, ela é apenas uma aproximação e só é válida para grandes conjuntos de dados [Whi90].

Em [Lau96] é apresentada uma fórmula recursiva para se calcular os graus de liberdade. Dados dois modelos log-lineares hierárquicos H_0 e H_1, a *deviance* segue assintoticamente a distribuição do χ^2 com graus de liberdade (df) iguais a:

$$df = dim(H_{A_0}) - dim(H_{A_1}) \tag{3.5}$$

onde A_0 e A_1 são as classes geradoras dos modelos H_0 e H_1. A seguinte fórmula recursiva é usada para calcular a dimensão $dim(A)$:

1. Se $A = \{a\}$ é completo, ou seja, a classe geradora contém apenas um gerador, então:

$$dim(A) = \prod_{\delta \in a} |\mathcal{I}_\delta| - 1$$

2. Se A é a união de duas classes geradoras A_1 e A_2, então:

$$dim(A) = dim(A_1) + dim(A_2) - dim(A_1 \cap A_2)$$

Com estas equações é possível calcular os graus de liberdade para o cálculo da probabilidade de ajuste do modelo. A interseção de duas classes geradoras é o conjunto das variáveis em comum entre elas aplicadas gerador por gerador. Por exemplo, para $A_1 = [AB]$ e $A_2 = [BC][A, C, D]$, $A_1 \cap A_2 = [B][A]$. Em geral testa-se um modelo contra o modelo saturado, que possui zero grau de liberdade. É possível testar também um modelo qualquer contra outro e verificar hipóteses sobre independência parcial e condicional entre variáveis. No caso de existir zeros na tabela de contingência, [Bad95] apresenta uma fórmula para fazer o ajuste dos graus de liberdade.

3.4.4 Seleção do Modelo

Para cada λ-termo igual a zero tem-se um novo modelo log-linear, com o qual pode-se testar seu ajustamento aos dados. O número de modelos possíveis cresce exponencialmente com o número de variáveis e não se conhece um algoritmo polinomial que consiga obter uma solução ótima, ou seja, o modelo que melhor se ajusta aos dados. Algumas heurísticas foram desenvolvidas pela comunidade estatística com o intuito de selecionar um modelo log-linear, como a *backward elimination* e a *forward selection*. A *backward elimination* remove interações do modelo saturado e a *forward selection* insere interações até que um determinado critério de parada seja atingido. Nesta dissertação foi utilizada a *forward selection* trabalhando apenas com modelos decomponíveis.

Para verificar se uma aresta é ou não significante, deve-se calcular a diferença entre as *deviances* e calcular o valor p utilizando a distribuição do χ^2 com graus de liberdade igual à diferença entre os graus de liberdade dos dois modelos. Em geral, para valores p maiores do que 0.05, uma aresta é considerada significante.

O algoritmo 1, a seguir, consiste basicamente em adicionar arestas ao modelo. Todas as arestas da lista $\mathcal{L}$ são arestas que podem ser adicionadas ao modelo $\mathcal{M}_a$, mantendo-o decomponível. O processo pára quando não houver mais nenhuma aresta significante que

Entrada: $\mathcal{M}_0$: modelo decomponível inicial
Saída: modelo decomponível selecionado
1: $\mathcal{M}_a \leftarrow \mathcal{M}_0$
2: calcule $\mathcal{L}$ para o modelo $\mathcal{M}_a$ $\{\mathcal{L}$ = lista das arestas que podem ser adicionadas ao modelo$\}$
3: **while** $(\mathcal{L} \neq \emptyset)\&(\mathcal{M}_a.PValue < limPValue)$ **do**
4: calcule para cada aresta de $\mathcal{L}$ seu valor p
5: **if** $\nexists e \in \mathcal{L}$ tal que e seja significante **then**
6: **return** $\mathcal{M}_a$
7: $\mathcal{M}_a = \mathcal{M}_a \cup \{e\}$, tal que e é a mais significante
8: calcule $\mathcal{L}$ para o modelo $\mathcal{M}_a$
9: **return** $\mathcal{M}_a$

Algoritmo 1: *Forward Selection* usando modelos decomponíveis

possa ser adicionada ou quando o valor p do modelo seja maior ou igual a um limite mínimo.

Este é um método automático para a geração de modelos decomponíveis. Entretanto, a participação do usuário não pode ser deixada de lado. Ao invés de simplesmente retornar o melhor modelo, o método pode retornar os modelos intermediários, gerados na linha 7. Deste modo o usuário tem a possibilidade de escolher o melhor modelo baseado nos valores p de cada modelo e ainda nas interações entre as variáveis presentes neles. Além disto, o usuário pode incluir ou excluir interações entre as variáveis gerando novos modelos. Tudo isto de acordo com o nível de conhecimento do usuário sobre o domínio da aplicação. Para usuários com menos conhecimento sobre os dados, o método permite que ele os entenda melhor. Para aqueles com maior conhecimento sobre o método, ele permite que o usuário participe na seleção do modelo.

3.5 Análise dos Resíduos e Detecção de *Outliers*

Com o modelo log-linear pode-se gerar a tabela de contingência. Se o modelo for aceito como um modelo válido para o conjunto de dados, então os valores da tabela gerada por ele devem ser parecidos com os valores da tabela original.

A diferença entre os valores das células da tabela de contingência e os valores gerados por um modelo log-linear podem ser padronizados usando a equação 3.6. F_i é a frequência observada na tabela de contingência e E_i é a frequência esperada, gerada pelo modelo. Através dela obtém-se o conjunto dos resíduos padronizados, seguindo a distribuição normal padronizada. Com isto é possível marcar as células que possuem um valor muito diferente do esperado. Por exemplo, se o valor $|z_i|$ for maior do que 2, significa que existe

> 0.95 de probabilidade da célula i da tabela de contingência não ter sido gerada pelo mesmo modelo de probabilidade do restante da tabela. Estas células são chamadas de células *outlying*.

$$z_i = \frac{F_i - E_i}{\sqrt{E_i}} \tag{3.6}$$

A existência de muitas células *outlying* significa que o modelo não é um bom modelo para os dados. Isto implica na necessidade de se encontrar outro modelo para os dados, mesmo que com um valor p menor. Mesmo que se tenha um bom modelo podem existir células da tabela de contingência que possuam $|z_i|$ grande, devido à existência de *outliers* nos dados.

Nesta dissertação os valores considerados nos testes foram 2, 2.5 e 3, cujas probabilidades são 0.95, 0.98 e 0.99 respectivamente. Células cujo resíduo for maior do que estes valores são consideradas *outliers*.

3.6 Resumo

Este capítulo apresentou a descrição do problema tratado nesta dissertação na seção 3.1. Foi proposto o uso de modelos log-lineares para tratar a detecção de *outliers* em dados categóricos e esta abordagem foi apresentada ao longo do capítulo.

No próximo capítulo será apresentado o protótipo FOCAD - *Finding Outliers in Categorical Data*, que implementou o método descrito neste capítulo.

Capítulo 4

Aspectos de Implementação

Este capítulo apresenta o protótipo FOCAD - *Finding Outliers in Categorical Data*. Ele foi implementado com objetivo de validar o método apresentado no capítulo 3.

A seção 4.1 mostra a visão geral do protótipo. Na seção 4.2 são apresentadas as principais classes e métodos desenvolvidos. Os algoritmos e estrutura de dados utilizados para sua implementação são apresentados na seção 4.3.

4.1 Visão Geral

O FOCAD é um protótipo de um sistema de análise de dados categóricos utilizando modelos log-lineares, principalmente ajuste, seleção de modelos e detecção de *outliers*. Ele foi desenvolvido na linguagem $Java^{TM}$, versão 1.4.

As funcionalidades básicas do FOCAD são: entrada de dados, seleção do modelo log-linear e análise de resíduos. Nos próximos parágrafos cada uma destas etapas será apresentada de forma simplificada.

A entrada de dados no FOCAD é feita através de arquivos texto formatados. Nesta implementação não foi considerada a captura de dados de um sistema gerenciador de banco de dados ou outras fontes, como XML (*eXtensible Marckup Language*) pois o objetivo era analisar os algoritmos de ajustamento e seleção do modelo. Isto pode ser facilmente acoplado ao FOCAD através do uso JDBC (*Java Database Connectivity*) e API's (*Application Programming Interfaces*) para tratamento de XML disponíveis para a linguagem Java.

A seleção do modelo pode ser feita através do item de menu *Model*$\longrightarrow$*ForwardSelection*. Uma lista dos modelos testados nos passos do algoritmo 1 é apresentada, juntamente com seu valor p e $d.f.$ Esta lista pode ser alterada, com a remoção ou a inserção de novos modelos. Isto possibilita que o usuário com elevado conhecimento da aplicação e, principalmente do conjunto de dados, possa entender o que o algoritmo fez e assim entrar

com novos modelos. Desta forma o usuário pode participar do processo de seleção do modelo, inserindo seu conhecimento no processo. Além disto, no caso de usuários menos experientes, ele pode entender melhor os relacionamentos entre as variáveis através dos modelos listados, verificando como eles foram gerados.

Para cada modelo é possível visualizar a tabela de contingência formada pelos valores reais, valores esperados e ainda dos resíduos padronizados. Na tabela de resíduos padronizados as células com valores acima do limite indicado pelo usuário são destacadas.

Outra funcionalidade interessante do protótipo é a apresentação do grafo que representa o modelo log-linear decomponível. Isto facilita a interpretação do modelo pelo usuário, permitindo que ele veja graficamente as interações entre as variáveis.

4.2 Diagrama de Classes

A figura 4.1 apresenta o diagrama das principais classes do FOCAD. Nele foram inseridas apenas as classes referentes ao método apresentado no capítulo 3. O FOCAD possui ainda outras classes auxiliares e para sua interface com o usuário que não estão neste diagrama.

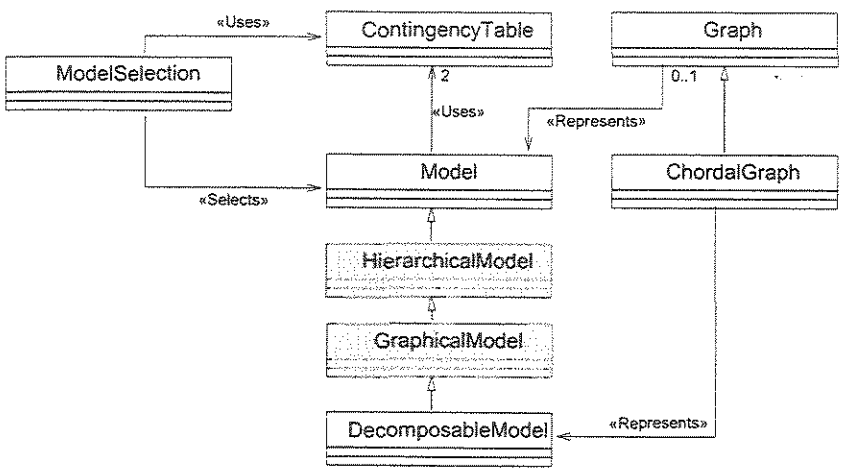


Figura 4.1: Diagrama de classes do FOCAD.

A classe *ContingencyTable* encapsula uma tabela de contingência. A forma de armazenamento dos dados será mostrada a seguir, na seção 4.3.1. Os principais métodos desta classe são:

1. getCell(*key*): dada uma chave *key*, ele devolve a célula da tabela de contingência correspondente à chave *key*. A chave de consulta é formada por um nível de cada variável. Tomando a tabela 3.2(b), um exemplo de chave de consulta é {Cerveja, Masculino, Sul}.

2. getMarginal($Var[]$): gera uma tabela marginal a partir das variáveis contidas no vetor $Var[]$.

É necessário o acesso seqüencial na tabela de contingência para criar uma tabela marginal e ainda para os algoritmos de ajuste do modelo log-linear. Isto é feito através dos métodos initPointerToCurrentCell(), nextCell() e moreCells(). O primeiro inicializa um ponteiro interno para a primeira célula da tabela de contingência; o segundo avança o ponteiro para a próxima célula; e o terceiro verifica se existem mais células na tabela de contingência.

A classe *Graph* encapsula a estrutura de grafos. Os requisitos desta classe foram determinados pelos algoritmos de ajustamento do modelo log-linear. Além dos métodos básicos para criação de um grafo que esta classe possui, como o método addEdge, dois métodos principais dela são:

1. getPEO(): este é o método responsável pelo cálculo da ordem de eliminação perfeita. Ele é necessário para o cálculo dos cliques do grafo e ainda para o ajuste do modelo aos dados. Ele é implementado pelo algoritmo 6 apresentado na seção 4.3.2.

2. isDecomposable(): este método verifica se um grafo qualquer é um grafo decomponível. Ele é usado para verificar se um modelo log-linear qualquer é um modelo decomponível. Ele é implementado através do algoritmo 7 apresentado na seção 4.3.2.

A classe *ChordalGraph* é uma especialização da classe *Graph*. Ela encapsula a estrutura de grafos cordais. Seu principal método é getCliques() que encontra os cliques máximos de um grafo cordal. Ele é implementado através do algoritmo 8 (seção 4.3.3).

A classe *Model* encapsula o modelo log-linear genérico. Um método necessário para esta classe é o método fit(), que faz o ajustamento do modelo aos dados, ou seja, calcula a tabela de contingência com os valores esperados. Nesta dissertação os modelos utilizados foram os modelos decomponíveis. Por este motivo o método fit() não foi implementado na classe *Model*, mas ele pode ser implementado através de algum método iterativo como o IPF. As classes *HierarchicalModel* e *GraphicalModel*, destacadas no diagrama 4.1, também não foram implementadas no protótipo. Elas foram incluídas no diagrama para mostrar a especialização das diferentes classes de modelos log-lineares como na figura 3.2.

A classe *DecomposableModel* implementa o modelo log-linear decomponível. O principal método desta classe é o método fit(). Ele calcula os valores esperados para as células da tabela de contingência através da equação 3.3, não necessitando de nenhum método iterativo.

A implementação do cálculo dos resíduos é feita na classe *Model*, pelo método getResiduals(). Ele foi incluído nesta classe pois ela conhece os valores reais da tabela de

contingência e seus valores esperados. O referido método calcula e cria uma tabela de contingência cujas células são os resíduos padronizados definidos pela fórmula 3.6.

Por fim, a seleção do modelo é feita através da classe *ModelSelection*, mais especificamente pelo método ForwardSelection que implementa o algoritmo 1. Ele devolve uma lista de modelos gerados.

4.3 Algoritmos

Nesta seção serão apresentados os algoritmos e estruturas de dados referentes às classes e métodos apresentados na seção anterior.

4.3.1 Tabela de Contingência

Os algoritmos e estrutura de dados apresentados nesta seção são referentes à classe *ContingencyTable*. Nesta dissertação considera-se que toda a tabela de contingência é armazenada em memória principal. Primeiramente será apresentada uma estrutura de dados baseada em árvore e depois outra estrutura organizada em vetores. A segunda foi a utilizada na dissertação, mas a primeira é apresentada pois ela serviu de base para a confecção dos algoritmos.

A primeira estrutura de dados para armazenamento da tabela de contingência é uma árvore. Cada variável está em um nível da árvore. Cada nó armazena todos os níveis de uma variável, de forma ordenada, e ponteiros para o próximo nível da árvore. O nó folha armazena os valores contidos nas células da tabela de contingência. A figura 4.2 é um exemplo desta estrutura para armazenamento da tabela 3.2(b).

Entrada: T: tabela de contingência
key: chave de pesquisa
Saída: Retorna a célula da tabela de contingência caso exista ou -1 caso contrário

```
1: t ← T
2: for i = 0 to k − 1 do
3:    j ← BinarySearch(t.entries, key[i])
4:    if (j = −1) then
5:       return −1
6:    else
7:       t ← t.next[j]
8: return (t)
```

Algoritmo 2: Seleção de uma célula na tabela de contingência.

O algoritmo 2 é o responsável por selecionar uma célula da tabela de contingência

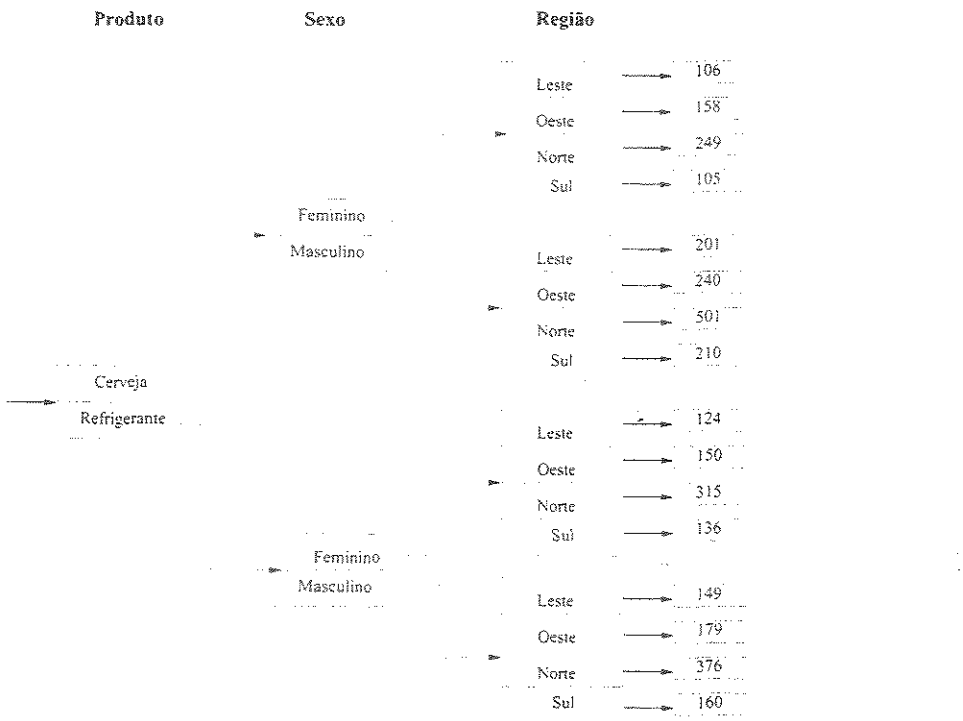


Figura 4.2: Exemplo da estrutura de dados em árvore para armazenar uma tabela de contingência.

armazenada na estrutura apresentada. Ele é um algoritmo simples, que procura em cada nó da árvore pelo ponteiro correspondente ao próximo nível através de uma busca binária. O tempo do algoritmo é $O(\ln n)$, onde n é o número de células da tabela.

Lema 4.1 *O algoritmo 2 está correto e tem complexidade de tempo* $O(\ln n)$*, onde* n *é o número de células da tabela de contingência.*

Rascunho da prova por indução no número de variáveis da tabela. Seja k o número de variáveis na tabela de contingência. Para $k = 1$ o algoritmo está correto, já que consiste em uma busca binária. A hipótese de indução é que o algoritmo está correto para $k = m$ variáveis. O passo de indução é provar que ele está correto para $k = m + 1$ variáveis. Suponha que a nova variável da árvore é um nó folha e a chave existe. Pela hipótese de indução, o algoritmo terá encontrado um ponteiro para o novo nó e com uma busca binária será encontrado o ponteiro para a célula. A prova é trivial se chave não existir. Se a nova variável for um nó raiz ou interno a prova é semelhante a esta. Portanto o algoritmo está correto.

Seja $O(\ln l_{k_i}) + c$ o tempo gasto em um passo do algoritmo 2, onde c é constante e l_{k_i} é o número de níveis da variável k_i, ou seja, o tempo da busca binária. Logo a

complexidade do algoritmo é $((O(\ln l_{k_0}) + c) + (O(\ln l_{k_1}) + c) + \ldots + (O(\ln l_{k_{|k|-1}}) + c) = O(\ln \prod_{i=0}^{|k|-1} l_{k_i}) + (|k| - 1) * c = O(\ln n)$, já que $n = \prod_{i=0}^{|k|-1} l_{k_i}$. □

A estrutura de armazenamento da tabela de contingência em árvore permite um acesso rápido aos dados. Contudo, o espaço necessário para seu armazenamento é grande, já que é necessário manter uma cópia dos mesmos valores nos nós internos da árvore. Uma forma de lidar melhor com isto é manter as células da tabela em um único vetor e conseguir uma forma de indexá-lo. Seja então um vetor vet com n posições onde n é o número de células da tabela de contingência. E sejam os vetores $var[i]$ ($0 \leq i \leq k$, k = número de variáveis) com l_i posições (l_i é o número de níveis da variável i), onde cada entrada é um valor distinto da variável i, mantidos de forma ordenada. Dada uma chave key, pode-se determinar sua posição no vetor vet pelo algoritmo 3, que é uma adaptação do algoritmo 2 para o caso da tabela ser armazenada em vetores. A diferença entre os dois reside no fato deste necessitar guardar os intervalos durante sua execução, enquanto o primeiro fazia isto pelos ponteiros entre os nós da árvore.

Entrada: tabela de contingência
  key: chave de pesquisa
Saída: Retorna a célula da tabela de contingência caso exista ou -1 caso contrário

```
1:  ini ← 0
2:  end ← n − 1
3:  for i = 0 to k − 1 do
4:    j ← BinarySearch(var[i], key[i])
5:    if (j = −1) then
6:      return −1
7:    else
8:      tmp ← (end − ini)/⌈var[i].length⌉
9:      ini ← ini + tmp * j
10:     end ← ini + tmp − 1
11: return (ini)
```

Algoritmo 3: getCell: Seleção de uma célula na tabela de contingência.

A figura 4.3(a) mostra o vetor vet com os elementos da tabela de contingência 3.2(b), na mesma ordem que estão as folhas na figura 4.2. A figura 4.3(b) mostra os vetores $var[i]$, onde serão feitas as buscas para se calcular a posição no vetor vet. Para a chave de consulta {Cerveja, Masculino, Sul} o algoritmo 3 irá subdividir o vetor vet sucessivamente nos seguintes intervalos: 0−15, 0−7, 4−7 e 7−7. A posição 7 será devolvida e $vet[7] = 210$.

As duas formas de representação da tabela de contingência são equivalentes. No entanto, a segunda é mais eficiente em relação ao espaço em memória gasto. Como o tempo de acesso é o mesmo, a segunda forma de armazenamento foi utilizada nesta

dissertação. Na implementação do FOCAD não foi considerado o armazenamento dos dados em disco, mas como eles estão organizados na forma de vetores, torna-se fácil fazer esta implementação.

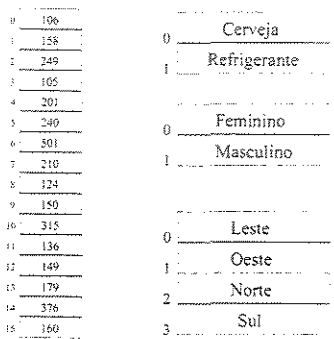


Figura 4.3: Exemplo da estrutura de dados em vetores para armazenamento da tabela de contingência.

A entrada do método é um arquivo formatado, logo a criação da tabela é feita a partir dele. Um algoritmo para criar a tabela de contingência deve ler o arquivo e criar os vetores para cada variável. Uma forma eficiente de fazer isto é usar uma árvore balanceada AVL para cada variável, armazenando seus valores distintos. É necessária uma leitura no arquivo original para criar estes vetores e outra leitura para fazer a contagem de cada categoria. Utiliza-se então o algoritmo 3 para encontrar a célula correspondente à linha de entrada do arquivo.

Para fazer o acesso seqüencial na tabela de contingência, basta manter um "ponteiro" para cada posição do vetor $var[i]$. Isto foi feito pois é necessário conhecer a chave da célula corrente. A atualização deste ponteiro pode ser feita em tempo linear no número de variáveis pelo algoritmo 4. Com este algoritmo o acesso no vetor *vet* é feito em ordem crescente.

Por fim, a geração de uma tabela marginal pode ser feita utilizando-se o algoritmo 5. A criação da nova tabela é simples, já que basta copiar os vetores de índices das variáveis que se deseja e criar um novo vetor para armazenar as células da tabela marginal. A partir daí deve-se varrer toda a tabela original e para cada célula montar uma chave de pesquisa, ou seja, os valores das variáveis que estarão na tabela marginal. Então adiciona-se o valor da célula corrente da tabela original ao valor da célula correspondente à chave k montada da tabela marginal. O tempo do algoritmo é $O(n(\ln n'))$, onde n é o número de células da tabela de contingência original e n' o número de células da tabela marginal.

Entrada: P: ponteiro a ser atualizado
T: a tabela de contingência correspondente
Saída: retorna -1 caso não exista mais células ou o ponteiro atualizado

```
1: for i = k − 1 to 0 do
2:     if (P[i] ≥ |l_{k_i}|) then
3:         P[i] = 0
4:     else
5:         P[i] = P[i] + 1
6:         return
7: return −1
```

Algoritmo 4: Atualização do ponteiro para a célula atual.

Entrada: T: tabela de contingência
var: variáveis para gerar a marginal
Saída: Retorna a tabela marginal

```
1: criar new_tab e inicializa com zero
2: for all células em T do
3:     k = montar chave para célula de new_tab a partir da chave em T
4:     c =getCell(k, new_tab)
5:     Adicionar valor da célula atual de T em c
6: return new_tab
```

Algoritmo 5: Geração da tabela marginal.

4.3.2 Grafos

Nesta seção são apresentados os algoritmos relativos aos métodos das classes *Graph* e *ChordalGraph* apresentados na seção 4.2. Nesta dissertação trabalha-se apenas com grafos simples e não direcionados, portanto eles serão referenciados simplesmente como grafos. Os algoritmos apresentados aqui são algoritmos eficientes e simples de serem implementados.

Tarjan e Yannakakis apresentam em [TY84] um algoritmo para encontrar uma ordem de eliminação perfeita (*p.e.o.*). Dado um grafo cordal $G = (V, E)$, o algoritmo *Maximum Cardinality Search* [TY84] encontra uma *p.e.o* em tempo $O(n+m)$, sendo n o número de vértices do grafo e m o número de arestas. Caso o grafo não seja cordal, ele irá produzir uma ordenação qualquer.

No algoritmo 6 (*Maximum Cardinality Search*) é mantida uma lista de conjuntos $list(i), i \leq 0 \leq n$. Cada $list(i)$ armazena todos os vértices não numerados adjacentes a i vértices numerados. Assim, inicialmente $list(0)$ contém todos os vértices do grafo (linha 2). A variável j indica o maior conjunto não vazio e i é a próxima posição para ordenação. $size(v)$ indica o número de vértices numerados adjacentes a v. Se v é numerado, $size(v) = -1$. Em cada passo do algoritmo um vértice v é removido de $list(j)$ e numerado (linhas 8 e 9). Para cada vértice w adjacente a v, w é movido do conjunto $list(i)$ para $list(i+1)$ (linhas 11, 12 e 13). j deve ser incrementado e i decrementado (linha 14).

Entrada: $G = (V, E)$
Saída: π: ordem de eliminação do grafo.

```
 1: for i = 0 to n − 1 do
 2:    list(i) ← ∅
 3: for all v ∈ V do
 4:    size(v) ← 0
 5:    list(0).add(v)
 6: i ← n; j ← 0
 7: while i ≥ 1 do
 8:    v ← list(j).first();
 9:    π(v) ← i; π^{-1}(i) ← v; size(v) ← −1
10:    for (v, w) ∈ E|size(w) ≥ 0 do
11:       list(size(w)).delete(w)
12:       size(w) ← size(w) + 1
13:       list(size(w)).add(w)
14:    i ← i − 1; j ← j + 1
15:    while (j ≥ 0) e list(j) = ∅ do
16:       j ← j − 1
```

Algoritmo 6: *Maximum Cardinality Search*

Para verificar se um grafo é cordal é apresentado primeiramente o conceito de *Fill-In* de um grafo. O *Fill-In* é um "preenchimento" do grafo para torná-lo um grafo cordal, ou seja, é um conjunto de arestas. Seja então um grafo $G = (V, E)$ e uma ordenação π de seus vértices. O *Fill-In* produzido pela ordenação π é um conjunto de arestas $F(\pi)$ formado da seguinte forma:

$$F(\pi) = \{\{v, w\} | \{v, w\} \notin E \text{ e existe um caminho de } v \text{ para } w \text{ contendo somente } v, w \text{ e vértices ordenados antes de } v \text{ e } w\} \quad (4.1)$$

Se $F(\pi) = \emptyset$, então π é uma ordem de eliminação perfeita de G, ou seja, G é um grafo cordal [TY84].

Tarjan e Yannakakis apresentam o algoritmo *Test for Zero Fill-In* [TY84] para verificar se um algoritmo é cordal. O algoritmo verifica se, dada uma *p.e.o.*, o *Fill-In* produzido por ela é vazio. Este algoritmo possui complexidade temporal $O(n + m)$. Com este algoritmo e o *Maximum Cardinality Search* é possível verificar se um grafo é cordal em tempo linear.

O algoritmo 7 funciona da seguinte forma. Dada uma *p.e.o.* π, percorra ela verificando se não é necessário criar nenhuma aresta em $F(\pi)$. No algoritmo 7, $index(w)$ é a posição na qual o vértice w está ordenado segundo π; $f(v)$ armazena o vértice w tal que $\pi^{-1}(w) < \pi^{-1}(v)$, ou seja, o "pai" de v. A linha 8 do algoritmo diz o seguinte: se o índice do "pai" de v estiver antes de w, então o *Fill-In* não será vazio.

Entrada: $G = (V, E)$ e π
Saída: *true* se π produzir $F(\pi) = \emptyset$ ou $false$ caso contrário.

```
1: for i = 0 to n − 1 do
2:    w ← π^{-1}(i); f(w) ← w; index(w) ← i
3:    for v|(v, w) ∈ E e π(v) < i do
4:       index(v) ← i
5:       if f(v) = v then
6:          f(v) ← w
7:    for v|(v, w) ∈ E e π(v) < i do
8:       if index(f(v)) < i then
9:          return false
10: return true
```

Algoritmo 7: *Test for Zero Fill-In*

Em [Gol80] são apresentadas as seguintes proposições de Fulkerson e Gross [FG65]:

Proposição 4.1 *Dados um grafo $G = (V, E)$ e um vértice u, um clique maximal é da forma $\{u\} \cup X_u$, onde X_u é o conjunto de vértices que estão na adjacência de u e estão após u, segundo π, ou seja, $X_u = \{x \in \delta(u) | \pi(u) < \pi(x)\}$.*

Proposição 4.2 *Um grafo cordal com n vértices possui n cliques maximais, se e somente se o grafo não possui arestas.*

Baseado nestas proposições, o algoritmo *CLIQUES* (apresentado no livro [Gol80], página 99) foi proposto para calcular os cliques maximais de um grafo cordal. Ele percorre os vértices segundo uma *p.e.o.*, e para cada vértice v_i ele forma o conjunto X_{v_i}. O algoritmo *CLIQUES* também possui complexidade $O(n+m)$.

Entrada: $G = (V, E)$ e π
Saída: Devolve os cliques maximais do grafo.

```
1:  for all v ∈ V do
2:     S(v) ← 0
3:  for 1 to n do
4:     v ← π(i)
5:     X ← {x ∈ δ(v) | π^{-1} < π^{-1}(x)}
6:     if δ(v) = ∅ then
7:        print(v)
8:     if X ≠ ∅ then
9:        u ← π(min{π^{-1}(x) | x ∈ X})
10:       S(u) ← max{S(u), |X| − 1}
11:       if S(v) < |X| then
12:          print({v} ∪ X)
```

Algoritmo 8: *CLIQUES*

4.3.3 Modelo

Nesta seção são apresentados os algoritmos relativos às classes *Model* e *DecomposableModel*, principalmente o algoritmo de ajustamento do modelo.

No caso do modelo não ser um modelo decomponível, é necessário o uso de um algoritmo iterativo como o IPF. Nesta dissertação ele não foi considerado por se tratar de um algoritmo caro computacionalmente. Deste modo, o ajuste do modelo foi implementado apenas para modelos decomponíveis, pelo método fit() da classe *DecomposableModel*. Para isto é levada em consideração a fórmula 3.3 apresentada na seção 3.4.2 e reescrita aqui para facilitar o entendimento do algoritmo:

$$\hat{p}_{\Delta}(i) = \frac{1}{|\mathcal{I}_{\Delta}|} \times \prod_{k=1\ldots n} \frac{N(i_{\{X_k\} \cup (\partial_{X_k} \cap \{X_k,\ldots,X_n\})})}{N(i_{\partial_{X_k} \cap \{X_k,\ldots,X_n\}})}$$

O algoritmo percorre todas as células da tabela de contingência para calcular a frequência esperada $\hat{p}_{\Delta}(i)$. $\mathcal{I}_{\Delta}$ é o valor do somatório de todas as células da tabela. A

ordem para percorrer as variáveis ($k = 1 \ldots n$) é dada pela ordem de eliminação perfeita dos vértices do grafo que representa o modelo. Em cada iteração é necessário calcular as tabelas marginais $N(i_{\{X_k\} \cup (\partial_{X_k} \cap \{X_k, \ldots, X_n\})})$ e $N(i_{\partial_{X_k} \cap \{X_k, \ldots, X_n\}})$.

A medida de ajustamento do modelo utilizada nesta dissertação foi a *deviance* (equação 3.4, página 27). Para calculá-la, é necessário apenas percorrer as tabelas de contingência contendo os valores reais e esperados, ou seja, em tempo linear é possível calcular a *deviance*.

O cálculo dos resíduos também é simples. Basta varrer toda a tabela de contingência original e a tabela com as frequências esperadas, aplicando a equação 3.6 (página 30). O tempo do algoritmo é linear no tamanho da tabela de contingência.

4.4 Resumo

Este capítulo apresentou uma visão geral do FOCAD, desenvolvido em Java. Ele é um protótipo de um sistema de análise de dados categóricos utilizando modelos log-lineares, principalmente ajuste e seleção de modelos e detecção de *outliers*. A seção 4.2 mostrou suas principais classes e métodos e a seção 4.3 os algoritmos necessários à implementação.

Capítulo 5

Resultados

Este capítulo apresenta os resultados de testes realizados ao longo deste trabalho. Além de testes de desempenho, foram realizados testes com dados reais da área de telecomunicações, fornecidos pela empresa CPqD Telecom & IT Solutions, chamado simplesmente de CPqD por facilidade.

A seção 5.1 descreve os tipos de dados utilizados nos testes. A seção 5.2 apresenta um comentário sobre o teste realizado com o algoritmo *Nested Loop* de detecção de *outliers* baseados em distância. Na seção 5.3 são mostrados os resultados do método proposto nesta dissertação. A seção 5.4 traz resultados específicos de desempenho realizados com dados sintéticos.

5.1 Seleção dos Dados e Pré-processamento

Para os testes foram utilizados dados reais da área de telecomunicações, gentilmente cedidos pelo CPqD. Eles foram extraídos de um *Data Warehouse* construído com o objetivo de análise da carteira de clientes de uma determinada companhia telefônica. O responsável pelo desenvolvimento do *Data Warehouse* participou dos testes na seleção dos dados e análise dos resultados.

Como todo trabalho em mineração de dados, a primeira fase consistiu no estudo do domínio da aplicação. Para isto, duas abordagens foram adotadas: (i) entrevistas com um especialista, neste caso o responsável pelo desenvolvimento do *Data Warehouse*; e (ii) utilização de algoritmos de *clustering* e árvores de decisão. Foi utilizado o módulo de mineração de dados da ferramenta Microsoft SQL Server 2000®.

A próxima fase foi a seleção dos atributos considerados relevantes para os testes de detecção de *outliers*. Na base de dados (neste caso o *Data Warehouse*) existiam 49 atributos, sendo 16 numéricos, 4 do tipo data/hora e o restante discretos e/ou categóricos. Destes, os 9 atributos descritos na tabela 5.1 foram considerados mais relevantes.

Dentre os atributos selecionados, 3 eram numéricos e 1 do tipo data/hora. Para os testes apresentados na seção 5.3 eles foram discretizados, de acordo com a tabela de 5.1. Para a discretização foi considerado o domínio da aplicação. Por exemplo, o atributo `horario-chamada` foi discretizado em `diurno` e `noturno`. Em outro tipo de aplicação esta discretização talvez não faria sentido, mas neste contexto sabe-se que as chamadas feitas durante o dia são mais caras do que as chamadas noturnas.

Os dados foram então extraídos de um servidor Microsoft SQL Server 2000 e transformados em arquivos texto separados por vírgula (.csv).

Nome do atributo	Descrição	Categorização
F-Entre-Faturam	É o tempo entre a data da chamada e o seu faturamento.	`pouco` (< 3 dias), `médio` (entre 3 e 10 dias), `muito` (≥ 10 dias)
Categoria-objeto	Tipo do serviço prestado pela companhia telefônica.	`cabo`, `calling card`, `internet`, `telefone fixo`
Tempo-chamada	Duração da chamada.	`pouco` ($<$ 7min), `médio` (entre 7 e 60 minutos), `muito` ($>$ 1h)
Horario-chamada	Horário em que a chamada foi realizada.	`diurno` (06:00h – 18:00h) e `noturno` (0:00 – 5:59h e 18:00h – 23:59h)
Categoria-segmento	Indica se o cliente é uma pessoa física ou jurídica.	`pessoa física` e `pessoa jurídica`
Categoria-chamada	A chamada pode ser interestadual ou intra-estadual.	`interestadual`, `intra-estadual`
Tipo-moradia	Tipo da moradia do cliente.	`própria`, `alugada`, `cedida`
Categoria-cliente	Indica se o cliente é um empregado ou não da própria companhia telefônica.	`consumidor`, `empregado`
M-dias-pgto	Tempo médio gasto para o cliente efetuar o pagamento.	< 10, $10 \leq M < 50$, $>= 50$

Tabela 5.1: Descrição dos atributos utilizados durante os testes.

5.2 Testes do algoritmo *Nested Loop*

O algoritmo *Nested Loop* de Knorr [KN98], descrito na seção 2.5.2, foi implementado e testado com os dados apresentados na seção anterior. O objetivo deste teste foi tentar utilizar o algoritmo com dados reais da área de telecomunicações.

Além dos atributos numéricos, foram utilizados os atributos categóricos também. Contudo, os métodos baseados em distância necessitam que uma função de distância seja definida entre os elementos do conjunto. No conjunto de dados utilizado, vários atributos são categóricos e não possuem uma função de distância definida. Para contornar este problema foi feito um mapeamento de cada categoria para um valor numérico, com o objetivo de tornar possível o uso da função de distância euclidiana. Talvez fosse necessário criar uma função de distância específica para cada subconjunto testado. Entretanto, esta abordagem foi descartada em virtude do grande número de possibilidades existentes. Mesmo que se encontrasse uma boa função de distância para um determinado conjunto, esta função provavelmente não iria funcionar para outro conjunto de dados. Isto tornaria o processo de detecção de *outliers* muito árduo para o usuário, já que seria exigido dele a especificação da função de distância para cada conjunto com o qual ele desejasse trabalhar.

Mesmo com a dificuldade encontrada, foram gerados alguns subconjuntos com os dados numéricos. No entanto, em nenhum dos testes realizados foram encontrados *outliers* que fossem relevantes para o usuário, ou seja, os *outliers* não passavam nenhuma informação útil para o usuário. É uma tarefa difícil analisar este resultado, pois o algoritmo apenas retorna um elemento do conjunto que foi considerado *outlier* sem nenhuma informação adicional.

Com este teste notou-se que os métodos de detecção de *outliers* baseados em distância são difíceis de serem aplicados quando os dados são categóricos. Não foi testado outro método baseado em distância em virtude de ser necessária a especificação da função de distância quando os atributos são categóricos. Acredita-se que em bancos de dados existam muitos atributos categóricos, tornando métodos baseados em distância difíceis de serem utilizados na prática. Isto levou ao desenvolvimento do método apresentado no capítulo 3, cujos resultados dos testes são apresentados na próxima seção.

5.3 Testes do método proposto

Esta seção apresenta os resultados obtidos com o método descrito no capítulo 3 aplicados aos dados da área de telecomunicações descritos na seção 5.1.

A tabela 5.2 apresenta os valores de p para os valores da distribuição normal padronizada, utilizados como parâmetro para considerar uma célula *outlier* ou não durante os testes. Por exemplo, se o valor do resíduo de uma célula da tabela de contingência for

maior ou igual a 3, significa que existe 0.27% de chance desta célula pertencer ao mesmo modelo que gerou o restante da tabela. Isto indica a presença de um *outlier*, seja inerente à população ou um erro ocorrido.

z	Valor de p	1-p
2.0	0.9545	0.0455
2.5	0.9876	0.0124
3.0	0.9973	0.0027

Tabela 5.2: Valores de p para a distribuição normal.

A seguir são apresentados os resultados de dois testes realizados. O objetivo é ilustrar a utilização da abordagem proposta nesta dissertação e mostrar sua eficácia.

5.3.1 Teste I

Neste teste foram utilizados os atributos descritos na tabela 5.3. Na primeira coluna estão os atributos e o apelido dado ao atributo durante os testes. Por exemplo, o atributo `tipo-moradia` recebeu o apelido `A`. A segunda coluna mostra os valores dos atributos e seus apelidos.

Atributo	Valores do atributo
A (Tipo-moradia)	1: própria
	2: alugada
	3: cedida
B (Categoria-cliente)	1: consumidor
	2: empregado
C (M-dias-pgto)	1: $10 \leq x < 50$
	2: < 10
	3: ≥ 50
D (Categoria-objeto)	1: cabo
	2: *calling-card*
	3: internet
	4: telefone fixo

Tabela 5.3: Descrição dos atributos utilizados no teste I.

Neste teste a tabela de contingência foi gerada a partir de 16.770 registros do banco

de dados. As células que possuem valores zero foram marcadas como zeros estruturais para que o algoritmo de ajustamento não tentasse ajustá-las.

Os parâmetros do algoritmo de seleção de modelos foram: (i) número máximo de iterações=20; (ii) valor p mínimo de um modelo para terminar=0.99; e (iii) valor mínimo de p para adicionar uma aresta ao grafo, ou seja, para acrescentar uma interação entre os atributos=0.05.

A tabela 5.4 apresenta os modelos gerados pelo algoritmo 1, linha 7 (veja seção 3.4.4 para mais detalhes sobre o algoritmo). Os modelos intermediários são apresentados de modo que o usuário possa usá-los para entender os inter-relacionamentos entre os atributos. Na tabela estão os modelos gerados, a medida *deviance* e o valor p do modelo. Esta é a probabilidade para o ajuste do modelo. O valor DF na tabela corresponde aos graus de liberdade ajustado do modelo (podem existir zeros na tabela de contingência).

O modelo 4 foi selecionado e a figura 5.1 mostra o grafo que o representa. Nota-se que existe interação entre os atributos A e B; A e C; e C e D. A *deviance* do modelo é muito pequena (15.12) e o modelo possui um ótimo ajustamento aos dados reais ($p = 0.993$).

	Modelo	Deviance	DF	p
1	[A] [B] [C] [D]	275.07	43	0.000
2	[D] [C] [AB]	166.16	41	0.000
3	[CD] [AB]	84.85	35	0.000
4	[CD] [AC] [AB]	15.12	31	0.993

Tabela 5.4: Modelos selecionados no teste I.

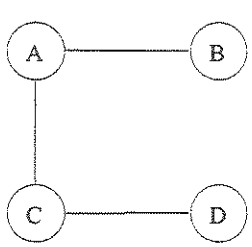


Figura 5.1: Grafo que representa o modelo 4 apresentado na tabela 5.4.

A tabela 5.5 contém os valores das células da tabela de contingência para o teste I. Para cada célula são apresentados os valores reais, ou seja, os valores contidos no banco de dados; os valores ajustados gerados pelo modelo 4 da tabela 5.4 e os resíduos padronizados. As três últimas colunas indicam as células que possuem o resíduo maior que 2, 2.5 e 3.

Esta tabela foi apresentada aqui apenas para demonstrar que o modelo conseguiu gerar uma tabela de contingência com valores bem próximos dos valores reais.

Note que a diferença entre os valores gerados pelo modelo são bem próximos dos valores reais. Apenas as células marcadas exibem uma discrepância entre os dois valores, o que caracteriza um *outlier*. Tome a linha 4 por exemplo. O valor real da célula é 49 e o valor gerado pelo modelo é de 19.7133. Agora, olhe a linha 7(note as demais células também). O valor real da célula é 4 e o valor gerado pelo modelo é 3.8025.

O limite para considerar uma célula *outlier* é um parâmetro a ser informado pelo usuário. Porém, os resíduos podem ser ordenados de forma a identificar os mais discrepantes. Esta é uma importante característica, pois além de informar que um elemento é um *outlier*, o método também provê um valor bem definido em 0 e 1; neste caso a probabilidade do valor da célula obedecer ao modelo gerado.

Tabela 5.5: Tabela de contingência do teste I.

	A	B	C	D	Real	Ajustado	Resíduos	> 2.0	> 2.5	> 3.0
1	1	1	1	1	0	0.0000	0.0000			
2	1	1	1	2	0	0.0000	0.0000			
3	1	1	1	3	0	0.0000	0.0000			
4	1	1	1	4	49	19.7133	6.5962	X	X	X
5	1	1	2	1	1	1.2085	-0.1897			
6	1	1	2	2	2	0.8096	1.3231			
7	1	1	2	3	4	3.8025	0.1013			
8	1	1	2	4	4	2.9613	0.6036			
9	1	1	3	1	1	0.6120	0.4960			
10	1	1	3	2	0	0.0000	0.0000			
11	1	1	3	3	3	5.5611	-1.0860			
12	1	1	3	4	4	4.1022	-0.0505			
13	1	2	1	1	1	1.9621	-0.6868			
14	1	2	1	2	7	2.2744	3.1334	X	X	X
15	1	2	1	3	3	7.7565	-1.7079			
16	1	2	1	4	7	6.9576	0.0161			
17	1	2	2	1	0	0.0000	0.0000			
18	1	2	2	2	0	0.0000	0.0000			
19	1	2	2	3	0	0.0000	0.0000			
20	1	2	2	4	0	0.0000	0.0000			
21	1	2	3	1	0	0.0000	0.0000			
22	1	2	3	2	2	0.1410	4.9517	X	X	X
23	1	2	3	3	2	1.1606	0.7792			

continua na próxima pagina

Tabela 5.5: Tabela de contingência do teste I (continuação).

	A	B	C	D	Real	Ajustado	Resíduos	> 2.0	> 2.5	> 3.0
24	1	2	3	4	2	1.4478	0.4589			
25	2	1	1	1	7	8.7918	-0.6043			
26	2	1	1	2	11	10.5019	0.1537			
27	2	1	1	3	54	36.8882	2.8174	X	X	
28	2	1	1	4	22	37.2636	-2.5004	X	X	
29	2	1	2	1	2	0.8140	1.3145			
30	2	1	2	2	1	0.5611	0.5859			
31	2	1	2	3	2	2.7108	-0.4317			
32	2	1	2	4	2	2.3630	-0.2362			
33	2	1	3	1	0	0.0000	0.0000			
34	2	1	3	2	0	0.0000	0.0000			
35	2	1	3	3	10	6.6749	1.2870			
36	2	1	3	4	3	5.5114	-1.0697			
37	2	2	1	1	0	0.0000	0.0000			
38	2	2	1	2	2	0.8291	1.2859			
39	2	2	1	3	2	2.9122	-0.5346			
40	2	2	1	4	3	2.9419	0.0339			
41	2	2	2	1	0	0.0000	0.0000			
42	2	2	2	2	0	0.0000	0.0000			
43	2	2	2	3	0	0.0000	0.0000			
44	2	2	2	4	0	0.0000	0.0000			
45	2	2	3	1	0	0.0000	0.0000			
46	2	2	3	2	0	0.0000	0.0000			
47	2	2	3	3	1	0.5270	0.6516			
48	2	2	3	4	1	0.4351	0.8564			
49	3	1	1	1	1395	1405.1509	-0.2708			
50	3	1	1	2	1665	1678.4558	-0.3284			
51	3	1	1	3	5918	5895.6454	0.2911			
52	3	1	1	4	5972	5955.6423	0.2120			
53	3	1	2	1	100	99.1451	0.0859			
54	3	1	2	2	68	68.3428	-0.0415			
55	3	1	2	3	336	330.1630	0.3212			
56	3	1	2	4	292	287.8097	0.2470			
57	3	1	3	1	33	32.0322	0.1710			
58	3	1	3	2	44	43.3377	0.1006			
59	3	1	3	3	310	308.0744	0.1097			

continua na próxima pagina

Tabela 5.5: Tabela de contingência do teste I (continuação).

	A	B	C	D	Real	Ajustado	Resíduos	> 2.0	> 2.5	> 3.0
60	3	1	3	4	260	254.3733	0.3528			
61	3	2	1	1	18	13.8861	1.1040			
62	3	2	1	2	25	16.5869	2.0657	X		
63	3	2	1	3	51	58.2623	-0.9514			
64	3	2	1	4	65	58.8552	0.8010			
65	3	2	2	1	0	0.0000	0.0000			
66	3	2	2	2	0	0.0000	0.0000			
67	3	2	2	3	1	3.2628	-1.2527			
68	3	2	2	4	1	2.8442	-1.0935			
69	3	2	3	1	0	0.0000	0.0000			
70	3	2	3	2	0	0.0000	0.0000			
71	3	2	3	3	1	3.0445	-1.1717			
72	3	2	3	4	0	0.0000	0.0000			

5.3.2 Teste II

A tabela 5.6 mostra os atributos utilizados no teste II.

Os modelos gerados no segundo teste aparecem na tabela 5.7. Diferente do teste I, neste não se conseguiu um modelo com valor p muito alto. Entretanto, os modelos 14 e 15 possuem *deviance* pequena. Quando isto ocorre é interessante analisar os resíduos, pois podem existir poucos resíduos acima do limite fornecido pelo usuário. O modelo 14 possui 6 células com resíduo acima de 2. Já o modelo 15 possui várias células, mesmo com o valor p um pouco mais alto. Por este motivo o modelo 14 foi aceito.

A tabela 5.8 mostra apenas as linhas cujas células possuem resíduos maiores que 2. A tabela completa não foi incluída aqui pois é muito grande. Ela está no apêndice A.

5.4 Desempenho

Existem dois pontos principais em relação ao desempenho do método proposto nesta dissertação:

Tempo de ajuste do modelo: optou-se pela utilização dos modelos log-lineares decomponíveis pois eles podem ser ajustados por uma fórmula direta, dada pela equação 3.3. Deste modo, não é necessário o uso de métodos iterativos como o IPF que são lentos. Mesmo assim, o processo de ajuste necessita que toda a tabela de contingência seja percorrida. Como o tamanho dela é determinado pelo número

Atributo	Valores do atributo
A (F-Entre-Faturam)	1: médio
	2: muito
	3: pouco
B (Categoria-objeto)	1: cabo
	2: *calling-card*
	3: internet
	4: fixo
C (Tempo-chamada)	1: médio
	2: muito
	3: pouco
D (Horario-chamada)	1: diurno
	2: noturno
E (Categoria-segmento)	1: pessoa jurídica
	2: pessoa física
F (Categoria-chamada)	1: interestadual
	2: intra-estadual

Tabela 5.6: Descrição dos atributos utilizados no teste II.

	Modelo	Deviance	DF	p
1	[A] [B] [C] [D] [E] [F]	94	7568.59	0
2	[F] [D] [C] [BE] [A]	93	3920.98	0
3	[F] [C] [DE] [BE] [A]	92	1835.32	0
4	[F] [DE] [CE] [BE] [A]	90	1167.3	0
5	[DE] [CE] [BF] [BE] [A]	88	881.2	0
6	[BF] [DE] [CE] [BE] [AE]	86	639.15	0
7	[DE] [CE] [BEF] [AE]	84	461.23	0
8	[CDE] [BEF] [AE]	80	285.00	0
9	[CDE] [CEF] [BEF] [AE]	76	179.25	0
10	[BEF] [CEF] [CDE] [ADE]	72	135.63	0
11	[BCEF] [CDE] [ADE]	66	120.65	0
12	[BCEF] [BCDE] [ADE]	61	82.57	0.0344
13	[BCEF] [BCDE] [ABDE]	53	63.69	0.1494
14	[BCDEF] [ABDE]	45	51.48	0.2351
15	[BCDEF] [ABDEF]	31	36.02	0.2452

Tabela 5.7: Modelos selecionados no teste II.

	A	B	C	D	E	F	Real	Ajustado	Resíduos	> 2.0	> 2.5	> 3.0
67	1	3	3	1	2	1	1014	912.1789	3.3713	X	X	X
139	2	3	3	1	2	1	1330	1425.3353	-2.5252	X	X	
184	3	2	2	2	2	2	1	0.0984	2.8749	X	X	
195	3	3	1	1	2	1	38	24.8713	2.6325	X	X	
196	3	3	1	1	2	2	60	39.3476	3.2924	X	X	X
204	3	3	2	1	2	2	2	0.4186	2.4443	X		

Tabela 5.8: Resumo da tabela de contingência do teste II.

de atributos do conjunto de dados e pelo número de níveis de cada atributo, o tempo de ajuste do modelo cresce exponencialmente em relação ao número de atributos.

Tempo de seleção do modelo: nesta dissertação foi utilizada a heurística *forward selection* apenas para modelos log-lineares decomponíveis. Mesmo assim, o número possíveis de modelos a serem considerado é muito grande.

Os testes de desempenho foram realizados utilizando dados sintéticos. Foi gerado um conjunto de 9 atributos com 3 valores distintos cada. Com isto, a tabela de contingência formada a partir deste conjunto possui $3^9 = 16.683$ células.

Todos os testes foram executados em um PC Intel Xeon com 2 processadores de 2.8GHz; 2Gb de memória RAM rodando o sistema operacional Linux Red Hat *release* 9 (Shrike).

A seguir são apresentados os testes de desempenho realizados para os principais pontos citados acima.

5.4.1 Ajuste do modelo

O tempo de ajuste do modelo foi medido em relação ao número de atributos do conjunto de dados. A figura 5.2 apresenta o gráfico do tempo de ajuste do modelo em função do número de atributos. Para uma tabela de contingência com 16.583 células, foram necessários 78 segundos em média para o ajuste de cada modelo considerado pelo *forward selection*.

5.4.2 Seleção do modelo

A figura 5.3 mostra o gráfico do número de modelos testados pelo *forward selection* em função do número de atributos.

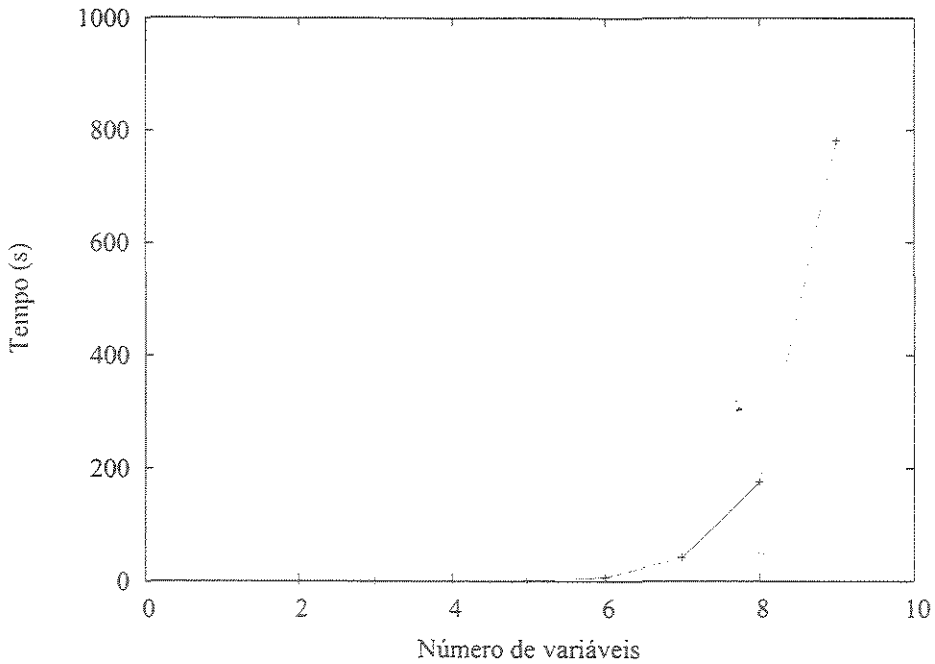


Figura 5.2: Tempo para ajuste do modelo log-linear decomponível.

5.4.3 Comentários

Os resultados apresentados mostraram que o método proposto não pode ser utilizado para conjuntos de dados com muitos atributos em virtude do tempo gasto na seleção do modelo. Nos testes realizados com 8 e 9 atributos, o tempo de resposta é aceitável. Contudo, para um conjunto com dez ou mais atributos, o tempo de seleção do modelo torna-se impraticável.

A sugestão proposta para lidar com grandes conjuntos é fazer a seleção do modelo manualmente em conjuntos menores. Com isto, o usuário pode testar as interações entre os atributos e deixá-las para incluí-las no modelo final. Por fim, tendo o modelo em mãos, o usuário pode fazer o ajuste dele para todo o conjunto em um tempo aceitável.

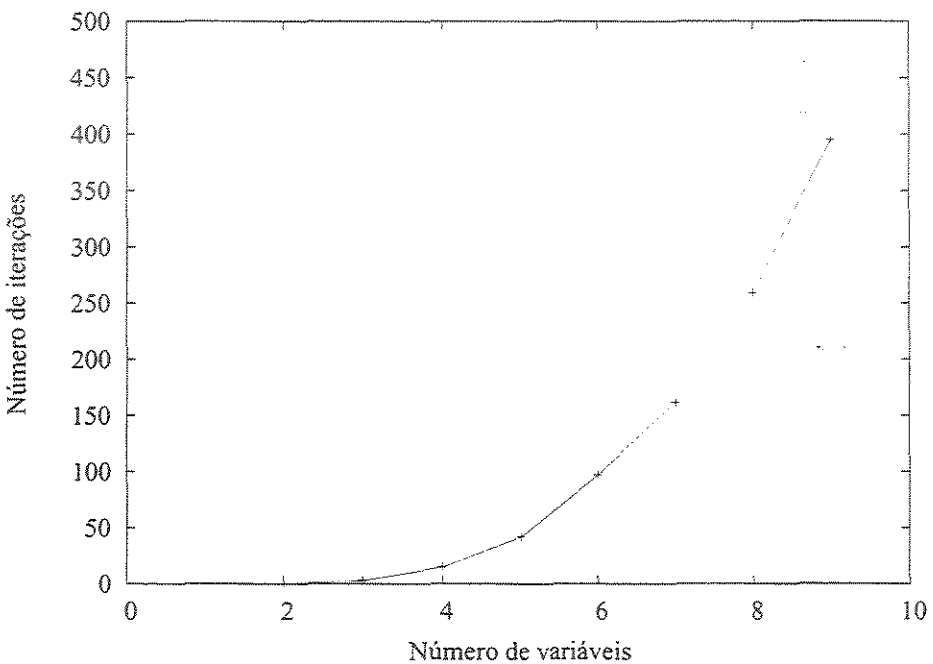


Figura 5.3: Número de iterações x número de atributos.

Capítulo 6

Conclusões

Esta dissertação concentrou-se no problema de detecção de *outliers* em conjuntos de dados com atributos categóricos. A detecção de *outliers* pode trazer informações não esperadas e importantes para algumas aplicações, como por exemplo: descoberta de fraudes em sistemas telefônicos e de cartão de crédito, sistemas de detecção de intrusão e detecção de erros em banco de dados.

Com testes realizados em dados reais notou-se que métodos de detecção de *outliers* baseados em distância são difíceis de serem aplicados quando os dados são categóricos. A principal dificuldade é a necessidade de uma função de distância, que nem sempre é bem definida para dados categóricos. Assim, o usuário deve especificar a função de distância, o que torna o método difícil de ser aplicado na prática.

Devido às dificuldades encontradas, esta dissertação propôs uma nova abordagem, no âmbito de mineração de dados, baseada no uso de modelos log-lineares. Além de retornar os *outliers* encontrados, o método desenvolvido retorna um modelo descrevendo os dados e dois valores de probabilidade: um para o ajuste do modelo aos dados e um para os *outliers*. Isto propicia ao usuário meios para que ele entenda os *outliers* encontrados.

O método desenvolvido foi aplicado a dados reais da área de telecomunicações e os resultados foram promissores. Os *outliers* encontrados diferem significantemente do modelo selecionado para os dados. Por outro lado, os demais valores gerados pelo modelo estão bem próximos dos valores reais. Pelos testes realizados, o método consegue responder em um bom tempo para conjuntos com até 9 atributos. Para um número de atributos maior, o tempo de seleção do modelo pode tornar-se impraticável devido ao grande número de modelos possíveis. Talvez esta não seja uma restrição muito grande, pois utilizar muitos atributos em um único teste dificulta a interpretação dos resultados.

As principais contribuições desta dissertação foram:

1. Proposta de uma nova abordagem, no âmbito de mineração de dados, para detecção de *outliers* em conjuntos de dados com atributos categóricos, que:

- provê um modelo para os dados reais;
- permite a interação com o usuário; e
- retorna uma medida para os *outliers* e para o modelo selecionado.

2. Implementação de um protótipo de sistema de análise de dados categóricos, permitindo ao usuário:
   - selecionar e ajustar um modelo aos dados;
   - fazer testes estatísticos entre modelos; e
   - detectar os *outliers*.

6.1 Trabalhos Futuros

Esta dissertação deixa possibilidades para os seguintes trabalhos futuros:

- **Detecção de *outliers* em conjuntos com atributos numéricos e categóricos.** Esta dissertação tratou apenas de atributos categóricos. Existem na literatura propostas de vários métodos para tratar atributos numéricos. Já que os bancos de dados possuem ambos os tipos de dados, é importante que se integre o método proposto nesta dissertação com métodos que lidam com atributos numéricos.

- **Melhorar o desempenho do método.**
  O tempo de ajuste do modelo é aceitável. O principal problema é o tempo de seleção do modelo. Ele é um problema difícil de ser resolvido, porém o uso de heurísticas pode melhorar o tempo de resposta.

- **Integração com OLAP**
  Na literatura já existem propostas de integração de ferramentas de mineração de dados e OLAP. A integração do método proposto com OLAP mostra-se promissora. O método proposto trabalha com atributos categóricos, os quais podem ser as dimensões de um cubo. É necessário apenas fazer a adaptação de algumas funções para tratar as hierarquias existentes em um cubo OLAP.

- **Desenvolver um sistema para detecção de erros em bancos de dados**
  A ocorrência de erros em bancos de dados é grande, seja por erros de digitação, erros de aplicação ou até mesmo erros provenientes de conversão de dados. Um sistema que possibilite a identificação de tais erros reduziria muito o trabalho manual, pois o usuário não precisaria procurar pelos erros, mas apenas fazer uma correção semi-automática.

Bibliografia

[AAR96] R. Agrawal A. Arning e P. Raghavan. A linear method for deviation detection in large databases. Em *Int'l Conference on Knowledge Discovery in Databases and Data Mining*, página 164, Agosto,1996.

[AMS+95] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, e A. I. Verkamo. Fast discovery of association rules, 1995.

[Bad95] J. H. Badsberg. *An Environment for Graphical Models.* Tese de Doutorado, Institute for Electronic Systems - Aalborg University, 1995.

[BKNS99] M. M. Breunig, H.P. Kriegel, R. T. Ng, e J. Sander. OPTICS-OF: Identifying local outliers. Em *Principles of Data Mining and Knowledge Discovery*, páginas 262–270, 1999.

[BKNS00] M. M. Breunig, H.P. Kriegel, R. T. Ng, e J. Sander. LOF: identifying density-based local outliers. Em *ACM SIGMOD 2000*, páginas 93–104, 2000.

[BL84] V. Barnett e T. Lewis. *Outliers in statistical data.* Wiley, 1984.

[CP99] X. Chen e I. Petrounias. Mining temporal features in association rules. Em J. M. Zytkow e J. Rauch, editores, *3rd European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'99)*, volume 1704, páginas 295–300, Prague, 1999. Springer.

[CTF03] Z. Chen, J. Tang, e A. W-C Fu. Modeling and efficient mining of intentional knowledge of outliers. Em *Seventh International Database Engineering and Applications Symposium (IDEAS'03), Hong Kong, China.* IEEE Computer Society, 2003.

[DSZ99] Yu D., Sheikholeslami S., e A. Zhang. FindOut: Finding outliers in very large datasets. Relatório Técnico 99-03, Department of Computer Science and Engineering State University of New York at Buffalo Buffalo, 5, 1999.

[EH99] D. W. Cheung E. Hung. Parallel algorithm for mining outliers in large database. Relatório Técnico, Department of Computer Science and Information Systems, The University of Hong Kong, 1999.

[EKSX96] M. Ester, H.P. Kriegel, J. Sander, e X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. Em *2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, páginas 266–231, 1996.

[FF01] C. C. Fabris e A. A. Freitas. Incorporating Deviation-Detection functionality into the OLAP Paradigm. Em *XVI SBBD - Simpósio Brasileiro de Banco de Dados*, páginas 274–285, 2001.

[FG65] D. R. Fulkerson e O. A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15:835–855, 1965.

[FPSS96] U. Fayyad, G. Piatetsky-Shapiro, e P. Smyth. Knowledge discovery and data mining: Towards a unifying framework. Em *Int'l Conference on Knowledge Discovery and Data Mining*, 1996.

[Fre87] D. H. Freeman, Jr. *Applied Categorical Data Analysis.* Marcel Dekker, Inc., 1987.

[Gol80] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Computer Science and Applied Mathematics. Academic Press, Inc., 1980.

[Haw80] D. Hawkins. *Identification of Outliers.* Chapman and Hall, 1980.

[HK01] J. Han e M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2001.

[HKT01] J. Han, M. Kamber, e A. K. H. Tung. *Geographic Data Mining And Knowledge Discovery*, capítulo Spatial Clustering Methods in Data Mining: A Survey. Taylor & Francis, Inc., 2001.

[JP95] R. Jiroušek e S. Přeučil. On the effective implementation of the iterative proportional fitting procedure. *Computational Statistics & Data Analysis*, 19:177–189, 1995.

[KN97] E. M. Knorr e R. T. Ng. A unified notion of outliers: Properties and computation. Em *Proc. Knowledge Discovery and Data Mining (KDD)*, páginas 219–222, 1997.

[KN98] Edwin Knorr e Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. Em *Proc. 24th VLDB Conference, New York, USA*, páginas 392–403, 1998.

[KN99] E. M. Knorr e R. T. Ng. Finding intensional knowledge of distance-based outliers. Em *The VLDB Journal*, páginas 211–222, 1999.

[Kno02] E. M. Knorr. *Outliers and Data Mining: Finding Exceptions in Data.* Tese de Doutorado, The University of British Columbia, Abril 2002.

[KNZ01] E. M. Knorr, R. T. Ng, e R. H. Zamar. Robust space transformations for distance-based operations. Em *Proc. Knowledge Discovery and Data Mining (KDD)*, páginas 126–135, 2001.

[Lau96] S. L. Lauritzen. *Graphical Models.* Oxford Science Publication, 1996.

[MM00] J. Maletic e A. Marcus. Automated identification of errors in data sets. Relatório Técnico, The University of Memphis, Division of Computer Science, 2000.

[MM01] R. C. Miller e B. A. Myers. Outlier finding: focusing user attention on possible errors. Em *ACM Symposium on User Interface Software and Technology*, páginas 81–90, 2001.

[NH94] R. T. Ng e J. Han. Efficient and effective clustering methods for spatial data mining. Em *Proc. 20th VLDB Conference, Santiago, Chile*, páginas 144–155, 1994.

[Orr98] K. Orr. Data quality and systems theory. *Communications of the ACM*, 41(2):66–71, 1998.

[RRS00] S. Ramaswamy, R. Rastogi, e K. Shim. Efficient algorithms for mining outliers from large data sets. Em *Proc. of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, USA*, volume 29, páginas 427–438. ACM, 2000.

[SAM98] S. Sarawagi, R. Agrawal, e N. Megiddo. Discovery-driven exploration of OLAP data cubes. Em *Proc. 6th Int. Conf. Extending Database Technology, EDBT*, volume 1377, páginas 168–182. Springer-Verlag, 23–27 1998.

[Set01] W. W. Setzer. *Meios Eletrônicos e Educação: uma visão alternatina*, capítulo Dado, informação, conhecimento e competência, páginas 239–275. Escrituras Editora, 2001.

[SLZ01] S. Shekhar, C.T. Lu, e P. Zhang. Detecting graph-based spatial outliers: algorithms and applications (a summary of results). Em *Int'l Conference on Knowledge Discovery and Data Mining*, páginas 371–376, 2001.

[TY84] R. E. Tarjan e M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

[Whi90] J. Whittaker. *Graphical Models in Applied Multivariate Statistics*. Wiley, 1990.

[ZRL96] T. Zhang, R. Ramakrishnan, e M. Livny. BIRCH : An efficient data clustering method for very large databases. Em *ACM SIGMOD Conference on Management of Data*, páginas 103–114, 1996.

Apêndice A

Resultados dos testes.

A tabela A.1 apresenta os resultados do teste apresentado na seção 5.3.2.

Tabela A.1: Tabela de contingência completa do teste II.

	A	B	C	D	E	F	Real	Ajustado	Resíduos	> 2.0	> 2.5	> 3.0
1	1	1	1	1	1	1	0	0	0			
2	1	1	1	1	1	2	0	0	0			
3	1	1	1	1	2	1	0	0	0			
4	1	1	1	1	2	2	0	0	0			
5	1	1	1	2	1	1	0	0	0			
6	1	1	1	2	1	2	0	0	0			
7	1	1	1	2	2	1	0	0	0			
8	1	1	1	2	2	2	0	0	0			
9	1	1	2	1	1	1	0	0	0			
10	1	1	2	1	1	2	0	0	0			
11	1	1	2	1	2	1	0	0	0			
12	1	1	2	1	2	2	0	0	0			
13	1	1	2	2	1	1	0	0	0			
14	1	1	2	2	1	2	0	0	0			
15	1	1	2	2	2	1	0	0	0			
16	1	1	2	2	2	2	0	0	0			
17	1	1	3	1	1	1	0	0	0			
18	1	1	3	1	1	2	0	0	0			
19	1	1	3	1	2	1	0	0	0			
20	1	1	3	1	2	2	0	0	0			
21	1	1	3	2	1	1	0	0	0			
22	1	1	3	2	1	2	0	0	0			

continua na próxima pagina

Tabela A.1: Tabela de contingência completa do teste II (continuação).

	A	B	C	D	E	F	Real	Ajustado	Resíduos	> 2.0	> 2.5	> 3.0
23	1	1	3	2	2	1	0	0	0			
24	1	1	3	2	2	2	0	0	0			
25	1	2	1	1	1	1	11	10.2506	0.2341			
26	1	2	1	1	1	2	75	74.3171	0.0792			
27	1	2	1	1	2	1	5	5.9016	-0.3711			
28	1	2	1	1	2	2	1	2.582	-0.9845			
29	1	2	1	2	1	1	0	0	0			
30	1	2	1	2	1	2	2	2.4135	-0.2662			
31	1	2	1	2	2	1	3	4.9836	-0.8886			
32	1	2	1	2	2	2	10	7.3443	0.98			
33	1	2	2	1	1	1	0	0	0			
34	1	2	2	1	1	2	0	0	0			
35	1	2	2	1	2	1	0	0	0			
36	1	2	2	1	2	2	1	0.3689	1.0392			
37	1	2	2	2	1	1	0	0	0			
38	1	2	2	2	1	2	0	0	0			
39	1	2	2	2	2	1	1	1.0492	-0.048			
40	1	2	2	2	2	2	1	0.7869	0.2402			
41	1	2	3	1	1	1	146	139.665	0.5361			
42	1	2	3	1	1	2	1438	1445.7673	-0.2043			
43	1	2	3	1	2	1	23	19.918	0.6906			
44	1	2	3	1	2	2	15	16.2295	-0.3052			
45	1	2	3	2	1	1	7	12.0675	-1.4588			
46	1	2	3	2	1	2	134	128.519	0.4835			
47	1	2	3	2	2	1	8	9.4426	-0.4695			
48	1	2	3	2	2	2	9	8.3934	0.2094			
49	1	3	1	1	1	1	46	45.7582	0.0357			
50	1	3	1	1	1	2	83	86.8944	-0.4178			
51	1	3	1	1	2	1	248	268.5316	-1.2529			
52	1	3	1	1	2	2	436	424.8298	0.5419			
53	1	3	1	2	1	1	12	12.6833	-0.1919			
54	1	3	1	2	1	2	33	36.9929	-0.6565			
55	1	3	1	2	2	1	361	368.2969	-0.3802			
56	1	3	1	2	2	2	611	583.1367	1.1538			
57	1	3	2	1	1	1	1	0.9244	0.0786			

continua na próxima pagina

Tabela A.1: Tabela de contingência completa do teste II (continuação).

	A	B	C	D	E	F	Real	Ajustado	Resíduos	> 2.0	> 2.5	> 3.0
58	1	3	2	1	1	2	0	0	0			
59	1	3	2	1	2	1	4	8.6623	-1.5841			
60	1	3	2	1	2	2	5	4.5195	0.226			
61	1	3	2	2	1	1	2	1.0569	0.9173			
62	1	3	2	2	1	2	0	0	0			
63	1	3	2	2	2	1	21	17.0047	0.9689			
64	1	3	2	2	2	2	16	14.931	0.2767			
65	1	3	3	1	1	1	580	560.1915	0.8369			
66	1	3	3	1	1	2	1907	1923.2315	-0.3701			
67	1	3	3	1	2	1	1014	912.1789	3.3713	X	X	X
68	1	3	3	1	2	2	2385	2473.2779	-1.7751			
69	1	3	3	2	1	1	53	43.863	1.3796			
70	1	3	3	2	1	2	197	201.347	-0.3063			
71	1	3	3	2	2	1	778	740.7412	1.369			
72	1	3	3	2	2	2	1745	1807.8896	-1.4791			
73	2	1	1	1	1	1	0	0	0			
74	2	1	1	1	1	2	0	0	0			
75	2	1	1	1	2	1	0	0	0			
76	2	1	1	1	2	2	0	0	0			
77	2	1	1	2	1	1	0	0	0			
78	2	1	1	2	1	2	0	0	0			
79	2	1	1	2	2	1	0	0	0			
80	2	1	1	2	2	2	0	0	0			
81	2	1	2	1	1	1	0	0	0			
82	2	1	2	1	1	2	0	0	0			
83	2	1	2	1	2	1	0	0	0			
84	2	1	2	1	2	2	0	0	0			
85	2	1	2	2	1	1	0	0	0			
86	2	1	2	2	1	2	0	0	0			
87	2	1	2	2	2	1	0	0	0			
88	2	1	2	2	2	2	0	0	0			
89	2	1	3	1	1	1	0	0	0			
90	2	1	3	1	1	2	0	0	0			
91	2	1	3	1	2	1	2	2	0			
92	2	1	3	1	2	2	2	2	0			

continua na próxima pagina

Tabela A.1: Tabela de contingência completa do teste II (continuação).

	A	B	C	D	E	F	Real	Ajustado	Resíduos	> 2.0	> 2.5	> 3.0
93	2	1	3	2	1	1	0	0	0			
94	2	1	3	2	1	2	0	0	0			
95	2	1	3	2	2	1	0	0	0			
96	2	1	3	2	2	2	0	0	0			
97	2	2	1	1	1	1	13	13.7432	-0.2005			
98	2	2	1	1	1	2	99	99.6384	-0.064			
99	2	2	1	1	2	1	10	9.7049	0.0947			
100	2	2	1	1	2	2	6	4.2459	0.8513			
101	2	2	1	2	1	1	0	0	0			
102	2	2	1	2	1	2	2	1.519	0.3903			
103	2	2	1	2	2	1	15	13.3934	0.439			
104	2	2	1	2	2	2	18	19.7377	-0.3911			
105	2	2	2	1	1	1	0	0	0			
106	2	2	2	1	1	2	0	0	0			
107	2	2	2	1	2	1	0	0	0			
108	2	2	2	1	2	2	0	0	0			
109	2	2	2	2	1	1	0	0	0			
110	2	2	2	2	1	2	0	0	0			
111	2	2	2	2	2	1	3	2.8197	0.1074			
112	2	2	2	2	2	2	1	2.1148	-0.7666			
113	2	2	3	1	1	1	181	187.2514	-0.4568			
114	2	2	3	1	1	2	1946	1938.367	0.1734			
115	2	2	3	1	2	1	30	32.7541	-0.4812			
116	2	2	3	1	2	2	28	26.6885	0.2539			
117	2	2	3	2	1	1	12	7.5949	1.5984			
118	2	2	3	2	1	2	76	80.8861	-0.5433			
119	2	2	3	2	2	1	26	25.377	0.1237			
120	2	2	3	2	2	2	23	22.5574	0.0932			
121	2	3	1	1	1	1	53	53.0145	-0.002			
122	2	3	1	1	1	2	105	100.674	0.4312			
123	2	3	1	1	2	1	427	419.5971	0.3614			
124	2	3	1	1	2	2	632	663.8225	-1.2351			
125	2	3	1	2	1	1	12	11.0605	0.2825			
126	2	3	1	2	1	2	37	32.2598	0.8346			
127	2	3	1	2	2	1	490	480.3917	0.4384			

continua na próxima pagina

Tabela A.1: Tabela de contingência completa do teste II (continuação).

	A	B	C	D	E	F	Real	Ajustado	Resíduos	> 2.0	> 2.5	> 3.0
128	2	3	1	2	2	2	736	760.6202	-0.8927			
129	2	3	2	1	1	1	1	1.071	-0.0686			
130	2	3	2	1	1	2	0	0	0			
131	2	3	2	1	2	1	18	13.5354	1.2135			
132	2	3	2	1	2	2	5	7.0619	-0.7759			
133	2	3	2	2	1	1	0	0	0			
134	2	3	2	2	1	2	2	0.9217	1.1232			
135	2	3	2	2	2	1	20	22.1802	-0.4629			
136	2	3	2	2	2	2	20	19.4753	0.1189			
137	2	3	3	1	1	1	627	649.0258	-0.8646			
138	2	3	3	1	1	2	2246	2228.2148	0.3768			
139	2	3	3	1	2	1	1330	1425.3353	-2.5252	X	X	
140	2	3	3	1	2	2	3982	3864.6478	1.8877			
141	2	3	3	2	1	1	30	38.2509	-1.3341			
142	2	3	3	2	1	2	178	175.5854	0.1822			
143	2	3	3	2	2	1	928	966.1933	-1.2287			
144	2	3	3	2	2	2	2413	2358.1391	1.1297			
145	3	1	1	1	1	1	0	0	0			
146	3	1	1	1	1	2	0	0	0			
147	3	1	1	1	2	1	0	0	0			
148	3	1	1	1	2	2	0	0	0			
149	3	1	1	2	1	1	0	0	0			
150	3	1	1	2	1	2	0	0	0			
151	3	1	1	2	2	1	0	0	0			
152	3	1	1	2	2	2	0	0	0			
153	3	1	2	1	1	1	0	0	0			
154	3	1	2	1	1	2	0	0	0			
155	3	1	2	1	2	1	0	0	0			
156	3	1	2	1	2	2	0	0	0			
157	3	1	2	2	1	1	0	0	0			
158	3	1	2	2	1	2	0	0	0			
159	3	1	2	2	2	1	0	0	0			
160	3	1	2	2	2	2	0	0	0			
161	3	1	3	1	1	1	0	0	0			
162	3	1	3	1	1	2	0	0	0			

continua na próxima pagina

Tabela A.1: Tabela de contingência completa do teste II (continuação).

	A	B	C	D	E	F	Real	Ajustado	Resíduos	> 2.0	> 2.5	> 3.0
163	3	1	3	1	2	1	0	0	0			
164	3	1	3	1	2	2	0	0	0			
165	3	1	3	2	1	1	0	0	0			
166	3	1	3	2	1	2	0	0	0			
167	3	1	3	2	2	1	0	0	0			
168	3	1	3	2	2	2	0	0	0			
169	3	2	1	1	1	1	0	0	0			
170	3	2	1	1	1	2	0	0	0			
171	3	2	1	1	2	1	1	0.3934	0.967			
172	3	2	1	1	2	2	0	0	0			
173	3	2	1	2	1	1	0	0	0			
174	3	2	1	2	1	2	0	0	0			
175	3	2	1	2	2	1	1	0.623	0.4777			
176	3	2	1	2	2	2	0	0	0			
177	3	2	2	1	1	1	0	0	0			
178	3	2	2	1	1	2	0	0	0			
179	3	2	2	1	2	1	0	0	0			
180	3	2	2	1	2	2	0	0	0			
181	3	2	2	2	1	1	0	0	0			
182	3	2	2	2	1	2	0	0	0			
183	3	2	2	2	2	1	0	0	0			
184	3	2	2	2	2	2	1	0.0984	2.8749	X	X	
185	3	2	3	1	1	1	0	0	0			
186	3	2	3	1	1	2	1	0.8657	0.1443			
187	3	2	3	1	2	1	1	1.3279	-0.2845			
188	3	2	3	1	2	2	1	1.082	-0.0788			
189	3	2	3	2	1	1	1	0.3376	1.1402			
190	3	2	3	2	1	2	3	3.5949	-0.3138			
191	3	2	3	2	2	1	2	1.1803	0.7545			
192	3	2	3	2	2	2	0	0	0			
193	3	3	1	1	1	1	0	0	0			
194	3	3	1	1	1	2	0	0	0			
195	3	3	1	1	2	1	38	24.8713	2.6325	X	X	
196	3	3	1	1	2	2	60	39.3476	3.2924	X	X	X
197	3	3	1	2	1	1	0	0	0			

continua na próxima pagina

Tabela A.1: Tabela de contingência completa do teste II (continuação).

	A	B	C	D	E	F	Real	Ajustado	Resíduos	> 2.0	> 2.5	> 3.0
198	3	3	1	2	1	2	0	0	0			
199	3	3	1	2	2	1	37	39.3114	-0.3687			
200	3	3	1	2	2	2	59	62.2431	-0.4111			
201	3	3	2	1	1	1	0	0	0			
202	3	3	2	1	1	2	0	0	0			
203	3	3	2	1	2	1	1	0.8023	0.2207			
204	3	3	2	1	2	2	2	0.4186	2.4443	X		
205	3	3	2	2	1	1	0	0	0			
206	3	3	2	2	1	2	0	0	0			
207	3	3	2	2	2	1	0	0	0			
208	3	3	2	2	2	2	0	0	0			
209	3	3	3	1	1	1	5	2.7828	1.3292			
210	3	3	3	1	1	2	8	9.5537	-0.5027			
211	3	3	3	1	2	1	78	84.4858	-0.7056			
212	3	3	3	1	2	2	200	229.0744	-1.921			
213	3	3	3	2	1	1	0	0	0			
214	3	3	3	2	1	2	6	4.0676	0.9581			
215	3	3	3	2	2	1	80	79.0655	0.1051			
216	3	3	3	2	2	2	201	192.9712	0.578			