Classificação de dados estacionários e não estacionários baseada em grafos

João Roberto Bertini Junior

Classificação de dados estacionários e não estacionários baseada em grafos

João Roberto Bertini Junior

Orientador: ***Prof. Dr. Zhao Liang***

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências - Ciências de Computação e Matemática Computacional.

USP – São Carlos
Novembro de 2010

Agradecimentos

- A Deus;
- Aos meus pais João Roberto Bertini e Luzia Monzani Bertini por todo o amor e incentivo ao longo de todos esses anos;
- À minha namorada Rafaela, pelo carinho, apoio e compreenção.
- Ao meu orientador e amigo Zhao Liang, pela dedicação, pelos ensinamentos e pelo trabalho desenvolvido;
- Ao professor Alneu Lopes pelas várias ideias e pela ajuda no desenvolvimento deste trabalho;
- À CAPES pelo apoio financeiro.

Resumo

Métodos baseados em grafos consistem em uma poderosa forma de representação e abstração de dados que proporcionam, dentre outras vantagens, representar relações topológicas, visualizar estruturas, representar grupos de dados com formatos distintos, bem como, fornecer medidas alternativas para caracterizar os dados. Esse tipo de abordagem tem sido cada vez mais considerada para solucionar problemas de aprendizado de máquina, principalmente no aprendizado não supervisionado, como agrupamento de dados, e mais recentemente, no aprendizado semissupervisionado. No aprendizado supervisionado, por outro lado, o uso de algoritmos baseados em grafos ainda tem sido pouco explorado na literatura. Este trabalho apresenta um algoritmo não paramétrico baseado em grafos para problemas de classificação com distribuição estacionária, bem como sua extensão para problemas que apresentam distribuição não estacionária. O algoritmo desenvolvido baseia-se em dois conceitos, a saber, 1) em uma estrutura chamada grafo K-associado ótimo, que representa o conjunto de treinamento como um grafo esparso e dividido em componentes; e 2) na medida de pureza de cada componente, que utiliza a estrutura do grafo para determinar o nível de mistura local dos dados em relação às suas classes. O trabalho também considera problemas de classificação que apresentam alteração na distribuição de novos dados. Este problema caracteriza a mudança de conceito e degrada o desempenho do classificador. De modo que, para manter bom desempenho, é necessário que o classificador continue aprendendo durante a fase de aplicação, por exemplo, por meio de aprendizado incremental. Resultados experimentais sugerem que ambas as abordagens apresentam vantagens na classificação de dados em relação aos algoritmos testados.

Abstract

Graph-based methods consist in a powerful form for data representation and abstraction which provides, among others advantages, representing topological relations, visualizing structures, representing groups of data with distinct formats, as well as, supplying alternative measures to characterize data. Such approach has been each time more considered to solve machine learning related problems, mainly concerning unsupervised learning, like clustering, and recently, semi-supervised learning. However, graph-based solutions for supervised learning tasks still remain underexplored in literature. This work presents a non-parametric graph-based algorithm suitable for classification problems with stationary distribution, as well as its extension to cope with problems of non-stationary distributed data. The developed algorithm relies on the following concepts, 1) a graph structure called optimal K-associated graph, which represents the training set as a sparse graph separated into components; and 2) the purity measure for each component, which uses the graph structure to determine local data mixture level in relation to their classes. This work also considers classification problems that exhibit modification on distribution of data flow. This problem qualifies concept drift and worsens any static classifier performance. Hence, in order to maintain accuracy performance, it is necessary for the classifier to keep learning during application phase, for example, by implementing incremental learning. Experimental results, concerning both algorithms, suggest that they had presented advantages over the tested algorithms on data classification tasks.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

O aprendizado de máquina compreende o desenvolvimento de algoritmos que permitem que computadores aprendam a partir de dados empíricos. Abordagens frequentes incluem os aprendizados *supervisionado*, *semissupervisionado* e *não supervisionado*, diferenciando-se com relação à presença de rótulos nos dados, que os distinguem entre diferentes classes (Mitchell, 1997), (Duda *et al.*, 2001), (Han e Kamber, 2006). Quando um conjunto de instâncias de dados[1] apresenta uma classe associada, um algoritmo de aprendizado supervisionado pode ser usado para inferir, com base nesse conjunto, uma função com o objetivo de predizer a classe de novos dados. Neste contexto, diferenciam-se problemas de *classificação* e de *regressão*, no primeiro caso os valores das classes pertencem a um intervalo finito de valores discretos (Hastie *et al.*, 2009); no segundo caso, os rótulos assumem valores em um intervalo contínuo (Cogger, 2010). Um cenário oposto ao que se aplica o aprendizado supervisionado é o que apresenta total abstinência de classes no conjunto de dados. Neste caso, dito não supervisionado, a tarefa mais comum é tentar agrupar os dados segundo alguma relação de similaridade, com o objetivo de extrair alguma informação útil do conjunto de dados. Este tipo de tarefa chama-se *agrupamento de dados* (Jain *et al.*, 1999), e é aplicada a dados onde não há informações sobre classe. Outro tipo de tarefa que não depende da informação de classe é a *redução de dimensionalidade* (Belkin e Niyogi, 2003) e, por essa razão, também classifica-se como não supervisionada. Uma abordagem intermediária, aplicável a conjuntos de dados que dispõem de dados rotulados e não rotulados, é o aprendizado semissupervisionado. Neste tipo de aprendizado o objetivo é aprender a partir de dados rotulados e não rotulados (Zhu, 2008). De maneira geral, o aprendizado semissupervisionado é dividido em duas abordagens, classificação semissupervisionada cujo objetivo é atribuir classe a dados não rotulados, sejam os já existentes ou dados ainda não vistos (Chapelle *et al.*, 2006); e o agrupamento de dados

[1] Neste trabalho 'instância' e 'exemplo' de dados são usadas indistintamente.

semissupervisionado, no qual o objetivo é agrupar os dados, mas obedecendo algumas restrições que impedem que determinados exemplos estejam no mesmo grupo ou vice-versa (Kulis *et al.*, 2009).

De fato, frente aos problemas inerentes ao aprendizado de máquina, diversas abordagens de pesquisa têm sido propostas. Nos últimos anos, soluções baseadas em grafos (ou redes[2]) têm inspirado muitas pesquisas na área de aprendizado de máquina e mineração de dados. Este crescente interesse pode ser justificado devido a algumas vantagens quando algum tipo de conhecimento é representado por meio de grafos. Métodos baseados em grafos são capazes de (i) capturar a estrutura topológica de um conjunto de dados; (ii) permitem representação hierárquica, ou seja, um grafo pode ser particionado em subgrafos que, por sua vez, pode ser divido em subgrafos ainda menores e assim por diante; (iii) permitem a detecção de agrupamentos ou classes com formas arbitrárias (Karypis *et al.*, 1999), (Schaeffer, 2007); (iv) possibilitam combinar estruturas locais e estatísticas globais de um conjunto de dados (Hein *et al.*, 2007), (Strogatz, 2001).

Talvez a área de pesquisa mais ativa no que diz respeito ao uso de algoritmos baseados em grafos em aprendizado de máquina seja o aprendizado não supervisionado, principalmente agrupamento de dados. De acordo com Schaeffer (2007), a tarefa de encontrar bons agrupamentos de dados utilizando grafos pode ser realizada por meio de duas abordagens, utilizar relações de similaridade entre vértices ou otimizar alguma função, particionando o grafo. No primeiro caso, calcula-se algum valor para cada vértice e os agrupamentos são formados por meio da comparação desses valores, de forma que, quanto maior a semelhança entre dois vértices, maior a chance de serem agrupados. Ou seja, neste tipo de método tenta-se, com o auxílio do grafo, produzir grupos que não só estejam bem conectados, mas inclua instâncias similares possivelmente de regiões esparsas. Relações de similaridade (Wong e Yao, 1992), adjacência (Capoccia *et al.*, 2005) e conectividade (Hartuv e Shamir, 2000) são comumente usados neste tipo de abordagem. No segundo caso, usa-se alguma função que quantifica a qualidade de um grupo de dados ou de uma partição. De posse de um grafo que representa os dados, a ideia é diretamente encontrar os grupos de modo a satisfazer algum critério estabelecido. Exemplos comuns de tais critérios são medidas de densidade aplicadas a subgrafos (Hu *et al.*, 2005), e medidas baseadas em corte de arestas (Kannan *et al.*, 2004). Recentemente a tarefa de agrupamento de dados, especialmente de grandes bases de dados, tem sido beneficiada pela teoria das *redes complexas*. Definida como redes que possuem topologia não trivial (vistas como grafo de grande escala), essas redes podem ser compostas por milhares ou até bilhões de vértices (Albert e Barabási, 2002), (Newman, 2003), (Dorogovtsev e Mendes, 2003). No domínio das redes complexas, agrupamento de dados pode ser visto como um problema de *detecção de comunidades*, no qual o objetivo é identificar grupos de vértices densamente conectados, denominados comunidades (Newman e Girvan, 2004).

Métodos de detecção de comunidades, bem como outros métodos de agrupamento de

[2] Os termos 'grafo' e 'rede' têm o mesmo significado e, portanto, são intercambiáveis.

dados que usam grafos, baseiam-se na separação de grupos de vértices fracamente conectados para revelar as partições. Uma vez que as superfícies de decisão devem encontrar-se em meio a vértices conectados de maneira esparsa. Ainda no que se refere a métodos baseados em grafos, se algumas das instâncias apresentam rótulos de classe, além de revelar os agrupamentos de dados por meio da separação dos grupos por conexões esparsas, é necessário espalhar os rótulos das instâncias rotuladas para as não rotuladas. Portanto, métodos semissupervisionados também supõem que os rótulos estão distribuídos de maneira suave entre os vértices, ou seja, se dois vértices, pertencentes a um agrupamento densamente conectado, são próximos, então também devem ser suas saídas (classes). Estas suposições permitem a distribuição dos rótulos em regiões densamente conectadas e a separação de classes em regiões de conexão esparsa (Chapelle *et al.*, 2006). Como já mencionado, o objetivo na classificação semissupervisionada é atribuir rótulos às instâncias não rotuladas, sendo que esta pode ser dividida em dois tipos de aplicação, a saber, *transdução* e *indução*. Métodos que pertencem ao primeiro caso são chamados métodos *transdutivos*, bem como os pertencem ao segundo caso são chamados métodos *indutivos* (Culp e Michailidis, 2008). Para esclarecer as diferenças entre as categorias considere o cenário no qual um conjunto de dados apresenta apenas uma parte das instâncias rotuladas. No aprendizado transdutivo o objetivo é propagar os rótulos das instâncias rotuladas para as demais. Neste tipo de aprendizado a preocupação é classificar os dados já existentes, ou seja, não supõe a apresentação de novos dados. Já no aprendizado indutivo, considerando o mesmo cenário, a prioridade é aprender utilizando todos os dados disponíveis, rotulados e não rotulados, com o objetivo de induzir um classificador capaz de classificar novos dados de entrada.

De acordo com Zhu (2008) os métodos semissupervisionados baseados em grafos, são por natureza, transdutivos. Uma vez que o grafo é formado conectando-se vértices que representam dados rotulados e não rotulados, a tarefa de espalhar os rótulos é facilitada pela estrutura do grafo. De fato, isso explica porque a tarefa de transdução compreende a maioria dos algoritmos baseados em grafos propostos na literatura, (Belkin e Niyogi, 2004), (Zhou e Schölkopf, 2004). A tarefa de indução também tem sido explorada, geralmente por meio de alguma alteração em um algoritmo transdutivo (Zhu e Lafferty, 2005), (Chen *et al.*, 2009a). De fato, Delalleau *et al.* (2005) propuzeram um método geral para realizar indução através de um algoritmo transdutivo. No entanto, métodos semissupervisionados pressupõem a existência de dados não rotulados, e utilizam o grafo como uma maneira de ajustar os dados a uma estrutura capaz de revelar certas propriedades. De modo que, quando todos os dados possuem classe, utilizar uma estrutura para evidenciar as classes é redundante e métodos supervisionados devem ser empregados. Quando utilizado um método supervisionado, pressupõe-se que todas as instâncias disponíveis tenham classe conhecida (instâncias de treinamento). Neste caso o grau de dificuldade de um problema depende da variabilidade no espaço de atributos das instâncias de uma mesma classe relativa à diferença entre o espaço de atributos de uma classe diferente (Hastie *et al.*, 2009). De maneira que, mesmo que de posse do conjunto de treinamento,

a tarefa de classificar novas instâncias não é trivial. Um classificador deve, a princípio, extrair o máximo de informações dos dados de treinamento para inferir, com base nessas informações, a classe dos novos exemplos de dados (Theodoridis e Koutroumbas, 2008). Esta dificuldade se deve às várias variáveis envolvidas no processo de classificação, como características no conjunto de dados, diversidade na escolha do algoritmo para a construção do classificador, natureza do problema, entre outros.

Outro tipo de preocupação, inerente à classificação de dados, refere-se à mudanças na distribuição dos dados após o treinamento do classificador. Este problema, conhecido como *mudança de conceito* (*concept drift*) (Helmbold e Long, 1994), (Widmer e Kubat, 1996), inviabiliza um classificador, treinado somente com dados iniciais, de manter um bom desempenho, uma vez que a distribuição dos novos dados não correspondem à distribuição dos dados aprendida. Quando a distribuição dos dados se altera ao longo do tempo, diz-se que esta é *não estacionária*, já se a distribuição permanece a mesma esta é dita *estacionária*. Em geral, problemas com distribuição não estacionária são comuns em processamento de fluxo de dados, no qual o sistema precisa aprender e classificar em tempo real. Muitas aplicações reais requerem classificação em tempo real, de monitoramento em redes de sensores, (Cohen *et al.*, 2008) ao processamento de informações produzidas por sistemas muito maiores como a internet, ou sistemas de telecomunicações (Last, 2002). Uma maneira de manter o desempenho de classificação, mesmo na presença de mudanças no conceito, é atualizar o classificador constantemente com novos dados. A atualização constante do classificador ao longo do tempo corresponde a um sistema de *aprendizado incremental* (Giraud-Carrier, 2000). De maneira sucinta, um *classificador incremental* deve ter mecanismos que possibilitem a atualiazação do classificador com novos dados, durante a fase de aplicação e, também, mecanismos que possibilitem ao classificador livrar-se de dados antigos que não mais serão usados. Um classificador que não possui tais mecanismos de atualização é chamado *estático*, como é o caso do algoritmo KAOG. No que segue, são apresentados os objetivos e as motivações do trabalho, seguidos da organização do restante da tese.

1.1 Objetivos e motivações

Como exposto anteriormente, o aprendizado de máquina baseado em grafos constitui uma área bem estabelecida em domínios de aprendizado não supervisionado e semissupervisionado. Nesses tipos de problema, o grafo é usado para modelar os dados, de modo que, as estruturas presentes em sua distribuição sejam realçadas. Esta representação facilita as tarefas de agrupamentos ou de atribuição de rótulos, que são realizadas de acordo com a densidade e ao padrão de conexões entre os vértices do grafo. No entanto, o uso de grafos não tem sido devidamente estudado em problemas de aprendizado supervisionado. Em vista dos benefícios da representação de dados por meio de um grafo, da deficiência de métodos supervisionados baseados em grafos na literatura, bem como do sucesso do uso

de grafos em agrupamento de dados e na classificação semissupervisionada. *Este trabalho tem como objetivo explorar a classificação de dados em grafos. Especificamente, propõe dois algoritmos baseados em grafo para classificação de dados em domínios de dados com distribuição estacionária e não estacionária.*

Como mencionado, a classificação de dados é importante nas mais variadas áreas da ciência, tendo sido usada com sucesso em diversos domínios de aplicações. Dado a grande variedade de métodos existentes para a classificação de dados, pode-se questionar a necessidade do desenvolvimento de novas abordagens. No entanto, novos domínios de aplicações têm surgido e, em muitos casos, as técnicas convencionais de classificação não apresentam resultados satisfatórios. Além disso, o uso de grafos na representação dos dados pode proporcionar as seguintes vantagens em relação às técnicas tradicionais, por exemplo, 1) técnicas tradicionais apresentam dificuldade em tratar dados com distribuição arbitrária. O uso de grafos na representação dos dados permite representar classes com qualquer distribuição; 2) a grande maioria das técnicas de classificação utilizam uma única representação dos dados - geralmente na forma vetorial, e não consideram outras relações como a relação topológica entre os dados. O algoritmo proposto neste trabalho utiliza relações entre os dados, dispostos em um grafo, chamado grafo K-associado ótimo, bem como a informação de pureza, que quantifica o nível de mistura local de cada componente; 3) muitos algoritmos de aprendizado dependem de um ou mais parâmetros, de modo que, de posse de um determinado domínio de dados, o ajuste dos parâmetros deve ser feito com o fim de selecionar um, dentre os modelos de classificadores obtidos - o que, geralmente, implica em alto custo computacional. O uso de grafos na representação dos dados possibilitou o desenvolvimento de um algoritmo não paramétrico, o que consiste em uma propriedade interessante, pois evita a fase de seleção de modelos, tornando-o uma opção atrativa na prática.

Pesquisas adicionais com o classificador estático mostratram que a representação dos dados por meio do grafo K-associado, juntamente com a pureza, oferece a possibilidade de desenvolvimento de um método incremental para a classificação de dados não estacionários. Neste tipo de aplicação, os dados (rotulados e não rotulados) são apresentados continuamente, como um fluxo de dados, o que impõe ao sistema classificador restrições de tempo e de memória. No primeiro caso o classificador deve processar a nova instância em tempo hábil para evitar o acúmulo de exemplos de dados. Intuitivamente o classificador não deve levar mais tempo para processar um novo exemplo do que a média de tempo de chegada de exemplos. A restrição de memória está relacionada ao crescimento do uso da memória por parte do sistema classificador, e implica em como o classificador lida com os exemplos antigos. A preocupação com a memória diz respeito ao desempenho do classificador ao longo do tempo, a utilização desregrada da memória além de ser, por si só, um empecilho em um sistema computacional, também impacta diretamente no aumento no tempo de processamento do classificador. Note que as restrições de tempo e memória são interdependentes e precisam ser tratadas em conjunto para que o sistema classificador

apresente um comportamento estável. Por exemplo, o aumento no tempo de classificação implica em uso de memória auxiliar para armazenar os dados recém-chegados, que impacta na utilização da memória; o aumento no uso de memória, por sua vez, resulta em maior tempo no processamento de novos dados. De forma que, um algoritmo incremental precisa de mecanismos para lidar com os problemas de tempo e de memória. No primeiro caso, o método proposto para classificação incremental, apesar de ser baseado em instâncias, apresenta a possibilidade de lidar com componentes ao invés de instâncias isoladas, possibilitando ao algoritmo armazenar grande quantidade de dados, podendo acessá-los como componentes, o que torna o processo de classificação mais rápido. Para evitar o uso excessivo da memória, o classificador proposto, baseia-se na atualização do grafo, que é realizada por meio da inserção e remoção de componentes, de modo que o grafo utilizado pelo classificador apresente sempre tamanho relativamente constante.

1.2 Organização do trabalho

Com o objetivo de contextualizar o trabalho em meio às diversas abordagens que utilizam grafos no aprendizado de máquina, o Capítulo 2 apresenta trabalhos relacionados e conceitos que serão usados no desenvolvimento dos demais capítulos. O Capítulo 2 inicia com uma revisão dos conceitos da teoria dos grafos que serão usados no trabalho, em seguida, a Seção 2.2 aborda o aprendizado de máquina sob a concepção de métodos que utilizam grafos. Nesta seção são apresentados métodos de construção do grafo a partir de um conjunto de dados vetoriais do tipo atributo-valor. Bem como, uma breve exposição de métodos baseados em grafo no aprendizado não supervisionado - utilizados no agrupamento de dados. Com relação a métodos do aprendizado semissupervisionado, serão expostas três categorias desses métodos, a saber, iterativos, de caminhada aleatória e baseados em regularização. A Seção 2.2 é encerrada com uma concisa exposição sobre o aprendizado supervisionado que, apesar da existência de poucos métodos, a ênfase é dada em relação a métodos baseados em *kernel* em grafos. O Capítulo 2 também revisa conceitos da classificação supervisionada de dados, dividido em duas abordagens quanto à estacionariedade da distribuição dos dados. Na Seção 2.3 é revisado o tipo mais comum de classificação, a classificação de dados com distribuição estacionária. Esta seção enfatiza alguns problemas inerentes a este tipo de aplicação, como a seleção de modelos e o aprendizado baseado em comitês. Também são expostos alguns dos principais algoritmos de aprendizado, dos quais, alguns deles são usados como comparativo para o algoritmo proposto no Capítulo 3. O outro tipo de classificação, abordado neste trabalho, refere-se à problemas com distribuição de dados não estacionária e é apresentada na Seção 2.4. Nesta seção, além de apresentar os principais conceitos da área, como mudança de conceito e aprendizado incremental, também são apresentados alguns algoritmos do estado-da-arte e que serão usados para efeito de comparação no Capítulo 4 na qual é apresentada a versão incremental para o algoritmo proposto.

O Capítulo 3 apresenta o algoritmo baseado em grafo aplicável a problemas de classificação de dados estácionarios, nomeado KAOG. O algoritmo proposto baseia-se em um grafo chamado grafo K-associado ótimo, sendo este resultante de um processo iterativo de escolha dos melhores componentes de vários grafos K-associados. Dessa forma o capítulo inicia apresentando o grafo K-associado, bem como algumas das propriedades desta estrutura, como o cálculo da pureza para componentes. Em seguida é apresentado o método de construção do grafo ótimo, que é usado pelo classificador para estimar a probabilidade de um novo vértice conectar-se a cada um dos componentes do grafo. O processo de classificação, por sua vez, é exposto na Seção 3.2, que além de apresentar como o grafo ótimo é usado na classificação de novos dados, apresenta um exemplo de classificação seguido por um estudo de caso que considera a superfície de decisão do algoritmo KAOG em comparação com a superfície de decisão apresentada pelo KNN. A complexidade computacional para o algoritmo proposto é estimada na Seção 3.3. Finalizando o capítulo, a Seção 3.4 apresenta alguns exemplos ilustrando o comportamento da pureza nos grafos, K-associado e ótimo, bem como, um exemplo da construção de um grafo ótimo. A seção também apresenta os resultados comparativos em quinze domínios de dados, considerando três níveis de ruído, resultando em 45 domínios. Ao final da seção é realizada uma análise estatística dos resultados com o objetivo de deixar claro as reais contribuições do algoritmo proposto.

O Capítulo 4 descreve o algoritmo incremental KAOGINC, desenvolvido como uma extensão do algoritmo estático KAOG. O algoritmo incremental é apropriado para domínios em que os dados são apresentados ao longo do tempo, e possam apresentar mudanças de conceito. Por meio de mecanismos de adição e remoção de componentes do grafo, o algoritmo KAOGINC mantém o bom desempenho de classificação, mesmo sob mudanças de conceito. Neste tipo de algoritmo o custo computacional é de extrema importância, a complexidade computacional do algoritmo é apresentada na Seção 4.2. A Seção 4.3 apresenta uma comparação entre os algoritmos propostos neste trabalho, KAOG e KAOGINC, em situações que ocorrem mudanças de conceito, estes exemplos tem o objetivo de elucidar a importância do aprendizado incremental neste tipo de aplicação. Diferente da versão estática, o algoritmo KAOGINC possui um parâmetro, que é analisado quanto ao seu comportamento na Seção 4.4. Terminando o capítulo, a Seção 4.5 apresenta alguns resultados comparativos em dez domínios não estacionários, bem como uma breve análise para cada um deles. Por fim, o Capítulo 5 conclui o trabalho com uma análise geral dos resultados obtidos para ambos os algoritmos propostos, bem como, aponta as principais contribuições e direciona futuras pesquisas.

Capítulo 2

Revisão de conceitos e trabalhos relacionados

Neste capítulo são revisados alguns conceitos necessários para o desenvolvimento do trabalho, também, para efeito de contextualização, serão brevemente apresentados alguns trabalhos que relacionam grafos no aprendizado de máquina. O capítulo também apresenta os algoritmos de aprendizado utilizados na comparação dos resultados dos algoritmos propostos. No que segue a Seção 2.1, apresenta a revisão de alguns conceitos básicos da teoria dos grafos, seguido pela Seção 2.2 que apresenta, de forma sucinta, alguns dos métodos de aprendizado de máquina que utilizam grafos. A Seção 2.3 descreve classificação de dados de maneira geral, abordando alguns dos principais problemas envolvidos e enfatizando os problemas de aprendizado em comitês e de seleção de modelos. Nesta seção também são apresentados alguns métodos de classificação estacionária. Por fim, a Seção 2.4 apresenta a classificação de dados não estacionários, aprendizado em tempo real e incremental, bem como os algoritmos usados na comparação do algoritmo proposto no Capítulo 4.

2.1 Revisão de conceitos da teoria dos grafos

Nesta seção são revisados os conceitos da teoria dos grafos que são utilizados ao longo do trabalho. Os conceitos aqui descritos podem ser encontrados em várias referências da área, tais como, (Diestel, 2006), (West, 2000), (Gross, 2005), (Bollobás, 1998), entre outras. No que segue, considere as seguintes definições:

Grafo: Um *grafo* $G = (V, E)$ é formado por um conjunto de vértices $V = \{v_1, \ldots, v_N\}$ e um conjunto de arestas E que conectam pares de vértices. Um grafo pode ser *direcionado* ou *não direcionado*, no primeiro caso, também chamado de *dígrafo*, toda aresta tem uma origem e um destino. Uma aresta com origem $v_i \in V$ e destino $v_j \in V$ é representada por um par ordenado (v_i, v_j). Nos grafos não direcionados, as arestas não têm direção

e uma aresta que conecta o vértice $v_i \in V$ ao vértice $v_j \in V$ é representada por $e_{ij} = \{v_i, v_j\}$. Grafos que possuem arestas direcionadas e não direcionadas são chamados de grafos *mistos*.

Multigrafo: Em ambos os tipos de grafos, direcionados e não direcionados, é permitido que o conjunto de arestas E contenha várias instâncias da mesma aresta, i.e., o conjunto de aresta E é considerado um multiconjunto. Se uma aresta ocorre muitas vezes em E, as cópias da aresta são chamadas *arestas paralelas*. Grafos com arestas paralelas são chamados *multigrafos*. Quando toda aresta ocorre somente uma vez no conjunto E, o grafo é simples, i.e. não possui arestas paralelas.

Grafos com peso:. Muitas vezes é util associar valores numéricos (pesos) a arestas ou vértices de um grafo. Pesos em arestas podem ser representados por uma função $f : E \to \mathbb{R}$ que atribui a cada aresta $e \in E$ um peso $f(e)$. Dependendo do contexto, os pesos podem descrever diversas propriedades, como custo (de tempo ou distância), capacidade, força de interação ou similaridade, entre outras. Um grafo sem peso equivale-se a um grafo com peso unitário, i.e. $f(e) = 1$ para toda aresta $e \in E$.

Subgrafos: Um grafo $G' = (V', E')$ é um *subgrafo* do grafo $G = (V, E)$ se $V' \subseteq V$ e $E' \subseteq E$ e G é *supergrafo* de G'. Um *subgrafo induzido por aresta* corresponde a um grafo formado por um subconjunto de aresta do grafo G junto com os vértices conectados a essas arestas. Um *subgrafo induzido por vértice* (ou simplesmente subgrafo induzido) refere-se a um grafo formado por um subconjunto dos vértices juntamente com as arestas cujos vértices que as conectam pertencem a este subconjunto.

Adjacência: Dois vértices v_i e v_j são *adjacentes* ou *vizinhos* se existir a aresta e_{ij}. Duas arestas $e_1 \neq e_2$ são adjacentes se elas conectarem um vértice em comum. Se todo vértice em G for adjacente a todos os demais, então G é *completo*. Um subgrafo induzido e completo é chamado *clique*. O cojunto de vizinhos do vértice v_i é denotado por Λ_{v_i}, sendo N_{v_i} o número de vizinhos de v_i. Quando as arestas envolvem relação de similaridade (e.g. proximidade), pode-se ordenar os vizinhos em relação à proximidade, o conjunto dos K-vizinhos mais próximos de v_i é chamado *K-vizinhaça* de v_i e notado por $\Lambda_{v_i,K}$.

Grau: O *grau* de um vértice v_i em um grafo não direcionado $G = (V, E)$, denotado por g_i, corresponde ao número de arestas em E que possuem v_i como um vértice adjacente. Se G é um multigrafo, arestas paralelas são contadas de acordo com sua multiplicidade no conjunto E. Um vértice com grau 0 é *isolado*. A *média do grau* quantifica globalmente o que é medido localmente pelo grau dos vértices, e pode ser calculada como na

Equação (2.1).

$$\langle G \rangle = \frac{1}{|V|} \sum_{v_i \in V} g_i \tag{2.1}$$

Na equação, $\langle G \rangle$ corresponde a média do grau no grafo G e $|V|$ ao número de vértices do grafo. Quando somados, os graus dos vértices no grafo, cada vértice é contado duas vezes, de forma que, o número de arestas, $|E|$, pode ser calculado como na Equação (2.2).

$$|E| = \frac{1}{2} \sum_{v_i \in V} g_i = \frac{1}{2} \langle G \rangle |V| \tag{2.2}$$

E, dessa forma, pode-se relacionar as quantidades, média do grau, número de vértices e número de arestas como: $\langle G \rangle = 2(|E|/|V|)$.

Caminhadas, caminhos e ciclos: Uma *caminhada* de um vértice v_0 para o vértice v_k, no grafo $G = (V, E)$, corresponde a uma sequência $v_0, e_1, v_1, e_2, v_2, \ldots, v_{k-1}, e_k, v_k$ de vértices e arestas. O comprimento de uma caminhada é definido pelo número de arestas percorridas na caminhada. Uma caminhada é chamada *caminho* se $e_i \neq e_j$ para $i \neq j$, e um caminho é simples se $v_i \neq v_j$ para $i \neq j$. Um caminho com $v_0 = v_k$ é um *ciclo*; um ciclo é simples se $v_i \neq v_j$ para $0 \leq i < j \leq k - 1$. Um grafo acíclico é chamado *floresta*, uma floresta conectada é chamada *árvore*.

Componentes: Um grafo não direcionado $G = (V, E)$ é *conectado* se todo vértice puder ser alcançado a partir de qualquer outro vértice, i.e., se existir um caminho de todo vértice para qualquer outro vértice no grafo. Um grafo que consiste de apenas um vértice é considerado conectado. Grafos que não são conectados são chamados *desconectados*. Para um grafo desconectado $G = (V, E)$, um *componente* conectado de G é um subgrafo induzido $G' = (V', E')$ conectado e maximal (i.e. não existe um subgrafo $G'' = (V'', E'')$ com $V'' \supset V'$). Para verificar se um grafo é conectado e encontrar todos os componentes conectados utiliza-se a busca em largura ou em profundidade, com complexidade da ordem de $O(|V| + |E|)$. Grafos em que o número de arestas é linearmente comparável ao número de vértices são chamados *esparsos*, i.e. $|V| = O(|E|)$.

2.2 Aprendizado de máquina baseado em grafos

Os algoritmos baseados em grafos, em sua maioria, baseiam-se nas suposições de continuidade (*smoothness assumption*) em relação à distribuição dos dados e suas classes. Esta supõe que, se duas instâncias $\mathbf{x}_1$, $\mathbf{x}_2$ são próximas, então suas saídas correspondentes y_1, y_2 devem ser próximas. Bem como na suposição de separação por baixa densidade (*low-density assumption*), a qual, implica que a densidade deve ser baixa em regiões próximas a outras classes. Note que a primeira suposição é geral e serve para regressão e classificação e a segunda corresponde a uma variante da suposição de agrupamento (se

duas instâncias de dados estão no mesmo agrupamento, elas provavelmente pertencem à mesma classe). A suposição de suavidade está relacionada com o significado de proximidade entre $\mathbf{x}_1$ e $\mathbf{x}_2$, de modo que, esta pode ser incorporada à função de similaridade, usada para construir o grafo e dar pesos às arestas, e consequentemente, à matriz de adjacência $\mathbf{W}$.

2.2.1 Formação do grafo a partir de um conjunto de dados

Segundo Zhu (2008), a representação de um conjunto de dados como grafo, apesar de ser pouco estudada na literatura, consiste em um dos passo mais importantes no aprendizado baseado em grafos. Esta seção revisa algumas técnicas de formação de grafo a partir de um conjunto de dados $X = \{\mathbf{x}_1 \dots \mathbf{x}_N\}$, onde cada instância $\mathbf{x}_i = (x_{i1}, \dots, x_{ip}, y_i)$ é formada por p atributos e possui um rótulo associado $y_i \in \Omega = \{\omega_1, \dots \omega_\mu\}$ que distingue uma entre as μ classes. No geral a construção de um grafo a partir de um conjunto de dados do tipo atributo-valor consiste na abstração dos dados em vértices e das relações entre eles em arestas. Construir um grafo a partir dos dados de entrada consiste na fase crítica do aprendizado baseado em grafos. Trata-se de uma tarefa fundamental não apenas considerando o aprendizado não supervisionado, como o agrupamento de dados, mas também nos aprendizados semissupervisionado e supervisionado.

A Figura 2.1, ilustra alguns métodos para a construção de grafos, considerando os vértices na Figura 2.1(a) v_i, v_j e v_l como os vértices abstraídos dos exemplos de dados $\mathbf{x}_i$, $\mathbf{x}_j$ e $\mathbf{x}_l$, respectivamente. Dentre as técnicas mais estudadas para construção de grafos encontram-se as técnicas baseadas em relações de vizinhança, tais como grafos-ϵ e grafos KNN. No caso de grafos-ϵ, a conexão entre pares de vértices é feita considerando um raio pré-definido como limitante, ou seja, qualquer par de vértices v_i e v_j cuja distância dos exemplos $d(\mathbf{x}_i, \mathbf{x}_j) < \epsilon$ é conectado, como ilustra a Figura 2.1(b). Uma abordagem similar considera o conjunto de dados inicialmente como um grafo totalmente conectado, no qual as arestas recebem pesos de acordo com alguma função de similaridade, por exemplo, a função Gaussiana, como mostra a Equação (2.3).

$$w_{ij} = \frac{e^{(-\|\mathbf{x}_i - \mathbf{x}_j\|^2)}}{2\sigma^2}. \tag{2.3}$$

Na qual w_{ij} corresponde ao peso da aresta e_{ij} e o parâmetro σ, de forma análoga ao parâmetro ϵ nos grafos-ϵ, controla o tamanho da vizinhança, portanto, quanto maior o valor de σ menor a vizinhança. Este tipo de construção permite o aprendizado por meio de uma função diferenciável dos pesos (von Luxburg, 2007).

Nos grafos KNN, o parâmetro K é usado para especificar o número de vizinhos de cada vértice que serão conectados. Esta definição implica na construção de um dígrafo no qual uma aresta dirigida com origem no vértice v_i terá como destino o vértice v_j, se este estiver entre os K vizinhos de v_i, como mostra a Figura 2.1(c). Como as relações de similaridade são simétricas, no geral, usam-se grafos não dirigidos para modelagem de dados. Dessa

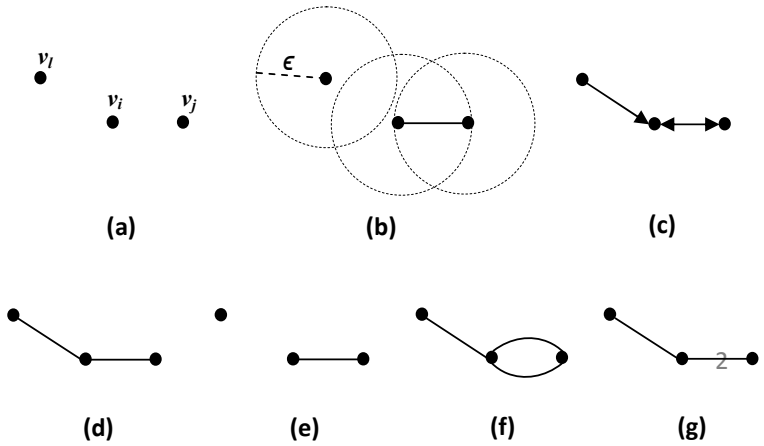

Figura 2.1: Exemplos de criação de grafos a partir de dados. Para a geração dos grafos (c)-(g) foi considerado $K = 1$.

forma, precisa-se abstrair as direções no grafo KNN dirigido previamente citado. Existem pelo menos três maneiras de se fazer isso, a primeira consiste, simplesmente, em ignorar as direções das arestas, tornando-as não dirigidas (Figura 2.1(d)). Outra alternativa é considerar apenas as arestas do dígrafo que possuem duas direções, ou seja, conectar o vértice v_i com o vértice v_j se v_i encontra-se na K-vizinhança de v_j e vice-versa; este tipo de construção chama-se grafo KNN mútuo e é mostrado na Figura 2.1(e). A terceira forma, usada na construção do grafo K-associado apresentado no Capítulo 3, consiste em considerar toda aresta como não dirigida e manter duas arestas quando o caso mútuo ocorrer, resultando em um multigrafo, como mostra a Figura 2.1(f); a Figura 2.1(e) ilustra outra maneira de notar esta estrutura.

Outras maneira de construir o grafo a partir dos dados, apresentadas por Zhu (2005), correspondem ao uso da tangente hiperbólica, na qual o peso da aresta e_{ij} é dado por $w_{ij} = tanh(\alpha_1(d(\mathbf{x}_i, \mathbf{x}_j) - \alpha_2) + 1)/2$. Onde α_1 e α_2 são parâmetros que controlam a inclinação e o raio de atuação, respectivamente. A ideia é criar uma conexão para cada exemplo, onde os pesos decaem suavemente enquanto a distância em relação ao padrão aumenta. Comparado aos grafos-ϵ, os grafos ponderados pela tangente hiperbólica são contínuos em relação aos pesos e também permitem aprendizado utilizando métodos de gradiente. Outra opção é atribuir os pesos utilizando a função exponencial, $w_{ij} = exp(d(\mathbf{x}_i, \mathbf{x}_j)^2/\alpha^2)$, também trata-se de um esquema contínuo de ponderação, com decaimento dado por α, mas com raio de atuação menos claro que na função hiperbólica.

2.2.2 Aprendizado não supervisionado baseado em grafos

Talvez a área de pesquisa mais ativa no que diz respeito ao uso de algoritmos baseados em grafos em aprendizado de máquina seja no aprendizado não supervisionado, principalmente o agrupamento de dados (Schaeffer, 2007). No agrupamento de dados o objetivo é agrupar os dados, de modo que, elementos de um mesmo grupo tenham mais características em comum entre si do que elementos de grupos distintos (Jain *et al.*, 1999).

Como mencionado por Jain e Dubes (1998) o processo de agrupamento de dados pode ser visto como um problema de aprendizado não supervisionado. Situações como estas são frequentes nas mais diversas áreas de pesquisa, tais como bioinformática (Jiang *et al.*, 2004), agrupamento de documentos da Web (Boley *et al.*, 1999), caracterização de grupos de clientes baseada em padrões de compra (Jank e Shmueli, 2005), identificação de material cerebral em imagens de MRI (Taxt e Lundervold, 1994), entre outros.

A maioria dos algoritmos de agrupamento de dados baseia-se em modelos estáticos que, embora eficientes em alguns casos, oferecem resultados pouco satisfatórios quando o conjunto de dados apresenta grupos de dados com formas, densidades e tamanhos variados. Isso se deve ao fato de que, esse tipo de algoritmo não se baseia na natureza individual de cada agrupamento enquanto a divisão acontece, uma vez que uma característica dominante diferente pode ser evidenciada para cada grupo. Por exemplo, técnicas de agrupamento baseadas em partição, tais como *K-means* (MacQueen, 1967) e Clarans (Ng e Han, 2002), tentam particionar o conjunto de dados em K agrupamentos, de forma a otimizar um critério previamente estabelecido. Nesses algoritmos é assumido que os agrupamentos tenham a forma de hiperelipses com tamanhos similares. De forma que fica impossível evidenciar grupos de dados que variam em tamanho, como mostrado na Figura 2.2(a); ou que variam em formato, como mostrado na Figura 2.2(b).

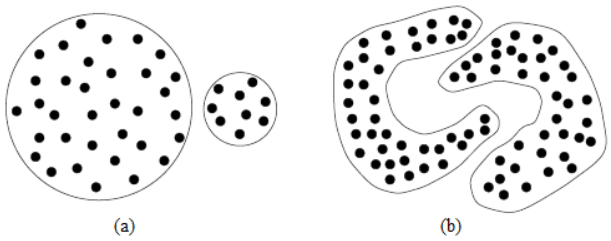

Figura 2.2: Agrupamentos de dados com formatos diferentes. (a) Hiperelipses (b) Irregulares (Karypis *et al.*, 1999).

Algoritmos baseados em grafos unem duas características desejadas: detecção de agrupamentos com formas variadas e possibilidade de representação hierárquica. Devido à representação em grafos, esses algoritmos buscam a estruturação topológica entre os dados de entrada e, consequentemente, são capazes de identificar formas de agrupamentos variados. Algoritmos que utilizam grafos para o agrupamento de dados transformam o problema de agrupamento em um problema de particionamento do grafo, o qual consiste em dividir o grafo em subgrafos disjuntos de modo a minimizar alguma função de custo associado ao corte das arestas (Li e Tian, 2007). De maneira geral, o processo de agrupamento utilizado por esses algoritmos acontece em duas etapas: modelagem dos dados em um grafo esparso, no qual os vértices representam os dados e as arestas representam a similaridade; e particionamento do grafo para a formação dos agrupamentos. O processo é ilustrado na Figura 2.3.

Encontrar a solução ótima para o problema de particionamento do grafo em subgrafos de tamanhos similares minizando o corte de aresta, no entanto, é um problema

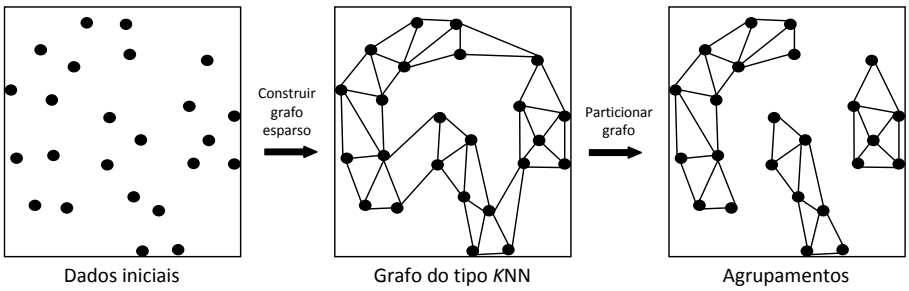

Figura 2.3: Etapas utilizadas por algoritmos baseados em grafos no agrupamento de dados.

NP-completo devido sua natureza combinatória (Schaeffer, 2007). De modo que, para realizar o particionamento de grafos grandes, alguns métodos heurísticos foram propostos, tais como os algoritmos Chamaleon (Karypis *et al.*, 1999) e CURE (Guha *et al.*, 1998). Mesmo considerando o uso de heurísticas no particionamento do grafo, o desempenho deste tipo de método fica comprometido quando o conjunto de dados é muito grande. Recentemente, as redes complexas despontaram como um poderoso método de representação que tem se mostrado adequado para representar grandes quantidades de dados e suas diversas relações (Albert e Barabási, 2002), (Newman, 2003). O termo redes complexas refere-se às redes que possuem topologia não trivial, dado que essas redes podem ser compostas por grande quantidade de vértices (milhões ou até bilhões). Muitas redes existentes na natureza, bem como na sociedade, podem ser caracterizadas como redes complexas; alguns exemplos são: a Internet, a *World Wide Web*, a rede neural biológica, redes sociais entre indivíduos e entre companhias e organizações, cadeias alimentares, redes do metabolismo e de distribuição como a corrente sanguínea, as rotas de entrega postal e a distribuição de energia elétrica, entre outras (Strogatz, 2001).

Uma característica notável em diversas redes complexas, encontradas na natureza ou construídas pelo homem, é a presença de estruturas locais conhecidas como *comunidades*. Uma comunidade é definida como um grupo de vértices densamente conectados, enquanto as conexões entre diferentes comunidades são esparsas (Newman e Girvan, 2004). Essas comunidades representam padrões de interação entre vértices da rede; sua identificação é crucial na análise e no entendimento dos mecanismos de crescimento e formação da rede, bem como nos processos que ocorrem na rede (Clauset, 2005). De maneira geral, a estrutura em comunidades revela similaridade, segundo algum critério, por meio de conexões entre os vértices pertencentes a um mesmo grupo, por exemplo, em redes sociais a presença de comunidades pode revelar grupos de indivíduos com mesmos interesses, relações de amizade, relações profissionais, etc. A identificação de comunidades em redes complexas é, portanto, um modelo de agrupamento de dados dispostos em forma de rede. Segundo Reichardt e Bornholdt (2006), a abordagem na qual a informação é codificada em vértices e a proximidade/afinidade é definida por meio das arestas, aplica-se diretamente às redes complexas. Considerando este cenário pode-se dizer que as redes complexas são apropriadas para a análise de dados em geral, especialmente para grandes bases de dados.

Diversos algoritmos foram desenvolvidos para identificar comunidades em redes complexas (Danon *et al.*, 2005), alguns dos principais serão apresentados a seguir.

Dentre os principais estão, o método proposto por Newman e Girvan (2004), que se baseia no conceito de *betweenness* de aresta. Esta quantidade, também definida para vértices, corresponde ao número de caminhos mais curtos entre qualquer par de vértice que passam pela aresta e_{ij}. A cada passo, a aresta de maior *betweenness* é removida, dividindo a rede em comunidades. O princípio em que este método baseia-se é simples, como o número de arestas conectando duas comunidades é pequeno, todos os caminhos que ligam vértices em comunidades diferentes devem passar por essas arestas. Assim sua participação na composição dos caminhos que ligam as duas comunidades é alta e o encerramento iterativo destas conexões favorece uma forma de identificar as comunidades. Um outro método de identificação de comunidades, desenvolvido por Zhou (2003a), baseia-se no conceito de movimento Browniano. Neste método, uma partícula Browniana começa sua trajetória em um vértice da rede, escolhendo o próximo vértice de maneira aleatória. As informações obtidas pela partícula são utilizadas para medir a distância entre os vértices, definindo as comunidades que compõem a rede e a estrutura de cada uma. O método foi estendido posteriormente em (Zhou, 2003b) com a definição de um índice de dissimilaridade entre vértices vizinhos, considerando a matriz de distâncias. O índice de dissimilaridade indica até que ponto dois vértices vizinhos devem pertencer à mesma comunidade, e pode ser usado na decomposição hierárquica da rede em comunidades. O método desenvolvido por Newman (2004) sugere um algoritmo que otimize o valor da modularidade, medida usada para qualificar determinada partição da rede em comunidades. Trata-se de um algoritmo de agrupamento hierárquico, onde inicialmente cada vértice da rede é considerado uma comunidade, e repetidamente, as comunidades são agrupadas em pares, maximizando o índice de modularidade. É importante ressaltar que apenas comunidades que possuem arestas ligando seus vértices podem ser possivelmente unidas pelo algoritmo, o que limita a um máximo de $|E|$ pares de comunidades, onde $|E|$ é o número de arestas da rede.

O método proposto por Boccaletti *et al.* (2007) utiliza sincronização para encontrar comunidades na rede. O método consiste em associar um oscilador a cada vértice da rede. Em seguida, valores iniciais para os parâmetros são dados de maneira que toda a rede fique sincronizada, então, como no processo de resfriamento simulado (*simulated annealing*), enquanto os parâmetros são afrouxados a sincronização total se desfaz em grupos sincronizados, revelando as comunidades. O método é baseado em um fenômeno de sincronização de comunidades utilizando osciladores de fase não idênticos (Boccaletti *et al.*, 2002), cada um associado a um vértice da rede e interagindo através das arestas da rede. Grupos de osciladores sincronizados representam um regime intermediário entre travamento de fase global e completa abstenção de sincronização, implicando em uma divisão da rede em grupos de vértices que oscilam na mesma frequência média. A ideia principal consiste em iniciar a rede com todos os elementos acoplados, e por meio de mudanças

dinâmicas nos pesos das interações entre vértices, obter agrupamento hierárquico e progressivo que detecta totalmente os módulos (comunidades) presentes na rede. Zhao *et al.* (2008) utilizaram uma rede de osciladores acoplados para segmentação de imagem. O método desenvolvido utiliza qualquer topologia de rede, não necessariamente um reticulado.

Os métodos previamente mencionados dependem somente da estrutura da rede para encontrar comunidades, pois nenhuma outra informação é disponível. Uma abordagem complementar desenvolvida por Reichardt e Bornholdt (2004), combina a ideia de utilizar uma modificação do modelo Hamiltoniano de Ising no particionamento de grafos, proposto por Fu e Anderson (1986), e o recente método de agrupamento de dados baseado no modelo de Potts para dados multivariados, proposto por Blatt *et al.* (1996). Basicamente Reichardt e Bornholdt (2004) estenderam o modelo de Blatt *et al.* (1996) para detecção de comunidades em redes complexas. O método é baseado na analogia a um modelo da mecânica estatística, o modelo ferromagnético de Potts. A ideia principal é mapear as comunidades de uma rede em mínimos locais dos domínios magnéticos, modelando um conjunto de elementos dotados de *spins* e dispostos em um reticulado.

2.2.3 Aprendizado semissupervisionado baseado em grafos

No aprendizado semissupervisionado considera-se que poucas instâncias possuem rótulos de classe, dentre as instâncias disponíveis. Portanto, no contexto semissupervisionado, pode-se dividir o conjunto de dados X em dois subconjuntos, um conjunto com dados rotulados $X_l = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\}$, e um com os dados não rotulados $X_u = \{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_N\}$, sendo que, na maioria dos casos têm-se $|X_u| >> |X_l|$. De maneira que, pode-se definir os vetores $\mathbf{y}_l = (y_1, \ldots, y_l)$, com os rótulos das instâncias pertencentes ao conjunto X_l, bem como o vetor $\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_N)$, como o vetor de rótulos dos dados, atribuídos por algum algoritmo. Uma vez construído, um grafo baseado nas relações de similaridade entre as instâncias do conjunto X, o problema de aprendizado semissupervisionado, consiste em encontrar um conjunto de rótulos para os dados não rotulados, de modo que sejam consistentes com os rótulos iniciais e com a geometria do grafo induzido (representado pela matriz de pesos $\mathbf{W}$). A matriz $\mathbf{W}$ está intimamente relacionada à Laplaciana do grafo, sendo esta responsável por representar a estrutura na qual os dados estão distribuídos.

Dentre as muitas maneiras de definir a Laplaciana do grafo, as mais utilizadas no aprendizado semissupervisionado são a Laplaciana normalizada $\mathcal{L}$, e a não normalizada $\mathbf{L}$, como mostradas nas Equações (2.4) e (2.5).

$$\mathcal{L} = \mathbf{I} - \mathbf{D}^{1/2}\mathbf{W}\mathbf{D}^{1/2} \tag{2.4}$$

$$\mathbf{L} = \mathbf{D} - \mathbf{W} \tag{2.5}$$

Onde a matriz de adjacência, $\mathbf{W}$, representa o grafo com peso $G = (V, E)$, e cujo

valor de $\mathbf{W}_{ij}$ é dado por uma função de similaridade $f : E \rightarrow \mathbb{R}$, com a restrição de que $\mathbf{W}_{ij} = 0$ se $e_{ij} \notin E$. A matriz $\mathbf{I}$ corresponde à matriz identidade e a matriz diagonal $\mathbf{D}$, ou matriz do grau de G, é definida como $\mathbf{D}_{ii} = \sum_j \mathbf{W}_{ij}$.

A Laplaciana do grafo pode ser usada em todo tipo de aprendizado baseado em grafos, como no aprendizado não supervisionado, no agrupamento espectral (Filippone *et al.*, 2008); no aprendizado semissupervisionado, em métodos que utilizam regularização baseada em grafos (Belkin e Niyogi, 2003), e no aprendizado supervisionado, e na definição de *kernels* em grafos (Vishwanathan *et al.*, 2010).

Métodos baseados em grafos têm sido amplamente pesquisados no contexto de aprendizado semissupervisionado (ver, por exemplo, (Belkin e Niyogi, 2004), (Chapelle *et al.*, 2006), (Zhu, 2008), (Culp e Michailidis, 2008) e suas referências). Na literatura, a tarefa de transdução é extensamente mais considerada que a indução, devido basicamente às características atribuídas à Laplaciana do grafo que representa os dados. De acordo com Hein *et al.* (2007), o uso de grafos no aprendizado semissupervisionado é atrativo ao menos por dois motivos, 1) por meio da Laplaciana associada ao grafo, pode-se gerar o processo de difusão dos rótulos e, 2) o grafo age como um regularizador dos dados, adaptando-se às várias densidades e formatos dos dados. De maneira simplista, muitos dos algoritmos semissupervisionado baseados em grafos, encaixam-se em uma das três categorias, *iterativos*, *baseados em caminhada aleátoria* e *baseado em regularização em grafos*.

Dentre os algoritmos iterativos, está o algoritmo proposto por Zhu e Ghahramani (2002). O algoritmo baseia-se na simples ideia de propagar os rótulos, a partir de todo vértice, para seus vizinhos, até que todos os vértice sejam rotulados. Os rótulos são atualizados considerando a iteração, $\hat{\mathbf{y}}^{(t+1)} = \mathbf{D}^{-1}\mathbf{W}\hat{\mathbf{y}}^{(t)}$ mantendo inalterado os rótulos previamente conhecidos em $\hat{\mathbf{y}}$. De forma que, os rótulos na iteração $t+1$, $\hat{\mathbf{y}}^{(t+1)}$, dependem dos rótulos na iteração anterior, $\hat{\mathbf{y}}^{(t)}$, bem como da matriz diagonal $\mathbf{D}$ e de adjacência $\mathbf{W}$. Zhou *et al.* (2004), propõem utilizar a Laplaciana normalizada, $\mathcal{L}$, para, iterativamente, espalhar os rótulos, como na equação: $\hat{\mathbf{y}}^{(t+1)} = \alpha\mathcal{L}\hat{\mathbf{y}}^{(t)} + (1-\alpha)\hat{\mathbf{y}}^{(0)}$. Onde o parâmetro $\alpha \in [0, 1)$ pondera entre favorecer os rótulos atuais, determinados por relação de vizinhança, e os rótulos iniciais, dados por $\hat{\mathbf{y}}^{(0)}$.

Szummer e Jaakkola (2002) propuseram uma maneira alternativa de propagar os rótulos através do grafo, que consiste na caminhada aleatória de Markov, onde $\mathbf{W}$ pode ser associado à matriz de transição. De forma simplista, uma caminhada aleatória em um grafo pode ser considerada como uma cadeia de Markov, que significa que a probabilidade de visitar um vértice vizinho depende do peso da aresta entre os dois vértices, $\mathbf{P}_{ij} = \mathbf{W}_{ij} / \sum_k \mathbf{W}_{ik}$. Onde Szummer e Jaakkola (2002) usaram um *kernel* Gaussiano para determinar $\mathbf{W}_{ij}$ de acordo com os pesos dos vizinhos e $\mathbf{W}_{ii} = 1$. Zhou e Schölkopf (2004) propuseram propagar os rótulos no grafo, por meio da matriz de transição $\mathbf{P} = (1-\alpha)\mathbf{I} + \alpha\mathbf{D}^{-1}\mathbf{W}$, utilizando caminhada aleatória preguiçosa. Callut *et al.* (2008) propuseram outro tipo de caminhada, chamada caminha discriminativa (*D-walk*), que basicamente, consiste em uma medida de *betweenness* para vértices, determinado pelo número

de visitas durante a caminhada. Para cada classe é realizada uma caminhada que calcula uma medida de *betweenness* para cada vértice, os vértices sem rótulo recebem o rótulo referente à classe de maior *betweenness* para o vértice.

Por último, a categoria de métodos utiliza a estrutura do grafo como regularizador em algum tipo de função de custo, cuja minimização determina os rótulos. Dados os rótulos $\mathbf{y}_l$, a consistência entre com os rótulos iniciais e os atribuídos pode ser dado pela Equação (2.6).

$$\sum_{i=1}^{l}(\|\hat{y}_i - y_i\|)^2 = \|\hat{\mathbf{y}}_l - \mathbf{y}_l\|^2 \tag{2.6}$$

Note que, a Equação (2.6) não utiliza nenhuma informação do grafo, portanto, não incorpora informações sobre a geometria dos dados, corresponde somente a uma função de perda (*loss function*). Por outro lado, a consistência com a estrutura dos dados, pode ser inferida pela suposição de agrupamento, i.e. dados próximos entre si devem pertencer à mesma classe, dessa forma considere a Equação (2.7).

$$\frac{1}{2}\sum_{i,j=1}^{N}\mathbf{W}_{ij}(\hat{y}_i - \hat{y}_j)^2 = \hat{\mathbf{y}}^\top\mathbf{L}\hat{\mathbf{y}} \tag{2.7}$$

De fato, a combinação da regularização do grafo dada pela Equação (2.7) com uma função de perda como a mostrada na Equação (2.6), têm sido explorada por diversos autores na literatura. Um exemplos é o trabalho de Zhu *et al.* (2003), que utiliza campo aleatório Gaussiano (*Gaussian random field*) para atribuir os rótulos e minimizar a função de energia, mostrada na Equação (2.8).

$$E = \sum_{i=1}^{N}(\hat{y}_i - y_i)^2 + \frac{1}{2}\sum_{i,j=1}^{N}\mathbf{W}_{ij}(\hat{y}_i - \hat{y}_j)^2 \tag{2.8}$$

No entanto, no trabalho de Zhu *et al.* (2003) os dados originalmente rotulados são mantidos inalterados, o que pode ser uma desvantagem quando alguns deles correspondem a ruído. Dessa forma, pode ser interessante deixar o algoritmo rerrotular as instâncias previamente rotuladas, como no trabalho de Belkin *et al.* (2004), que além disso, utiliza uma função de energia com termo regularizador e função de perda um pouco diferente, como mostra a Equação (2.9).

$$E = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 + \hat{\mathbf{y}}^\top\mathbf{S}\hat{\mathbf{y}} \tag{2.9}$$

Na qual, $\mathbf{S} = \mathbf{L}^q$ com $q \in \mathbb{N}$, conhecida como *regularização de Tikhonov*. Nesse tipo de método, dados inicialmente sem rótulos são inicializados com rótulos de classe aleatórios. De fato, diversos algoritmos têm sido propostos considerando esta abordagem, alterando a função de perda, o termo regularizador ou o algoritmo de minimização, alguns exemplos são os trabalhos de Belkin *et al.* (2005), Joachims (2003), Tsang e Kwok (2007), entre outros. De maneira que, pode-se obter uma função de custo geral cuja minimização

resulta na propagação de rótulos, como mostrado na Equação (2.10).

$$C(\hat{\mathbf{y}}) = \|\hat{\mathbf{y}} - \mathbf{y}\|^2 + \delta\hat{\mathbf{y}}^\top\mathbf{L}\hat{\mathbf{y}} + \delta\epsilon\|\hat{\mathbf{y}}\|^2 \tag{2.10}$$

Na equação, $\delta = \alpha/(\alpha - 1)$, onde $\alpha \in (0, 1)$ e o termo $\delta\epsilon\|\hat{\mathbf{y}}\|^2$, com $\epsilon > 0$, corresponde a um termo regularizador que previne situações degenerativas, por exemplo, quando um grafo tem um componente sem rótulo Chapelle *et al.* (2006).

Métodos transdutivos, por definição, não são apropriados para serem usados na classificação de novos dados, ainda que, em alguns casos, possam ser adaptados para tal propósito. A maneira mais direta de utilizar um método transdutivo para realizar classificação de novos dados é, primeiro espalhar os rótulos para todo o conjunto inicial e depois manter a estrutura do grafo para classificar novos dados. No contexto de classificação de novos dados, entretanto, não se considera que os novos dados sejam incluídos no grafo, o que requereria redistribuir os rótulos a cada nova classificação. Ao invés disso, pode-se incluir o novo vértice apenas para classificá-lo e depois retirá-lo do grafo. Uma abordagem geral para implementar indução a partir de um método transdutivo, proposto por Delalleau *et al.* (2005), consiste em rotular, inicialmente, todos os exemplos não rotulados disponíveis, por meio de algum método transdutivo baseado em grafo. Então, a classe de um novo exemplo ainda não visto $\mathbf{z}$ pode ser inferida de acordo com a Equação (2.11).

$$\varphi(\mathbf{z}) = \frac{\sum_j f_W(\mathbf{z}, \mathbf{x}_j)y_j}{\sum_j f_W(\mathbf{z}, \mathbf{x}_j)} \tag{2.11}$$

Na equação, $\varphi(\mathbf{z})$ corresponde à classe estimada para o novo exemplo $\mathbf{z}$, e $\mathbf{x}_j$ e y_j correspondem a instâncias de treino e suas respectivas classes. f_W é a função associada à matriz de similaridade, $\mathbf{W}$, obtida do conjunto de treinamento. Dessa forma, diferentes funções, f_W, definem diferentes classificadores, por exemplo, se f_W correspode à função K vizinhos mais próximos, então o problema é reduzido à classificação utilizando o algoritmos KNN (Wang e Zhang, 2008).

Podem-se elencar alguns métodos baseados em grafos propostos para indução de classificadores considerando a abordagem semissupervisionada. Dentre os principais algoritmos estão o modelo proposto por Zhu e Lafferty (2005) que utiliza uma combinação de métodos geradores, como misturas de Gaussianas (*mixture of Gaussians*), com regularização proporcionada pela Laplaciana do grafo. A ideia é combinar as vantagens das duas abordagens, o modelo de mistura é usado para modelar os dados localmente enquanto que a estrutura global é modelada pelo grafo. Uma abordagem similar foi proposta por Krishnapuram *et al.* (2005), na qual além do uso do grafo como regularizador, usa-se o algoritmo *Expectation-Maximization* (EM) para treinar o classificador (Hastie *et al.*, 2009). O que o torna custoso computacionalmente devido ao uso dos algoritmos EM. Alguns algoritmos baseados em grafo dependem fortemente do conjunto de dados em questão, por exemplo, o método proposto por Sindhwani *et al.* (2005), consiste na definição de um *kernel* não paramétrico que é obtido a partir de um grafo que depende diretamente do dado

em questão e considera dados rotulados e não rotulados. Belkin *et al.* (2006) sugeriram uma abordagem geral, dependente do conjunto de dados em consideração, para criar uma regularização geométrica que pode ser utilizada para solucionar problemas de aprendizado não supervisionados, semissupervisionados e supervisionados. O esquema geral baseia-se em dois termos de regularização, um que controla a complexidade do classificador e outro a complexidade da distribuição dos dados. Chen *et al.* (2009a) propuseram uma abordagem multigrafo que consiste em um problema de otimização convexa que considera muitos grafos. Para um determinado conjunto de dados, cada grafo é associado a um *kernel*. Por meio desta abordagem multigrafo, uma função pode ser aprendida utilizando dados rotulados e não rotulados, através da minimização da função de regularização que modela a contradição entre erro e suavidade da função. A natureza multigrafo do método previne uma eventual representação ruim quando se utiliza apenas um grafo garantindo bom resultado de classificação.

2.2.4 Aprendizado supervisionado baseado em grafos

Nota-se, através das descrições apresentadas que, o uso de algoritmos baseados em grafos tem sido bastante estudado no âmbito do aprendizado de máquina, especialmente no que se refere aos aprendizados, não supervisionado e semissupervisionado. No entanto, no aprendizado supervisionado algoritmos baseados em grafos não têm recebido a mesma atenção, principalmente na solução de problemas apresentados como dados vetoriais. Presumivelmente, pode-se considerar algum método semissupervisionado (como os propostos em (Belkin *et al.*, 2006) (Sindhwani *et al.*, 2005) (Chen *et al.*, 2009a), por exemplo), para atuar na classificação supervisionada, dado um número razoável de instâncias rotuladas. O problema, neste caso, é que métodos semissupervisionados foram concebidos para incorporar conhecimento utilizando-se de dados rotulados e não rotulados e, na maioria dos casos, o grafo é usado justamente para modelar (regularizar) os dados em uma estrutura que possibilite a propagação dos rótulos. E, no geral a regularização por meio do grafo é custosa computacionalmente, dessa forma, se um conjunto com dados rotulados é disponibilizado, um método regular de classificação é preferível.

Um tipo relacionado de classificação que utiliza algoritmos baseados em grafos refere-se à classificação relacional, e é aplicado a dados relacionais. Este tipo de dado é naturalmente representado por grafos e difere-se de dados típicos, pois viola a premissa de instâncias independentes, o que significa que a classe de uma instância pode não depender somente de seus atributos, mas estar relacionada a outras instâncias ou a uma cadeia de relações entre várias instâncias (Macskassy e Provost, 2003). Aplicações que geram este tipo de dados incluem, reconhecimento de cadeias moleculares em expressões gênicas (Segal *et al.*, 2003), classificação de artigos científicos relacionados (Lu e Getoor, 2003), e de documentos e patentes (Chakrabarti *et al.*, 1998), e de páginas da *web* (Neville *et al.*, 2003), são alguns exemplos. Visando a unificação do uso de grafo na classificação relacional, Macskassy e Provost (2007) propuseram uma abordagem geral para adaptar-se à

diversas aplicações no que diz respeito a dados relacionais. Nesta abordagem constrói-se um modelo considerando três partes, a saber: um classificador indutivo local, que faz uso do conjunto de treinamento para estimar a distribuição de probabilidade das diferentes classes; um classificador relacional, que também visa estimar a distribuição de probabilidade, mas agora considerando relações de vizinhança em um grafo e; um classificador de inferência coletiva, i.e. capaz de classificar um conjunto de dados de uma vez, usado para refinar as predições realizadas pelos classificadores anteriores.

Outra abordagem, usada para estimar relações de similaridades entre vértices em um grafo ou entre dois grafos é conhecida por *kernel* de grafos (*graph kernel*), ver por exemplo (Vishwanathan *et al.*, 2010). De forma sucinta, este tipo de método utiliza uma função *kernel* para estabelecer uma medida de similaridade no grafo. A maior dificuldade neste tipo de abordagem esta justamente na definição de uma função *kernel* adequada à estrutura do grafo cujo cálculo seja eficiente. De qualquer maneira, este tipo de método foi proposto para ser aplicado a grafos, isso significa que pode ser usado em dados relacionais. Já considerando dados vetoriais representados como grafos, a classificação de um novo dado exigiria que este fosse incluído no grafo. O que é análogo ao problema de se usar um método semissupervisionado transdutivo para classificar uma nova instância, exigindo alto custo computacional.

2.3 Classificação de dados com distribuição estacionária

Na classificação de dados, tem-se como objetivo atribuir, a um dado, por meio de seus atributos, uma classe (ou categoria) dentre as classes pré-definidas. Neste contexto, por padrões entende-se relações, regularidades ou estruturas inerentes a alguns conjuntos de dados (Mitchell, 1997). Através da detecção de padrões significantes nos dados disponíveis, espera-se que um sistema classificador possa realizar predições com relação a dados ainda não classificados (Bishop, 2006). Um classificador é resultado de um processo de aprendizado supervisionado ou semissupervisionado indutivo, que utiliza um conjunto de dados com instâncias rotuladas para treinar (ou construir) o classificador (Duda *et al.*, 2001).

De acordo com Duda *et al.* (2001) alguns dos subproblemas inerentes à classificação são: 1) seleção de atributos, 2) falta valores de atributo, 3) ruído e *outlier*, 4) seleção de modelo, 5) complexidade 6) subadaptação (*underfitting*) e superadaptação (*overfitting*), entre outros. Todos constituem por si só um assunto de pesquisa em classificação, por essa razão cada tópico será brevemente explicado.

Dentre os problemas citados, os cinco primeiros estão diretamente relacionados ao conjunto de dados. Na seleção de atributos, de modo geral, um conjunto de treinamento pode conter vários atributos, de dezenas a centenas, de modo que a utilização de todos pode ser inviável e ineficiente, uma vez que alguns atributos podem dizer mais que outros sobre a classe dos dados. Dessa forma, antes de treinar o classificador, uma prática

bastante utilizada é selecionar alguns dos atributos que melhor distinguem as classes, alguns dos principais métodos para seleção de atributos são revisados nos trabalhos de (Guyon e Elisseeff, 2003) e (Liu e Yu, 2005). Tópicos relacionados para escolha de atributos utilizam transformação de atributos, tais como *Principal Component Analysis* (PCA) e *Linear Discriminant Analysis* (LDA) (Martínes e Kak, 2001) e construção de atributos (Markovitch e Rosenstein, 2002). A falta de valores de atributos ocorre quanto uma instância, por alguma razão, não possui algum de seus atributos. Neste caso, o classificador precisa lidar com duas situações, no treino e na classificação. No treino, se o conjunto de dados for grande e as instâncias com falta de atributo forem poucas, estas podem ser retiradas, caso contrário alguma medida deve ser tomada, no geral ou o classificador dispõe de mecanismos para lidar com esse tipo de problema, por exemplo o C45, ou um tipo de imputação (Hruschka *et al.*, 2009) deve ser realizado.

Ainda considerando o dado em questão, um problema inerente a todas as bases de dados reais é o ruído. De forma bastante abrangente, toda propriedade de uma instância que não está relacionada à distribuição original dos dados, mas sim a uma distribuição aleatória, é definido como ruído. Ruídos são causados por diversos fatores, como erro na atribuição da classe a uma instância, erro na leitura de atributos por parte de sensores ou falha humana, etc. Já com relação à *outliers*, Liu *et al.* (2002) os definem como padrões estranhos, que não se assemelham ao restante dos dados em algum sentido. Em geral, assim como no caso de um ruído, um *outlier* pode ser ocasionado por alguma falha. No entanto, nem todo *outlier* é resultado de falha, de fato, alguns deles podem indicar alguma propriedade significante da aplicação em questão. Basicamente existem duas abordagens no tratamento de *outliers*. Em uma delas, o foco é procurar por essas anomalias nos dados, ou identificar os *outliers* (Angiulli e Pizzuti, 2005), por exemplo, identificar intrusão em redes de computadores. A outra abordagem consiste na eliminação de *outliers* nos dados usados na classificação de dados (treino e teste), onde é necessário que o algoritmo tenha mecanismos para desconsiderar *outliers* (Brodley e Friedl, 1996). Portanto, no contexto de classificação dados, desenvolver algoritmos que sejam robustos à ruídos e *outliers* é importante pois a eliminação destes pode diminuir o erro de generalização do classificador construído.

Apesar de ser atribuído, a um algoritmo de aprendizado, um conjunto de treinamento finito, espera-se que o classificador, obtido do treinamento, tenha capacidade de generalizar os conceitos aprendidos. A generalização da performance de um classificador está relacionado a sua capacidade de predição em dados não vistos, i.e. que não estão presentes no conjunto de treinamento. Determinar o erro de generalização é de extrema importância na prática, uma vez que este erro está relacionado ao desempenho esperado do classificador. O erro de generalização é utilizado também na escolha entre diferentes algoritmos e na seleção de modelos, para uma determinada aplicação. A escolha de um algoritmo de aprendizado entre vários outros pode depender de diversos fatores que não somente o erro de generalização, tais como tempo de resposta, uso da memória, natureza

do problema, administração de riscos ou custos, tipo de dado, quantidade de dados, entre outras (Duda *et al.*, 2001). Estes fatores estão relacionados à complexidade de um algoritmo de aprendizado, a complexidade determina como o algoritmo se comporta, em relação ao tempo, com o aumento de instâncias de treinamento. No contexto de classificação, pode-se dividir a complexidade de um algoritmo em dois, a complexidade de treino e a de teste/aplicação. No geral, a complexidade de treinamento é menos importante que a complexidade na fase de aplicação, uma vez que nesta fase o classificador encontra-se em uso. Uma vez que está relacionada com a velocidade em que o classificador processará novos dados de alguma aplicação real. Uma vez definidas as possibilidades de algoritmos de aprendizado, com relação aos fatores descritos, é necessário ajustar os parâmetros para as opções paramétricas de algoritmos. Cada possível variação de parâmetros para um determinado algoritmo de treinamento constitui uma diferente instância do classificador que é chamado *modelo*. Determinar qual o melhor modelo para um problema de classificação é um problema de seleção de modelos e será abordado adiante.

A preocupação com a generalização é justificada, também, para evitar o problema de superadaptação ou subadaptação do classificador (Sarle, 1995). De forma simplista, no primeiro caso, o classificador é demasiado complexo para o problema em questão, de modo que este apresenta ótima performance no treino mas tem sua capacidade de generalização comprometida. Já no segundo caso ocorre o inverso, o classificador é demasiado simples para o problema em questão. O ajuste de um método de classificação ao conjunto de treinamento tem relação com o *dilema viés-variância* no qual o viés corresponde à diferença entre a verdadeira distribuição de probabilidade condicional das classes reais e a probabilidade estimada pelo classificador. Já a variância corresponde à variação das estimativas feitas pelo classificador. Métodos de classificação lineares apresentam baixa variância, pois considerando muitos conjuntos de dados as opções de classificação são sempre limitadas a um hiperplano, e alto viés, principalmente para dados não lineares. Por outro lado, métodos de classificação não lineares, apresentam alta variância, devido à grande capacidade de adaptação ao dado em questão, e baixo viés, por conseguir representar mais detalhadamente os dados de treinamento. O caso ideal é minimizar tanto o viés quanto a variância que, no geral, são termos conflitantes.

2.3.1 Classificação multiclasse e comitês

Esta seção revisa o uso de comitês na classificação em geral, bem como em domínios multiclasse e domínios não estacionários. Um comitê consiste em um conjunto de classificadores cujas predições individuais são combinadas para obter uma predição final. Métodos baseados em comitê, em geral, apresentam bom desempenho e têm maior poder de escalabilidade e paralelização do que classificadores simples. As principais preocupações no que diz respeito ao desenvolvimento de um comitê são, a escolha dos classificadores bases e o método de combinação das saídas dos classificadores. Um esquema geral para um comitê é mostrado na Figura 2.4.

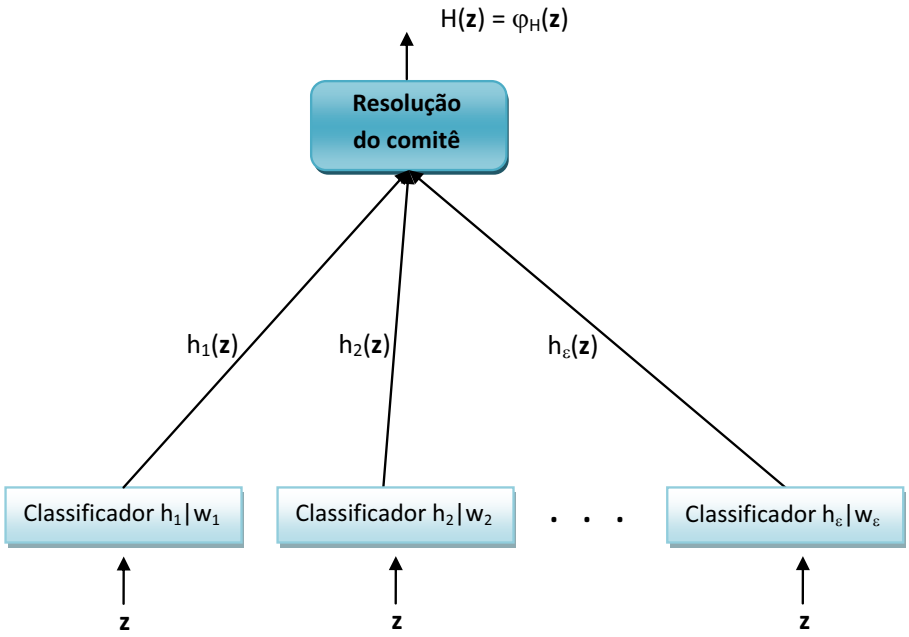

Figura 2.4: Esquema geral de classificação usando um comitê de classificadores.

Considere a classificação do exemplo de entrada $\mathbf{z}$, por meio do comitê, o exemplo é apresentado a todos os classificadores do comitê, que o processam e retornam uma saída $h_i(\mathbf{z})$. Esta saída depende de como o comitê é resolvido, por exemplo, nos casos onde o esquema de resolução é a votação, a saída dos classificadores pode ser a classe. Note que cada classificador possui um peso associado, pois em alguns casos a resolução do comitê envolve ponderação entre classificadores, de modo que, a saída pode representar alguma combinação entre resultado de classificação e peso. O peso, geralmente, é atualizado de acordo com o desempenho do comitê. A saída do comitê, $H(\mathbf{z})$, corresponde à classe mais provável atríbuida ao exemplo de entrada $\mathbf{z}$ segundo alguma regra de resolução.

Em domínios estacionários dois métodos bastante conhecidos para a construção de comitês são *bagging* e *boosting* (Opitz e Maclin, 2000). O método de *bagging* (Breiman, 1996) consiste em gerar ε conjuntos *boostrap* (Seção 2.3.2) e treinar cada classificador base em um conjunto *bootstrap* diferente. Breiman (1996) mostrou que o método de *bagging* é eficiente quando o algoritmo de aprendizado para a construção dos classificadores bases são instáveis, onde pequenas alterações no conjunto de treino resulta em grandes alterações na performance. O método de *boosting* (Schapire, 1990), consiste em uma família de métodos, mas de maneira geral, um novo classificador base é criado tendo como referência o desempenho dos últimos classificadores adicionados. De forma que, os exemplos classificados incorretamente têm maior possibilidade de estarem no conjunto de treinamento para a construção do novo classificador.

Já em domínios de dados com distribuição não estacionária, outros tipos de métodos para a criação de comitês têm sido propostos. Além de algoritmos específicos para a construção de comitês na resolução de problemas não estacionários, como alguns dos algoritmos apresentados na Seção 2.4.2; uma revisão dos principais métodos pode ser encontrada no trabalho de Fern e Givan (2003).

Algumas abordagens para solucionar problemas multiclasse podem ser vistas como aprendizado de um comitê de classificadores. Frequentemente problemas de classificação possuem mais que duas classes, e apesar de muitos algoritmos abordarem o problema naturalmente, como o KNN ou o Naive Bayes, outros necessitam de algumas alterações como no caso das redes neurais, cuja saída precisa ser associada a uma classe. Há também os classificadores que são estritamente binários, como no caso do SVM, que apesar de possuir outros tipos de extenções para o caso multiclasse, apresenta melhor resultado quando utilizado em sua forma binária combinadas em comitê (Hsu e Lin, 2002). Uma revisão de alguns dos principais métodos existentes para a combinação de classificadores pode ser encontrada no trabalho de Allwein *et al.* (2000).

Dentre estes métodos pode-se destacar dois, o método de um-contra-todos e o método um-contra-um. O primeiro caso, consiste em reduzir o problema de classificação entre as μ classes em μ subproblemas binários, onde cada subproblema discrimina uma dentre as μ classes, portanto cada classificador binário representa uma classe. Um comitê desse tipo pode ser formado construindo-se μ classificadores em μ conjuntos de treinamento diferentes. Cada conjunto é formado atribuindo-se a uma das classes do conjunto original, o rótulo 1, e às instâncias das demais classes o rótulo -1; para outro conjunto escolhe-se outra classe para atribuir-se rótulo 1 - até que todas as classes tenham sido representadas. No geral esse tipo de comitê é resolvido pela abordagem do vencedor leva tudo (*winner takes all*), i.e. a classe atribuída à instância de teste $\mathbf{z}$ é a classe referente ao classificador que apresenta a maior saída para $\mathbf{z}$.

Na segunda abordagem, um-contra-um, cada classe é comparada com toda outra classe. Um classificador binário é construído para discriminar entre todo par de classes, enquanto descarta o restante das classes. Esta abordagem requer construir $\mu(\mu - 1)/2$ classificadores. No teste de um novo exemplo, um esquema de votação é realizado e a classe com o maior número de votos é atribuída ao exemplo de dado a ser classificado. Resultados empíricos (Allwein *et al.*, 2000) favorecem esta abordagem em relação à anterior.

2.3.2 Seleção de modelos

A construção de um classificador envolve várias escolhas e variáveis, dentre as quais se enquadram as já citadas, pré-processamento, parâmetros do classificador, etc. Se considerada as várias combinações entre essas variáveis, nota-se que cada combinação resulta em uma instância de um classificador, ou um modelo. Indiscutivelmente, cada um desses modelos têm características próprias e até mesmo diferentes concepções sobre o problema em questão. Por essa razão, cada classificador terá um erro de generalização diferente, o que, inevitavelmente, leva a um problema de seleção de modelo.

Como mencionado o erro de generalização é de importância fundamental no processo de escolha de um classificador. Entretanto, obter uma medida confiável do erro de generalização não é tarefa trivial, especialmente quando não há dados suficientes. Em problemas

onde a quantidade de dados rotulados é grande (da ordem de milhares ou mais), é possível selecionar uma parte do conjunto suficientemente grande para treinar o classificador e utilizar o restante dos dados para testar e validar o classificador. Ainda assim, é necessário testar os modelos em diversos conjuntos de treino com diferentes tamanhos para definir um classificador adequado. No entanto, muitos dos problemas de classificação não dispõem de grande quantidade de dados rotulados, e utilizar o conjunto de treino para estimar o erro de generalização é uma estimativa otimista que não reflete o erro real. Neste tipo de problema, onde os dados rotulados são escassos (da ordem de centenas ou menos), é necessário algum método para estimar o erro real de generalização (Hastie *et al.*, 2009).

No que segue serão descritos dois dos métodos mais usados para estimar o erro de generalização, *Bootstrap* e validação cruzada. É importante discernir dois objetivos na estimativa do erro de generalização, seleção de modelos e avaliação do modelo. No primeiro o objetivo é estimar o desempenho de vários classificadores para escolher um deles. Uma vez escolhido o modelo, o objetivo no segundo caso, é determinar seu desempenho esperado. Os métodos discutidos nesta seção são baseados em amostragens. Métodos analíticos tais como AIC (Blatt *et al.*, 1973), MDL (Rissanen, 1983) e seleção de modelos Bayeisiana (Madigan e Raftery, 1994) e BIC (Schwarz, 1978) para obter o erro de generalização estão fora do escopo deste trabalho.

2.3.2.1 Bootstrap

Bootstrap é um método estatístico de reamostragem que pode ser usado para determinar o erro de generalização de um classificador. Um conjunto *bootstrap* é formado a partir do conjunto original e possui o mesmo número de elementos. No entanto, para a formação de cada conjunto *bootstrap* os exemplos de dados são selecionados aleatoriamente do conjunto original, podendo ocorrer repetições. Uma das maneiras para estimar o erro de generalização utilizando o método *Bootstrap*, consiste em treinar o classificador em B conjuntos *bootstrap* diferentes e estimar o erro utilizando o conjunto original como conjunto de teste. A estimativa do erro de generalização é dada segundo a média do desempenho dos B classificadores, como mostrado na Equação (2.12).

$$Erro_{bootstrap} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N}\xi(y_i, \varphi(\mathbf{x}_i)) \tag{2.12}$$

Na qual, $Erro_{bootstrap}$ corresponde ao erro de generalização estimado, N é o número de instâncias no conjunto de treinamento, ξ é uma função de erro e y_i e $\varphi(\mathbf{x}_i)$, correspondem à classe real e à classe estimada para a instância $\mathbf{x}_i$, respectivamente. Uma alternativa no conjunto usado para teste é considerar outro conjunto *bootstrap*, diferente do usado para o treinamento do modelo. No entanto, nos dois casos, é fácil verificar que o erro $Erro_{bootstrap}$ não é uma boa estimativa para o erro real, devido à existência de instâncias de dados comuns aos conjuntos de treino B e o conjunto original (ou outro conjunto *bootstrap*) que age como conjuntos de teste. Essa sobreposição de dados induz a uma

estimativa otimista do erro, o que torna a estimativa não confiável.

Outra maneira para estimar o erro de generalização utilizando o método *Bootstrap* considera a probabilidade de reamostragem como peso na estimativa do erro. Uma vez que cada amostragem de um conjunto *bootstrap* de tamanho N tem em média $0,632N$ instâncias sem repetição, Efron (1983) propôs obter o peso ponderando entre o erro no conjunto usado no treino e no conjunto original, usado no teste, como mostra a Equação (2.13).

$$Erro_{632} = 0,368E_{Tr} + 0,632E_{Te} \tag{2.13}$$

Na equação, $Erro_{632}$ denota a estimativa ponderada para o erro de generalização, E_{Tr} denota o erro no conjunto de treino e E_{Te} corresponde ao erro de teste, podendo este ser simplesmente a média do erro no conjunto original, em outros conjuntos *bootstrap* ou ainda a média do erro considerando outras variações, como por exemplo deixar de fora uma instância de treinamento a cada vez (*leave-one-out bootstrap*), como na Equação (2.14).

$$E_{Te} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|C^{-i}|}\sum_{b\in C^{-i}}\xi(y_i, \varphi(x_i)) \tag{2.14}$$

Na equação C^{-i} é o conjunto de índices dos conjuntos *bootstrap* b que não possuem a instância $\mathbf{x}_i$ e, novamente ξ é uma função de erro, y_i a classe da instância $\mathbf{x}_i$ e $\varphi(\mathbf{x}_i)$ a classe estimada pelo classificador. No entanto, apesar de resolver o problema de estimativa otimista do método *Boostrap* convencional, o *Boostrap 632* apresenta falha na estimativa de erro quando ocorre superadaptação (Efron e Tibshirani, 1997). O problema fica claro no exemplo citado por Breiman *et al.* (1984), no qual considera-se um problema de classificação com duas classes de tamanhos iguais e que o classificador 1NN seja usado, dessa forma, o erro no treinamento será $E_{Tr} = 0$, ao passo que o erro no conjunto de testes será $E_{Te} = 0,5$. Consequentemente o erro de generalização dado pelo método *Bootstrap 632*, $Erro_{632} = 0,368 \times 0 + 0,632 \times 0,5 = 0,316$, o que é uma estimativa ruim dado que o erro real é $0,5$. Em vista deste problema, Efron e Tibshirani (1997) propuseram o método *Bootstrap 632+*, que consiste em uma extensão do *Bootstrap 632* que considera a "quantidade" de superadaptação para estimar o erro. Esta correção corresponde à taxa de superadaptação relativa dada pela Equação (2.15).

$$\hat{R} = (E_{Te} - E_{Tr})/(\hat{\gamma} - E_{Tr}) \tag{2.15}$$

Na equação, $\hat{R}$ representa a taxa de superadaptação relativa que varia no intervalo $[0, 1]$ indicando o nível de superadaptação do classificador. Os erros E_{Te} e E_{Tr} são os erros obtidos nos conjuntos de teste e de treino, respectivamente. Dessa forma, $\hat{R} = 0$ implica que não existe superadaptação e $\hat{R} = 1$ significa que o classificador superadaptou-se completamente ao problema, o que equivale ao valor do erro de desinformação $(\hat{\gamma} - E_{Tr})$. Sendo que $\hat{\gamma}$ corresponde à taxa de erro sob nenhuma informação sobre o problema.

Esta taxa corresponde à predição de um determinado classificador se os atributos e as classes forem independentes, podendo ser estimada, para o caso multiclasse, com mostra a Equação (2.16).

$$\hat{\gamma} = \sum_{\omega_l} \hat{p}_l(1 - \hat{q}_l) \tag{2.16}$$

Na qual, $\hat{p}_l$ é a proporção observada de classes ω_l, e $\hat{q}_l$ é a proporção observada de classificações que resultaram na classe ω_l. Dessa forma, incorporando informações sobre o nível de superadaptação do classificador na Equação (2.13), é obtida a extensão, mais robusta à superadaptação, nomeada *Bootstrap 632+* e mostrada na Equação (2.17).

$$Erro_{632+} = (1 - \hat{w})E_{Tr} + \hat{w}E_{Te} \tag{2.17}$$

Na qual $\hat{w} = 0,632/(1-0,368R)$, e $\hat{w}$ varia entre $0,632$ a 1, fazendo com que $Erro_{632+}$ varie entre $Erro_{632}$ e E_{Te}. Em conjuntos pequenos onde classificadores pouco adaptativos foram usados, o método de estimativa *bootstrap 632+* é considerado a melhor escolha (Kim, 2009).

2.3.2.2 Validação cruzada

Um dos métodos mais usados para estimar o erro de generalização é a validação cruzada. Na validação cruzada o conjunto de treinamento é dividido aleatoriamente em dois, onde uma parte será usada para treinar o classificador e o restante para estimar o erro de generalização. No entanto, esta abordagem requer isolar um conjunto de validação, o que pressupõe disponibilidade de dados. Na maioria das vezes, entretanto, os dados rotulados são escassos para utilizar todo o conjunto de dados no processo de derivação do erro. Uma alternativa é usar a validação cruzada de k-conjuntos. Neste caso o conjunto de treinamento é dividido em k subconjuntos com tamanhos iguais (ou aproximados) N/k, no qual N é o número de elementos no conjunto de treinamento. Cada modelo é treinado k vezes, cada vez com um conjunto diferente, formado por $(k-1)$ conjuntos, e deixando um conjunto diferente a cada vez, para validação. Considere a divisão em conjuntos, mostrado na Figura 2.5, cada conjunto é dividido em k conjuntos, sendo que um é deixado para teste, mostrado em destaque (vermelho) na figura.

O erro para cada modelo é calculado como a média do erro nos k conjuntos de validação. No geral, k pode ser escolhido no intervalo $1 \leq k \leq N$, sendo que quando $k = N$, o método é chamado de *leave-one-out*, valores comumente usados e que resultam em bons resultados são $k = 5$ e $k = 10$ (Hastie *et al.*, 2009). A estimativa do erro de classificação é dada pela Equação (2.18).

$$Erro_{CV} = \frac{1}{N} \sum_{i=1}^{N} \xi(y_i, \varphi(\mathbf{x}_i)) \tag{2.18}$$

Na equação, $Erro_{CV}$ corresponde ao erro de generalização estimado, ξ a uma determi-

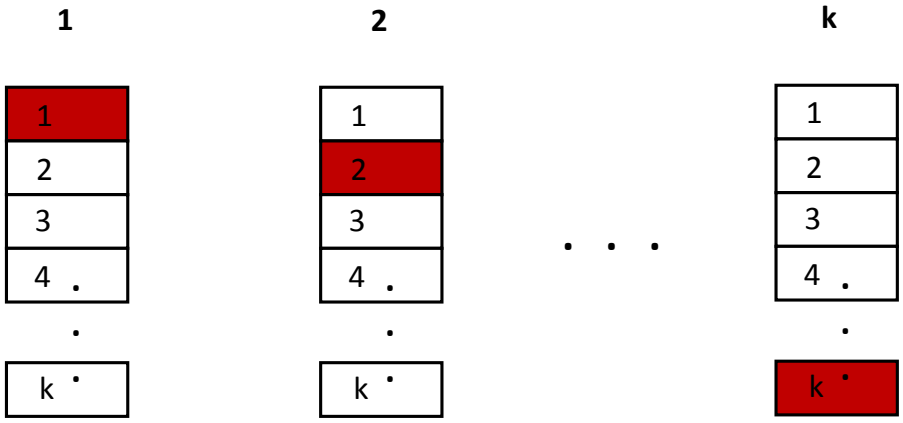

Figura 2.5: Exemplo de divisão de conjuntos para validação cruzada de k-conjuntos.

nada função de erro e, y_i e $\varphi(\mathbf{x}_i)$ referem-se à classe real e à classe inferida pelo classificador para a instância $\mathbf{x}_i$, respectivamente. Com o conjunto de treinamento fixo, a validação cruzada de k-conjuntos apresenta uma variação nos resultados devido à aleatoriedade das partições nos k conjuntos. Na literatura, esta variação é chamada de variância interna (ver (Efron e Tibshirani, 1997), (Braga-Neto e Dougherty, 2004). Manter as porcentagens das classes nos k conjuntos similar à apresentada no conjunto original, i.e. estratificar, diminui a variância interna. Outra forma, que pode ser aliada à estratificação, é repetir todo o processo de validação cruzada algumas vezes e considerar a média como estimativa final. Outras variações, como validação cruzada dupla com repetição, na qual são considerados dois níveis de repetição, uma validação dentro de outra, também são estudados na literatura para seleção de modelos (Filzmoser *et al.*, 2009). Kim (2009) verificou que, exceto para conjuntos de dados pequenos aplicados a classificadores pouco adaptativos, onde o *Bootstrap 632+* é a melhor escolha, a melhor estimativa para o erro de generalização é obtida pela validação cruzada com repetição. Portanto, é o método de estimativa mais recomendado para uso geral - o que motivou o uso neste trabalho.

2.3.3 Algoritmos de aprendizado estáticos

Esta seção apresenta alguns algoritmos de aprendizado para domínios de dados estacionários, notados como algoritmos estáticos, e aplicáveis a domínios cuja distribuição dos dados de teste não se diferem da distribuição dos dados de treino. Considere a seguinte notação, seja o conjunto de treinamento $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, com N instâncias, onde a i-ésima instância é representada por $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip}, y_i)$, sendo p o número de atributos e um rótulo de classe associado, y_i, pertencentes a uma das classes do conjunto $\Omega = \{\omega_1, \ldots, \omega_\mu\}$. Uma instância não rotulada para ser classificada é representada por $\mathbf{z} = (z_1, \ldots, z_p)$; a classe atribuída por um classificador ao exemplo de dado não rotulado $\mathbf{z}$ é representado por $\varphi(\mathbf{z})$.

No que segue são apresentados alguns algoritmos de aprendizado bem conhecidos para a construção de um classificador, sendo que alguns deles serão utilizados como comparativos para o algoritmo KAOG, apresentado no Cápitulo 3.

2.3.3.1 Aprendizado Bayesiano

Métodos de aprendizado Bayesiano permitem uma abordagem probabilística para inferência. Baseiam-se na idéia de que o problema em questão é governado por uma distribuição de probabilidade e que decisões ótimas podem ser tomadas considerando essas probabilidades junto ao dado observado. Esta abordagem baseia-se em quantificar os benefícios entre as várias decisões de classificação usando probabilidade e o custo envolvendo tal decisão.

De acordo com Mitchell (1997), o aprendizado bayesiano tem, ao menos, duas características interessantes. A primeira, algoritmos de aprendizado Bayesiano, além de serem práticos, permitem calcular explicitamente a probabilidade de uma dada hipótese. A segunda razão, é que, o aprendizado Bayesiano provê base para o entendimento de outros métodos de classificação que não lidam com probabilidades diretamente. Dentre as principais características do aprendizado bayesiano, pode-se destacar: 1) cada novo dado classificado pode incrementar ou decrementar a probabilidade estimada de uma hipótese estar correta; 2) o conhecimento armazenado no conjunto de treinamento pode ser combinado com novas instâncias para determinar a probabilidade final de uma hipótese; 3) métodos Bayesianos permitem uma predição probabilística das hipóteses; 4) novas hipóteses podem ser classificadas por meio da combinação de múltiplas hipóteses, ponderadas pelas respectivas probabilidades.

Para a apresentação do teorema de Bayes, considere $P(h)$ a probabilidade de que a hipótese h ocorra, também é chamado de probabilidade *a priori* de h e reflete algum conhecimento prévio sobre a chance de h ser a hipótese correta. Se não houver nenhum conhecimento prévio sobre as hipóteses, todas as hipóteses recebem a mesma probabilidade *a priori*. De modo similar $P(X)$ denota a probabilidade *a priori* de que o conjunto X seja observado. A probabilidade $P(X|h)$ denota a probabilidade de que X ocorra dado a hipótese h seja verificada. No caso do aprendizado de máquina o que interessa é $P(h|X)$, ou seja, a probabilidade de h ocorrer dado X, chamado de probabilidade *a posteriori* de h. O aprendizado Bayseano é baseado no teorema de Bayes, e fornece uma maneira de calcular a probabilidade *a posteriori* $P(h|X)$ dado as probabilidades $P(h)$, $P(X)$ e $P(X|h)$, a relação entre as probabilidades é mostrada na Equação (2.19).

$$P(h|X) = \frac{P(X|h)P(h)}{P(X)} \tag{2.19}$$

A equação de Bayes mostra que, observando o conjunto X, pode-se converter a probabilidade *a priori* $P(h)$, em uma probabilidade *a posteriori* $P(h|X)$, que é a probabilidade de ocorrer h dado o conjunto de treinamento X. A classificação aplica o teorema de Bayes a tarefas de aprendizado nas quais a função alvo pode assumir qualquer valor dentro de um conjunto finito Ω. Dado um conjunto de treinamento, a abordagem do algoritmo Naive Bayes para a classificação de um nova instância $\mathbf{z}$ é atribuir o valor mais provável $\varphi(\mathbf{z})$, dado os valores de atributos da nova instância $(z_1, z_2, \ldots, z_p)$ e usando o teorema

de Bayes, como mostra a Equação (2.20).

$$\varphi(\mathbf{z}) = arg\max_{\omega_j \in \Omega} \frac{P(\omega_j|z_1, z_2, \ldots, z_p)P(\omega_j)}{P(z_1, z_2, \ldots, z_p)} \tag{2.20}$$

Uma maneira de estimar ambos os termos da Equação (2.20) baseia-se no conjunto de treinamento. Por exemplo $P(\omega_j)$ pode ser estimado contando a frequência de exemplos de classe ω_j no conjunto de treinamento. Entretanto, estimar os diferentes termos $P(z_1, z_2, \ldots, z_p)$ dessa maneira, não é viável a menos que o conjunto de dados seja muito grande. O problema é que o número desses termos é igual ao número de instâncias possíveis vezes o número de classes.

A classificação baseia-se na simples suposição de que os valores dos atributos são condicionalmente independentes, dado o valor da classe. Em outras palavras, dado uma classe para a instância $\mathbf{z}$, a probabilidade de ocorrer a conjunção $z_1, z_2, \ldots, z_p$ é dada pelo produto das probabilidades individuais dos atributos $P(z_1, z_2, \ldots, z_p|\omega_j) = \prod_i P(z_i, \omega_j)$. Substituindo na Equação (2.20), pode-se estimar $\varphi(\mathbf{z})$ de acordo com a Equação (2.21).

$$\varphi(\mathbf{z}) = arg\max_{\omega_j \in \Omega} P(\omega_j) \prod_i P(z_i, \omega_j) \tag{2.21}$$

No qual $\varphi(\mathbf{z})$ denota a classe resultante da classificação. O método de aprendizado Bayesiano envolve estimar as varias probabilidades $P(X)$ e $P(h|X)$, baseadas na frequência no conjunto de treinamento. Uma diferença interessante entre o aprendizado Bayesiano comparado a outras técnicas, é que, neste não há busca explicita no espaço de hipóteses. Ao invés disso, as hipóteses são formadas simplesmente contanto a frequência de dados no conjunto de treinamento. A principal desvantagem do aprendizado Bayesiano é que assume-se que o problema seja definido em termos probabilísticos, e que todos os valores probabilísticos relevantes são conhecidos.

2.3.3.2 O método dos vizinhos mais próximos - KNN

O método dos K vizinhos mais próximos (K *Nearest Neighbors* (KNN)) é um método de aprendizado de máquina baseado em instância, uma vez que consiste simplesmente em armazenar o conjunto de treinamento em questão. Quando uma nova instância precisa ser classificada, um conjunto de instâncias similares é recuperado da memória e usado na classificação da nova instância. Esse tipo de classificador é baseado em memória, e não requer treinamento de um classificador. Dado uma entrada de teste $\mathbf{z}$, encontram-se os K vizinhos mais próximos, ou seja as K instâncias mais próximas, segundo alguma medida de distâncias, em relação à $\mathbf{z}$, e então, a classificação é feita usando o critério de maioria dos votos entre os K vizinhos. O conceito de proximidade implica o uso de uma medida de distância, o exemplo trivial é a distância Euclidiana, dentre outras possibilidades estão a distância *Manhattan* e a distância tangente (Duda *et al.*, 2001).

O algoritmo KNN calcula as distâncias entre o exemplo a ser classificado $\mathbf{z}$ e todas as instâncias, $\mathbf{x}_i$, pertencente ao conjunto rotulado X. Em seguida, as K instâncias

mais próximas à $\mathbf{z}$ são selecionadas, de modo que a classe atribuída a $\mathbf{z}$, será a classe mais comum entre as K instâncias selecionadas. Quando K é um valor par pode ocorrer empate na classificação da nova instância, nestes casos Hastie *et al.* (2009) sugerem resolver o impasses de maneira aleatória. Esta decisão, por mais curiosa que pareça, faz sentido pelo fato de que outro tipo de decisão pode favorecer outro valor de K. Por exemplo, se o critério de decisão em caso de empate fosse optar pela classe do vizinho mais próximo, toda vez que houvesse empate seria como usar $K = 1$. O processo de classificação feito pelo KNN pode ser escrito como na Equação (2.22).

$$\varphi(\mathbf{z}) = \{\omega_j | \max_{\omega_j \in \Omega} \sum_{\forall \mathbf{x}_i \in \Lambda_{\mathbf{z},K}} \delta_{\omega_j, y_i}\} \tag{2.22}$$

Na equação, $\Lambda_{\mathbf{z},K}$ corresponde à K-vizinhança do exemplo $\mathbf{z}$ definida pelos seus K vizinhos mais próximos, δ_{ij} é o delta de Kronecker, no qual $\delta_{ij} = 1$, se $i = j$, e $\delta_{ij} = 0$, caso contrário. Note que a classe do novo elemento, segundo a classificação pelo KNN, pode mudar várias vezes conforme o valor de K aumenta.

Apesar de sua simplicidade, o algoritmo KNN tem sido utilizado com sucesso em diversos problemas de classificação incluindo reconhecimento de escrita manual (Simard *et al.*, 1993), classificação de imagens de satélite (Michie *et al.*, 1994), etc. O KNN geralmente é aplicado com sucesso onde cada classe do conjunto rotulado possui vários dos padrões possíveis, e a borda de decisão é muito irregular. Um dos principais problemas associado ao método KNN, é justamente a escolha do seu único parâmetro K. A melhor escolha para K depende do dado em questão. Geralmente, valores pequenos para K produzem bordas de decisão irregulares, como mostra a Figura 2.6(a), tendendo a distinguir todo dado individualmente em regiões de ruído, o que compromete a generalização. Já valores altos para K reduzem o efeito de ruídos na classificação, mas tornam a borda de decisão entre as classes, menos distintas e mais suaves, como mostrado na Figura 2.6(b). A escolha de K, na maioria das vezes é feita através de tentativa e erro ou através de alguma heurística como validação cruzada.

À primeira vista o desempenho assintótico do método dos vizinhos mais próximos parece ser muito bom, considerando a simplicidade da regra. Uma explicação heurística para isto deve-se ao fato de que a variável associada à classe da instância a ser classificada, $\mathbf{z}$, é uma variável randômica. A probabilidade de que $\varphi(\mathbf{z}) = \omega_i$ é simplesmente a probabilidade *a-posteriori* $P(\omega_i|\mathbf{z})$. E se $\mathbf{z}$ for muito próximo de $\mathbf{x}_i$, pode-se assumir que $P(\omega_i|\mathbf{x}_i) \approx P(\omega_i|\mathbf{z})$. Assumindo que existe um número ilimitado de dados de treinamento, $\mathbf{x}_i$ será muito próximo do exemplo $\mathbf{z}$. Se a atribuição das classes nos dados de treinamento foi feita de maneira correta então o valor de $\mathbf{z}$ é o valor que maximiza a probabilidade *a-posteriori* $P(\omega_i|\mathbf{z})$.

KNN com peso Uma das variações naturais do KNN, o algoritmo KNN com peso, consiste em priorizar a ordem de vizinhança entre os K vizinhos mais próximos do exemplo de dado a ser classificado, i.e. quanto menor a distância do novo dado para um vizinho,

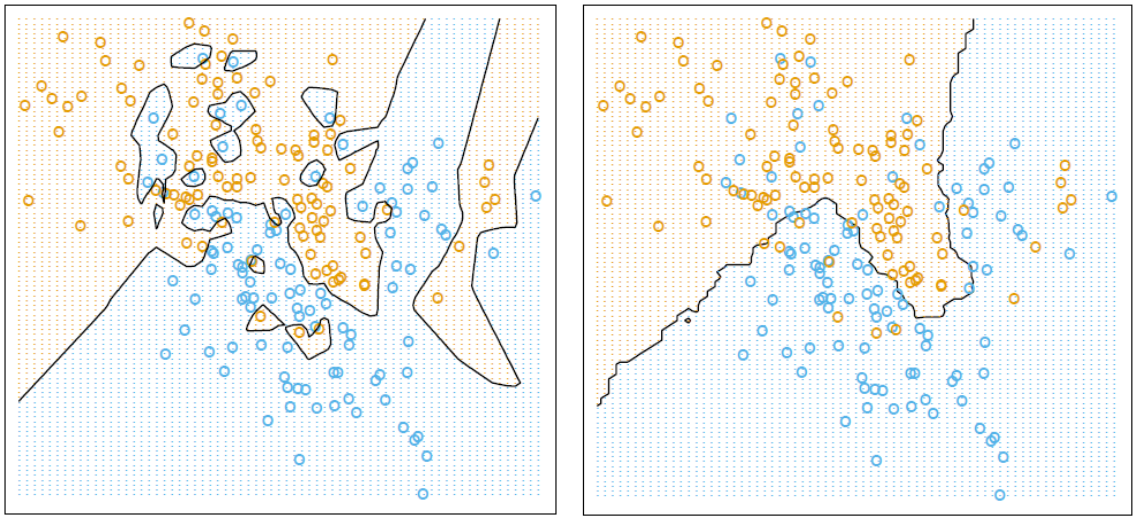

Figura 2.6: Superfícies de decisão para o método KNN, para os valores de K, (a) $K = 1$ e (b) $K = 15$ (Hastie *et al.*, 2009).

maior será a influência deste na classificação. Dessa forma a regra de classificação depende do inverso da distância, como mostra a Equação (2.23).

$$\varphi(\mathbf{z}) = \{\omega_j | \max_{\omega_j \in \Omega} \sum_{\forall \mathbf{x}_i \in \Lambda_{\mathbf{z},K}} \delta_{\omega_j, y_i} \frac{1}{d(\mathbf{x}_i, \mathbf{z})}\} \tag{2.23}$$

Outras formas de peso podem ser usadas, como $1/d(\mathbf{x}_i, \mathbf{z})^2$, que intenssifica o peso da distância. Note que ao inserir o peso baseado na distância, o aumento significativo no número de vizinhos tem pouca influência na classificação, pois instâncias rotuladas longe do exemplo a ser classificado contribui muito pouco para a decisão final.

*K*NN *baseado em protótipo* O algoritmo KNN pode ser adaptado para atuar como um método baseado em protótipo. Métodos baseados em protótipo representam o conjunto de treinamento por meio de um conjunto de pontos no espaço de atributos. Em geral, esses pontos, chamados protótipos, não são pontos do conjunto de treinamento - com exceção do caso do 1NN, onde cada instância do conjunto de treinamento pode ser visto como um protótipo. A grande vantagem desse tipo de método é a redução no espaço de busca, no entanto, para isso é necessário um método para definir os protótipos, algumas opções incluem métodos de agrupamentos como *k-means* (MacQueen, 1967), (Lloyd, 1982), e de quantização (Kohonen, 1989).

Uma vez definidos, cada protótipo tem uma classe associada e, em geral, cada classe têm um mesmo número de protótipos. Dado um novo exemplo a ser classificado, verifica-se a distância deste aos protótipos e a classificação é feita de acordo com a distância do protótipo mais próximo. Apesar de certas vantagens, o KNN baseado em protótipos ainda depende do parâmetro que controla o número de protótipos que deverão ser encontrados para cada classe. Há também a possibilidade de utilizar outros valores de K que não um, no entanto, esta possibilidade leva em conta dois parâmetros similares que só aumentará

o número de modelos a ser testado na fase de seleção de modelos, sem grandes vantagens no desempenho do algoritmo.

2.3.3.3 Árvores de decisão

As árvores de decisão enquadram-se na categoria dos métodos não métricos. Isto implica que este tipo de abordagem lida naturalmente com atributos nominais (Quinlan, 1986), todavia permitem variações que possibilitam lidar com atributos numéricos. Alguns dos principais métodos de indução de árvores de decisão são, o C4.5 (Quinlan, 1993), CART (Breiman *et al.*, 1984), BOAT (Gehrke *et al.*, 1999) e RainForest (Gehrke *et al.*, 2000).

A indução de uma árvore de decisão a partir de um conjunto de treinamento utiliza a abordagem dividir para conquistar. Por se tratar de uma estrutura em árvore, o processo de criação é naturalmente definido de maneira recursiva. Portanto, dado um conjunto de instâncias rotuladas, cria-se um nó e verifica-se se este nó será folha ou nó intermediário. No primeiro caso interrompe-se o crescimento da subárvore e o nó folha representa a classe de maior ocorrência no nó. No segundo caso, a subárvore continua sendo expandida, o que implica em selecionar um atributo e um limiar (*threshold*), deste atributo, que dividirá novamente o conjunto usado na criação do nó corrente para continuar a crescer a árvore.

Um ponto importante nesse tipo de método é saber quando parar de crescer a árvore ou parar de dividir os nós. Esta decisão implica no critério usado pelo algoritmo para decidir se, a partir do nó corrente, continua dividindo ou se torna o nó folha. Intuitivamente, um nó é definido como folha se toda instância do subconjunto possui uma mesma classe, que será a classe que o nó folha representará. Este caso, no entanto, seria o caso ideal onde o subconjunto em questão é puro. No caso geral, entretanto, para se gerar subconjuntos puros pode ser necessário muitas iterações e ocasionar o crescimento demasiado da árvore. O problema em gerar árvores grandes, como adverte o princípio de Occam (Mitchell, 1997), é que estas árvores geralmente apresentam características indesejadas, tais como, aumento no tempo de classificação, superadaptação e, por consequência, perda de generalização bem como dificuldade de entendimento e visualização humana.

Com o fim de evitar esses problemas, é necessário aceitar alguma mistura entre classes nos subconjuntos dos nós folha. Uma maneira usada pelo C4.5 (Quinlan, 1996) é somar os erros apresentados pelos nós filhos e verificar se esta soma é maior que o erro em classificar todas as instâncias do nó em questão como a classe mais frequente. No caso positivo o nó em questão é tornado folha e as suas subárvores são removidas (o que é chamada de pré-poda). Outro artifício para evitar o crescimento desnecessário da árvore é o uso do parâmetro m, que indica o número mínimo de instâncias de treinamento em um subconjunto necessário para que o respectivo nó ainda possa ser dividido.

Quando decide-se pela divisão de um determinado nó, a seleção de um atributo para a divisão em um determinado nó é feita por meio de alguma medida para quantificar a pureza desse atributo, a medida mais utilizada é a entropia (ou impureza de informação),

definida na Equação (2.24).

$$Ent(X_T) = -\sum_{\omega_j} P(\omega_j) log_2 P(\omega_j) \tag{2.24}$$

Na equação, $Ent(X_T)$ é a entropia do conjunto X_T, sendo este o conjunto de dados do nó T. O cálculo da entropia considera as probabilidades das classes, $P(\omega_j)$, que podem ser definidas como a porcentagem de instâncias do conjunto X pertencentes às classes $\omega_j \in \{\omega_1, \ldots, \omega_\mu\}$. A entropia mede a impureza do conjunto, variando no intervalo $[0, 1]$, onde 0 indica conjunto com apenas uma classe e 1 com igual número para todas as classe. A partir da entropia, pode-se definir o ganho de informação, como na Equação (2.25).

$$ganho = Ent(X) - \sum_{k=1}^{B} P_k Ent(X_k) \tag{2.25}$$

Na equação, $P_k = |X_k|/|X|$ é a fração de instâncias de X direcionadas ao filho k, na equação B refere-se ao grau da árvore, $B = 2$ no caso de árvore binária. O maior ganho de infromação representa a melhor divisão do conjunto em questão, pois como o primeiro termo é constante para cada nó, a divisão que resulta no menor valor para a somatória é a divisão que resulta em conjuntos mais puros.

Ocasionalmente, parar de crescer a árvore pode resultar no chamado efeito horizonte, que informalmente significa deixar de explorar todo o espaço de dados. Ao parar de particionar algum nó, as subárvores que resultariam desta divisão, nunca serão geradas, e o espaço de dados que estas subárvore explorariam não será visitado. Este efeito pode ocorrer quando a condição de parada de particionamento está mal ajustada ou a condição de parada é atingida mais cedo por conta do desempenho global da árvore. A saída para esse tipo de problema é a poda (ou pós-poda), onde se cresce a árvore ao máximo, i.e. até todas as folhas representarem conjuntos de uma só classe, então aplica-se a poda. A poda é regulada pelo parâmetro fator de confiança cf, para o qual, maior valor implica em poda mais agressiva. Apesar de existir duas possibilidades, pré-poda e pós-poda, a seleção de modelos em árvore de decisão deve considerar ambas, inclusive usando as duas juntas através da variação dos parâmetros m e cf.

A classificação de uma nova instância começa no nó raiz, onde se verifica um determinado valor de atributo da nova instância. Cada nó filho corresponde a um possível (e mutuamente exclusivas) caminho à verificação do atributo abordado no nó pai. O processo de classificação é feito percorrendo-se a árvore, de forma descendente, de acordo com as verificações corretas até atingir um nó folha que, por definição, determina uma classe sem exigir teste.

2.3.3.4 Redes neurais

As redes neurais artificiais podem ser vistas como um sistema de processamento paralelo e distribuído que consiste de um grande número de processadores (neurônios) simples

e massivamente conectados. As redes neurais proporcionam uma abordagem robusta na resolução de vários tipos de problemas, incluindo aproximação de funções, tomada de decisão, classificação, controle, etc. (Haykin, 2008). Definida de maneira simplista, uma rede neural é uma rede com processadores simples (cada um deles tendo, possivelmente, uma pequena quantidade de memória local) conectados por meio de canais de comunicação (conexões) aos quais usualmente estão associados valores (pesos) numéricos. De acordo com Palma Neto e Nicoletti (2005), uma rede neural pode então ser caracterizada,

- por seus processadores, chamados neurônios;
- pela função de ativação que representa o estado do neurônio;
- pelo padrão de conexão existente entre os neurônios, arquitetura da rede;
- por seu algoritmo de treinamento (também chamado de algoritmo de aprendizado).

O algoritmo de treinamento de uma rede é um conjunto de regras por meio das quais os pesos das conexões são ajustados usando um conjunto de treinamento. A arquitetura de uma rede neural pode ser caracterizada pelos neurônios que a compõe, pelo padrão de conexão entre eles e pela organização desses neurônios em camadas (quando for o caso). A arquitetura da rede é uma das maiores responsáveis pelo seu desempenho e, por essa razão, seu principal parâmetro. A definição da arquitetura da rede não é tarefa trivial, dentre as maneiras de se obter a melhor arquitetura para uma determinada tarefa pode-se categorizar os métodos em manuais e automáticos. No primeiro grupo, encaixa-se talvez o mais usado e menos eficiente deles, o método de tentativa e erro além dos métodos de seleção de modelos citados na Seção 2.3.2. A definição automática da arquitetura por sua vez pode ser dividida em dois grupos, métodos evolucionários e não evolucionários. Nos métodos evolucionários a estrutura da rede é obtida por meio de metodologias evolutivas, como algoritmos genéticos e busca estocástica baseada em populações, etc. (Schaffer *et al.*, 1992), (Yao, 1999). Já nos métodos não evolucionários compreende todos os outros métodos que constroem a rede por meios que não são evolucionários. Dentre as abordagens de métodos não evolucionários estão os algoritmos neurais construtivos (Parekh *et al.*, 2000), (Nicoletti e Bertini Jr., 2007), (Franco *et al.*, 2009) que constroem a rede ao longo do treinamento. Este tipo rede tem sido usada com sucesso em aplicações de diagnósticos médicos (Figueira *et al.*, 2006) a filtro de sinais (Giordano *et al.*, 2008). Outra categoria de método não evolucionários são os métodos baseados em poda (Reed, 1993), (Lahnajärvi *et al.*, 2002), a qual pode ser vista como uma abordagem contrária ao métodos construtivos, uma vez que o aprendizado é iniciado com uma rede bastante grande e por meio da poda de neurônios ajusta-se a rede ao problema em questão.

O desempenho da rede é consequência direta da arquitetura da rede treinada por um algoritmo de aprendizado, dessa forma pode-se dizer que arquitetura e algoritmo de aprendizado tenha um papel fundamental no aprendizado da rede.

O Perceptron (Rosenblatt, 1958) é um algoritmo básico para o aprendizado de um vetor de pesos $\mathbf{w}$ (de um único neurônio), também usado na definição de uma função linear discriminante. O Perceptron pode ser visto como um separador linear que pode ser treinado para classificar conjuntos de vetores como pertencentes a uma, de duas regiões separadas por um hiperplano. A regra de atualização do vetor de pesos utiliza somente reforço negativo, sempre que uma instância $\mathbf{x}_i$ é classificado incorretamente, o vetor de pesos $\mathbf{w}$ é atualizado de acordo com a Equação (2.26).

$$\mathbf{w} = \mathbf{w} - \eta(d - o)\mathbf{x}_i \tag{2.26}$$

Na qual d é a saída desejada, ou seja a classe correta no contexto de classificação; o é a saída do neurônio em questão. No entanto um único neurônio treinado com o Perceptron somente é capaz de distinguir entre classes linearmente separaveis. Para representar funções não lineares é necessário uma rede com vários neurônios dispostos em camadas com a restrição de que as saídas dos neurônio sejam uma função não linear de suas entradas (Mitchell, 1997). Uma possível solução é utilizar a função sigmóide como função de ativação para o Perceptron, como mostrado na Equação (2.27).

$$F(net) = \frac{1}{1 + e^{-net}} \tag{2.27}$$

Na equação, net corresponde ao potencial de ativação do neurônio, ou seja $net = \mathbf{w}^\top \mathbf{x}$. Note que os valores de F, variam de 0 a 1, aumentando monotonicamente com relação à entrada. Uma propriedade interessante da função sigmóide é o fato de que sua derivada pode ser facilmente expressa em termos de sua saída, como mostra a Equação (2.28).

$$F'(net) = \frac{dF(net)}{dnet} = F(net)(1 - F(net)) \tag{2.28}$$

Outras funções facilmente deriváveis podem ser usadas ao invés da função sigmóide, por exemplo, a tangente hiperbólica. Dada uma rede multicamadas formada por um número fixo de neurônios com este tipo de função de ativação e interconexões, é possível utilizar o algoritmo *backpropagation* para o ajuste dos pesos. O algoritmo baseia-se no método do gradiente descendente para minimizar o erro quadrático da rede com relação às saídas desejadas. O algoritmo *backpropagation* é composto por duas fases, na primeira, fase de propagação (*Feedforward*), as entradas propagam-se pela rede da camada de entrada à camada de saída. Na segunda fase, de retropropagação (*Feedback*), os erros propagam-se na direção contrária ao fluxo de dados, ou seja da camada de saída à primeira camada escondida. A regra de atualização é a mesma para todos os pesos, como mostra a Equação (2.29).

$$\mathbf{w}_{ji} = \mathbf{w}_{ji} + \Delta\mathbf{w}_{ji}, \quad \text{onde} \quad \Delta\mathbf{w}_{ji} = \eta e_i o_j \tag{2.29}$$

Na equação, o_j é a saída do neurônio u_j e η é a taxa de aprendizado. O que muda de acordo com a posição do neurônio u_i na rede, é o cálculo do erro e_i. Se o neurônio u_i for neurônio de saída, o erro é calculado diretamente, como mostrado na Equação (2.30).

$$e_i = (d_i - o_i)F'(net_i) \tag{2.30}$$

Já para o caso de um neurônio u_j estar em uma das camadas escondidas, como os mesmos não têm acesso direto ao erro, estes baseiam-se nos erros da camada posterior. O erro do neurônio u_j pode ser definido como na Equação (2.31).

$$e_j = F'(net_j) \sum e_i \mathbf{w}_{ji} \tag{2.31}$$

A Figura 2.7 ilustra três passos do processo de correção de erros do algoritmo *backprogation*, na figura é ilustrada uma rede com apenas uma camada intermediária.

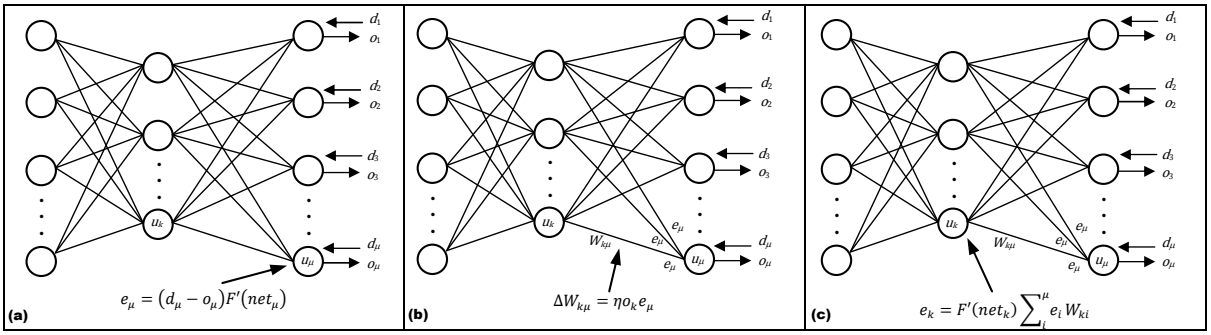

Figura 2.7: Exemplo de correção de erro realizado pelo *Backpropagation*.

Na Figura 2.7(a), os erros da camada de saída são calculados, em seguida ocorre a atualização dos pesos localizados entre a camada de saída e a camada escondida imediatamente anterior, como mostra a Figura 2.7(b). Na Figura 2.7(c) o cálculo do erro para cada neurônio baseia-se no erro retro-propagado da saída. O próximo passo seria atualizar os pesos entre a camada de entrada e a camada escondida, da mesma maneira como ilustrado na Figura 2.7(b). Apesar de tratar-se de uma ótima alternativa para classificação, as redes MLP sofrem da dificuldade em se obter um bom modelo, i.e. ajuste da arquitetura da rede e de outros parâmetros (Reed e Marks II, 1999).

2.3.3.5 Métodos baseados em kernel

Qualquer solução baseada em *kernel* pode ser dividida em duas partes, um módulo que realiza o mapeamento no espaço de atributos e um algoritmo de aprendizado desenvolvido para encontrar padrões neste espaço (Shawne-Taylor e Cristianini, 2004). Algoritmos no espaço de atributos utilizam a seguinte idéia:

$$\Phi : \mathbb{R}^p \to F, \qquad \mathbf{x}_i \to \Phi(\mathbf{x}_i) \tag{2.32}$$

Na equação, a instância $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip}, y_i)$ é mapeada para um espaço de atrib-

utos, F, potencialmente muito maior. Em um problema de aprendizado, considera-se o mesmo algoritmo em F ao invés de $\mathbb{R}^p$. As instâncias passam a ser escritas como na Equação (2.33).

$$(\Phi(\mathbf{x}_1), y_1), \ldots (\Phi(\mathbf{x}_n), y_n) \tag{2.33}$$

Determinado o mapeamento Φ, pode-se facilmente realizar classificação ou regressão em F. Por meio desta abordagem, é possível transformar um problema linearmente não separável em um problema linearmente separável em um espaço de dimensão maior (Müller *et al.*, 2001). A variabilidade necessária para encontrar uma boa representação dos dados é dada pelo mapeamento Φ. De maneira geral, o que importa não é a dimensão dos dados, mas sim sua complexidade. O que torna possível lidar com instâncias de alta dimensionalidade é o uso de um método altamente eficiente para calcular o produto escalar no espaço de atributos - o uso de funções *kernel*. Uma função *kernel* apresenta a seguinte propriedade.

$$k(\mathbf{x}_i, \mathbf{x}_j) = \Phi^\top(\mathbf{x}_i)\Phi(\mathbf{x}_j) \tag{2.34}$$

O uso desse tipo de função possibilita utilizar técnicas simples de classificação em altas dimensões sem pagar o preço computacional implícito no número de dimensões, uma vez que é possível calcular o produto escalar entre as imagens de dois exemplos em um espaço de atributos sem calcular explicitamente suas coordenadas. Um exemplo clássico de função *kernel* é o *kernel* Gaussiano $k(\mathbf{x}_i, \mathbf{x}_j) = exp(- \parallel \mathbf{x}_i - \mathbf{x}_j \parallel^2)/2\sigma^2$. Outras funções, bem como uma completa exposição sobre *kernels*, podem ser encontradas em (Shawne-Taylor e Cristianini, 2004). O principal problema associado ao uso desses métodos é encontrar um mapeamento ideal para o conjunto em questão.

*Máquinas de vetores suporte (*Support Vector Machines (SVM)*)* De maneira geral, uma SVM (Vapnik, 1999) pode ser vista como uma máquina linear com algumas propriedades interessantes. Uma SVM é uma implementação aproximada do método de minimização de risco estrutural. Este princípio é baseado no fato de que a taxa de erro de um classificador nos dados de teste é limitada pela soma da taxa do erro no treinamento, que depende da dimensão de Vapnik-Chervonenkis (VC) (Müller *et al.*, 2001). No que segue é apresentado uma introdução às SVMs. Considere a equação do hiperplano de decisão dada por: $\mathbf{w}^\top\mathbf{x}_i + b = 0$. Para um determinado vetor de pesos $\mathbf{w}$ e um viés b, a distância entre o hiperplano e a instância mais próxima é chamado de margem de separação, denotado por ρ. O objetivo de uma SVM é encontrar um hiperplano tal que a margem de decisão seja maximizada, chamado hiperplano ótimo e definido por $(\mathbf{w}_0, b_0)$. O hiperplano ótimo deve satisfazer as Equações (2.35).

$$\mathbf{w}_0^\top\mathbf{x}_i + b \geq 1 \quad \text{para} \quad y_i = 1 \quad \text{e} \quad \mathbf{w}_0^\top\mathbf{x}_i + b \leq 1 \quad \text{para} \quad y_i = -1 \tag{2.35}$$

Note que se o conjunto de treinamento for linearmente separável, é possível calcular $\mathbf{w}_0$ e b_0, tal que os termos da equação (2.35) sejam satisfeitos. As instâncias para as quais a primeira ou a segunda equação é satisfeita pela igualdade, são chamadas vetores suporte. Esses vetores são os dados que estão mais próximos da superfície de decisão e por essa razão são os mais difíceis de classificar. Dessa forma, esses dados têm impacto direto na localização do hiperplano separador. Considerando o caso mais geral, em que os dados são linearmente separáveis em um espaço definido pelo mapeamento Φ, como na seção anterior, o problema pode ser resumido da seguinte forma. Dado um conjunto de treinamento, encontrar os valores ótimos para $\mathbf{w}$ e b tal que satisfaçam a Equação (2.36).

$$y_i(\mathbf{w}^\top \Phi(\mathbf{x}_i) + b) \geq 1, \qquad i = 1, \ldots, N \tag{2.36}$$

De modo que o vetor de pesos minimize a função de custo $\Phi(\mathbf{w}) = \frac{1}{2} \parallel \mathbf{w} \parallel^2$. Esta restrição é conhecida como problema primal. Note que, se a única maneira de acessar o espaço de atributos é através do produto interno calculado pelo *kernel*, não é possível minimizar $\Phi(\mathbf{w})$ diretamente, uma vez que $\mathbf{w}$ está no espaço de atributos. Uma maneira de resolver este problema de otimização é utilizar o método de multiplicadores de Lagrange, como na Equação (2.37).

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \parallel \mathbf{w} \parallel^2 - \sum_{i=1}^{n} \alpha_i (y_i(\mathbf{w}^\top \Phi(\mathbf{x}_i) + 1) - 1) \tag{2.37}$$

Dessa forma, minimizando a Equação (2.37), encontram-se os coeficientes $\alpha_i, i = 1, \ldots, n$ que permitem definir a função não linear de decisão.

$$\varphi(\mathbf{z}) = sgn\left(\sum_{i=1}^{n} y_i \alpha_i k(\mathbf{z}, \mathbf{x}_i) + b\right) \tag{2.38}$$

Na equação, a função *sgn* atribui classe positiva caso o resultado da somatória seja maior ou igual a zero, e classe negativa, caso contrário.

2.4 Classificação de dados com distribuição não estacionária

Considere a aplicação de detecção ou filtragem de *spam* em um sistema de *email*. Neste tipo de problema o classificador precisa classificar *email* válido e *spam*, portanto trata-se de uma tarefa de classificação com duas classes. Note que, a descrição de ambas as classes podem variar ao longo do tempo, por exemplo, os emails válidos estão relacionados às preferências dos usuários e estas, na maioria das vezes, também mudam ao longo do tempo. Os emails que são considerados *spam* também estão em constante alteração; estes além de também dependerem das preferências do usuário dependem dos produtores de *spam* (*spammers*) cujo objetivo é passar o *spam* como *email* verdadeiro. Por essa razão, as variáveis usadas na discriminação das classes no tempo t podem não mais ser úteis no tempo $t + k$. Dessa forma, um classificador treinado em um conjunto de treinamento

inicial logo será inutilizado devido a alterações no problema. Esta aplicação é um exemplo de problema cuja distribuição dos dados se altera ao longo do tempo, em outras palavras, a distribuição dos dados é não estacionária.

A natureza não estacionária de certas aplicações deve-se à *mudança de conceito* (*concept drift*) (Schlimmer e Granger, 1986), (Widmer e Kubat, 1996). Conceito refere-se à distribuição dos dados em um determinado período no tempo. Na literatura, no entanto, o termo mudança de conceito têm sido aplicado a diferentes fenômenos relacionados a quedas e recuperações na performance de um sistema classificador (Syed *et al.*, 1999). De fato, outros termos têm sido usados na literatura para referir-se ao mesmo problema, tais como, substituição de conceito (*concept substitutions*), mudanças revolucionárias (*revolutionary changes*), mudança na população (*population drift*) (Kelly *et al.*, 1999), entre outras (ver (Narasimhamurthy e Kuncheva, 2007)). Apesar de existirem diferentes denominações para o termo mudança de conceito, todas referem-se ao mesmo problema que, de acordo com Kelly *et al.* (1999), ocorre devido à alterações nas probabilidades:

- *a priori* das classes $P(\omega_1), \ldots, P(\omega_\mu)$, i.e. alteração no número de exemplos das classes;
- condicional $P(\mathbf{x}|\omega_i)$, i.e. mudança na definição das classes;
- condicional *a posteriori* $P(\omega_i|\mathbf{x})$ i.e. alteração nos atributos, possivelmente em alguns deles;

Em termos gerais, a mudança de conceito pode ser caracterizada no que se refere à variação de conceitos quanto à velocidade e à recorrência. Com relação à velocidade, divide-se basicamente entre mudança gradual (*gradual drift*) e mudança abrupta (*abrupt drift*); quanto à recorrência, uma mudança de conceito é recorrente se conceitos antigos voltam a ser corrente. Esses tipos de mudanças são exemplificadas na Figura 2.8.

Na Figura 2.8, os retângulos azuis representam instâncias que pertencem à classe ω_1 e os círculos vermelhos às instâncias pertencentes a classe ω_2. Considere as Figuras 2.8(a)-(d) como uma sequência de diferentes distribuições de dados no tempo, iniciando em t_0. A mudança de conceito que ocorre entre as distribuições da figura (a) e (b), bem como a que ocorre entre as distribuições nas figuras (b) e (c) são abruptas. Note, no entanto, que a distribuição da figura (c) é similar à da figura (a), ou seja, a distribuição da figura (a) volta a ocorrer, o que caracteriza a mudança de conceito recorrente. Como mostrado na linha de tempo entre as Figuras 2.8(a) e (d) cada distribuição pode, eventualmente, permanecer estática durante um determinado período de tempo, por exemplo a primeira distribuição permanece estática de t_0 a t_i. No entanto, na próxima iteração t_{i+1} a distribuição é totalmente alterada, i.e. ocorreu uma mudança abrupta. A mudança na distribuição dos dados entre as figura (c) e (d), também é considerada abrupta, apesar de menos violenta que as anteriores. Nas Figuras 2.8(e)-(g), considere uma situação em que dois grupos de dados cruzam-se entre si, ao longo do tempo, de modo que a figura (e) corresponda

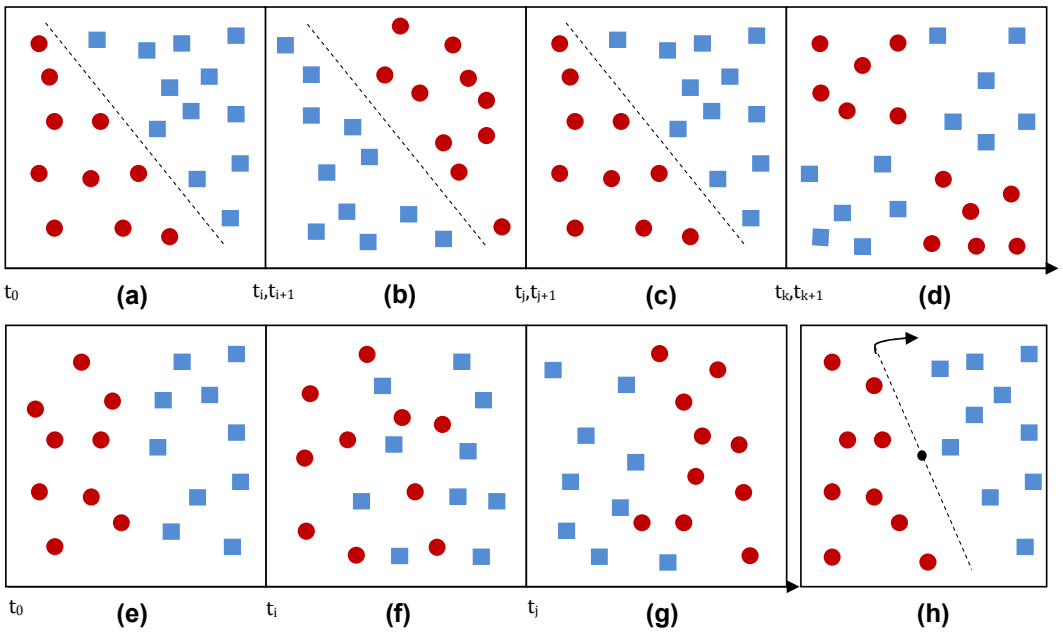

Figura 2.8: Tipos de mudança de conceito, (a)-(d), mudanças abruptas; (e)-(g) mudança gradual e (h) exemplo cuja mudança depende da velocidade em que o hiperplano é alterado.

ao tempo inicial, a figura (g) ao tempo final e a figura (f) a um determinado momento entre ambas. Como a distribuição representada pelas Figuras 2.8(e)-(g), sofre pequenas alterações ao longo do tempo, nesta é caracterizada a mudança de conceito gradual. Por fim, a Figura 2.8(h), representa uma distribuição determinada por um hiperplano que gira sobre um eixo ao longo do tempo. Considere que o hiperplano sofra uma inclinação de $\pi/4$ a cada determinado período de tempo, como este movimento é regular, trata-se uma mudança de conceito gradual e recorrente a cada 8 alterações no hiperplano. Note que, se a taxa for aumentada, por exemplo para π, as mudanças podem ser consideradas abruptas.

Note que é difícil caracterizar as diversas variações com relação à velocidade e recorrência, utilizando somente os termos, gradual, abrupto e recorrente. Em vista deste problema, Minku *et al.* (2010) propuseram uma categorização para tipos de mudanças de conceito em dez classes, sendo que duas delas possuem subclasses. Para a categorização os autores levaram em conta outros critérios além de velocidade e recorrência, tais como, frequência, que mede o quão frequente ocorre um desvio de conceito, e severidade, que mede a quantidade de mudanças de conceito ocorridas em um fluxo de dados, entre outras. Por tratar-se de um trabalho recente, não se pode afirmar que estas categorias serão bem aceitas na comunidade cientifica, no entanto trata-se de um esforço em direção à padronização e categorização de um importante tópico de pesquisa.

Ainda com relação ao problema de mudança de conceito Klinkenberg e Joachims (2000) alertaram sobre a possibilidade de detecção de falsos desvios de conceito, chamado de mudança de conceito virtual (*virtual concept drift*). Os autores demonstraram que quedas e recuperações no desempenho de um classificador, que a princípio poderia ser

qualificado como mudança de conceito, pode ocorrer em domínios de conceito estável devido à variação na amostragem dos dados. E que, nesta situação, é difícil dizer se o classificador está de fato diante de uma situação de mudança de conceito ou de nova informação de um conceito estático - causado por má distribuição nas amostragens.

2.4.1 Aprendizado incremental

Muitas aplicações reais requerem classificação em tempo real (incremental), no qual o sistema classificador precisa processar novas instâncias de dados no momento em que são apresentados (Yang e Zhou, 2008). Dentre vários outros exemplos, estão detecção de intrusos (Lane e Brodley, 1998) detecção de fraude em sistemas de cartão de crédito (Wang *et al.*, 2003), problemas relacionados a finanças (Last, 2002), inspeção ultrassônica de solda (Polikar *et al.*, 2004), separação de sinais acústicos (Murata *et al.*, 2002).

Giraud-Carrier (2000) diferencia *tarefa incremental* de *algoritmo incremental*, de forma que, uma tarefa de classificação é incremental quando as instâncias de treinamento, usadas em sua representação, são disponibilizadas ao longo do tempo, geralmente uma de cada vez.

Um algoritmo é incremental se para uma sequência de instâncias de treinamento, $\mathbf{x}_1, \ldots, \mathbf{x}_N$ o algoritmo produz uma sequência de classificadores diferentes, que depende do classificador anterior e das instâncias adicionadas. Em outras palavras, a adição de conhecimento, provindo de um novo exemplo de dado, altera o classificador, sendo que a alteração no classificador depende de como este está organizado internamente.

Note que ambas as definições não fazem restrição quanto ao número de instâncias processadas por vez, i.e. individualmente ou em conjuntos. No entanto, alguns autores consideram o processamento individual de instâncias de dados como aprendizado em tempo real. De fato, não há um consenso geral sobre tais nomenclaturas, neste texto será considerada a distinção feita por Minku *et al.* (2010), que considera o termo aprendizado incremental como termo geral que não faz restrições quanto à tarefa em questão - portanto vale as definições previamente citadas. Já o termo, aprendizado em tempo real, é considerado mais específico e refere-se somente à classificação onde o processamento (adição de conhecimento) de novas instâncias é realizado de maneira individual.

Vale ressaltar que, em princípio, qualquer tarefa incremental pode ser resolvida, por qualquer algoritmo estático. A solução trivial consiste em retreinar o classificador toda vez que novos exemplos são disponibilizados ou a cada determinado período de tempo. No entanto, esta prática apresenta ao menos três empecilhos, (i) retreinar um classificador, por completo, em geral é um processo custoso computacionalmente (ii) aceitando a proposta de retreinar o classificador, o problema é definir o espaço de tempo para que a substituição do classificador, onde espaço curto implica em maior gasto com treino e frequente perda de dados para classificação, ao passo que longos espaços entre a troca de classificador economiza tempo computacional mas torna-se mais vulnerável à mudança de conceito, o que deteriorará o desempenho do classificador. (iii) determinar o momento da

troca do classificador. i.e. eliminar o classificador corrente por um novo classificador, sem considerar o conjunto em que este será treinado. A taxa de acertos não é confiável para esta tarefa uma vez que esta pode variar se ocorre uma mudança de conceito.

De acordo com Widmer e Kubat (1996), para que o aprendizado incremental seja eficiente em ambientes com mudanças de conceito, um algoritmo de aprendizado deve ser capaz de detectar automaticamente mudanças no conceito, recuperar rapidamente o desempenho de classificação, ajustar-se ao novo contexto e fazer uso de experiências prévias em situações onde conceitos antigos reaparecem. Geralmente, um algoritmo de aprendizado incremental precisa de, 1) operadores que permitam alterar o conceito corrente; 2) a habilidade de decidir quando e quantos conceitos antigos devem ser esquecidos e 3) uma estratégia para manter armazenado conceitos corrente e acessá-los quando necessário.

Algoritmos de aprendizado incremental podem ser categorizados com relação à maneira que a memória é utilizada no tratamento de instâncias de dados antigos. Segundo Reinke e Michalski (1988) pode-se estabelecer três categorias. 1) Sem armazenamento de instâncias - algoritmos incrementais que não armazenam os exemplos de dados na forma que são apresentados, tais como, os algoritmos STAGGER (Schlimmer e Granger, 1986), VFDT (Domingos e Hulten, 2000), VFDTc (Gama *et al.*, 2003), entre outros. Geralmente, algoritmos baseados em redes neurais (Saad, 1999), naive bayes, etc., que não armazenam as instâncias propriamente, mas as utilizam para construir o classificador; 2) armazenamento total de instâncias - métodos que armazenam todas as instâncias apresentadas, tais como, o IBk (Aha *et al.*, 1991), GEM (Reinke e Michalski, 1988), o ID5 Utgoff (1988), ITI (Utgoff *et al.*, 1997), entre outros e; 3) armazenamento parcial de instâncias - métodos que armazenam alguns dos exemplos já apresentados, tais como, métodos baseados em janelas, o LAIR (Elio e Watanabe, 1991), o AQ11-PM (Maloof e Michalski, 2004) e FLORA (Widmer e Kubat, 1996).

Além da categorização quanto à manutenção da memória com relação a exemplos de dados, Maloof e Michalski (2000) e Maloof e Michalski (2004) prepuseram categorizar os algoritmos incrementais com relação ao armazenamento da descrição de conceitos passados. Por descrição de conceito entende-se algum tipo de estrutura que sintetiza um conceito, assim como um nó em uma árvore de decisão, ou uma regra em um sistema especialista. Esta segunda categorização, também pode ser dividida em três, a saber, 1) sem memória de conceito - algoritmos que não armazenam conceito, por exemplo, alguns algoritmos baseados em instâncias, tais como IB1 e IB2 Aha *et al.* (1991), não formam uma descrições de conceitos que generalizam as instâncias pertencentes a este conceito; 2) memória de conceito total - refere-se a sistemas iterativos, como o ITI (Utgoff *et al.*, 1997), descrevem os conceitos por meio dos conjuntos de treinamento, que permanecem na memória até que outro conjunto seja apresentado. Por último, 3) os algoritmos de memória parcial de conceito, no qual armazenam parte dos conceitos apresentados, em geral desfazem-se de conceitos antigos. Por exemplo, o algoritmo FAVORIT (Kubat e Krizakova, 1992) induz árvores a partir dos dados, mas mantém um peso para

cada nó da árvore. Esses pesos são atualizados com a chegada de novos dados de treinamento, e dependem desse reforço para manterem-se na árvore, pois esses pesos sofrem um decréscimo com a passagem do tempo e são retirados da árvore quando decrescem abaixo de certo limiar.

Note que as categorizações são distintas e referem-se a dois critérios diferentes para classificar algoritmos incrementais. Uma abordagem mais profunda sobre essas categorias,bem como, outros algoritmos que nestas se encaixam, pode ser encontrado no trabalho de Maloof e Michalski (2004).

2.4.2 Algoritmos de aprendizado incrementais

Esta seção apresenta alguns algoritmos de aprendizado para domínios de dados não estacionários, ou seja, domínios cuja distribuição dos dados de teste podem sofrer alterção ao longo do tempo. Como visto nas seções anteriores, este fenomeno é conhecido como mudança de conceito e pode ser resolvido por meio da aplicação de algoritmos que continuam aprendendo após a fase de treinamento inicial. Como alguns dos algorimos baseiam-se em comitê, considere as seguintes notações. Um comitê, notado por $H = \{h_1, \ldots, h_\varepsilon\}$, pode ser formado por até ε classificadores base, onde ε em alguns casos é determinado pelo usuário. Cada classificador h_k do comitê, é chamado de classificador base, e a saída k-ésimo classificador h_k, em reposta ao exemplo $\mathbf{z}$, é notada como $h_k(\mathbf{z})$. Os algoritmos recebem como entrada um fluxo de dados $S = \{X_1, Z_1, \ldots, X_T, Z_T\}$, onde $X_t = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ representa um conjunto com dados rotulados $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip}, y_i)$, e $Z_t = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_M\}$ representa o t-ésimo conjunto não rotulado no qual, $\mathbf{z}_j = (z_{j1}, z_{j2}, \ldots, z_{jp})$. Os algoritmos também utilizam a variável λ que assume um valor de classe e é usada como índice do vetor ϱ que representa as classes. A classe atribuída ao exemplo $\mathbf{z}$ pelo comitê é representada por $\varphi_H(\mathbf{z})$. No que segue são apresentado os quatros algortimos de aprendizado incremental que serão utilizados como comparativos para o algoritmo KAOGINC, apresentado no Capítulo 4.

2.4.2.1 O algoritmo SEA

O algoritmo baseado em comitê para fluxo de dados (*Streaming Ensemble Algorithm* (SEA)) (Street e Kim, 2001), notado aqui simplesmente como SEA, consiste em um comitê de classificadores, mais especificamente árvores de decisão (C4.5). Como visto na Seção 2.3.1, um comitê nada mais é do que um grupo de classificadores com uma política para a tomada de decisão final. No caso do algoritmo SEA, a tomada de decisão para a saída do comitê, consiste, simplesmente, na maioria dos votos. Dado um exemplo de teste, todas as árvores do comitê avaliam e atribuem uma classe como saída. A saída do comitê então será a classe mais comum entre as saídas dos classificadores.

O algoritmo SEA é puramente incremental, pois em sua versão original não aceita exemplos de dados isolados como entrada. Uma vez que é necessário um conjunto para treinar uma árvore de decisão. Neste algoritmo, uma nova árvore de decisão é treinada

toda vez que um novo conjunto de treinamento é apresentado. Como o algoritmo prevê um número fixo de classificadores no comitê, na adição de uma nova árvore ao comitê, duas situações podem ocorrer. Se o número máximo de classificadores ainda não foi atingido, o novo classificador é adicionado. Caso o número máximo tenha sido atingido, o novo classificador substituirá a árvore que mais cometeu erros nos conjuntos de testes. Por meio desta troca, o comitê garante estar sempre atualizado com novos conceitos. Vale ressaltar que, no algoritmo SEA, as árvores já adicionadas ao comitê não têm acesso a novo conhecimento. O Algoritmo 2.1, mostra como o comitê é formado e atualizado ao longo da apresentação do fluxo de dados.

Algoritmo 2.1 Algoritmo SEA

Entrada: $S = \{X_1, Z_1, ..., X_T, Z_T\}$ {Fluxo de dados, conjuntos rotulados e não rotulados}
$X_t = \{\mathbf{x}_1, ..., \mathbf{x}_N)\}$ {Conjuntos rotulados apresentados ao longo do tempo}
$Z_t = \{\mathbf{z}_1, ..., \mathbf{z}_M\}$ {Conjuntos não rotulados apresentados ao longo do tempo}
ε {Número de classificadores no comitê}
Saída: $\varphi_H(\mathbf{z})$ {classe para os dados não rotulados}

```
repita
    se X_t então
        avaliaComite(X_t)
        se t < ε então
            H(h_t) ⇐ treinaClassificador(X_t)
        senão
            h_k ⇐ encontraPior(H)
            H(h_k) ⇐ treinaClassificador(X_t)
        fim se
    senão
        para todo z_j ∈ Z_t faça
            ϱ ⇐ 0
            para k = 1 até ε faça
                λ ⇐ h_k(z_j)
                ϱ_λ ⇐ ϱ_λ + 1
            fim para
            φ_H(z_j) ⇐ {ω_λ|argmax(ϱ_λ)}
        fim para
    fim se
até que S = ∅
```

No algoritmo o método *avaliaComite()* classifica o conjunto rotulado X_t antes que este seja usado para o treinamento e armazena as estatísticas sobre o desempenho dos classificadores do comitê. Essas estatísticas serão usadas no método *encontraPior()* para determinar o algoritmo base com menor porcentagem de acerto. Uma vez que o comitê esteja com sua ocupação máxima, o pior classificador é retirado para que um novo classificador, treinado com o conjunto X_t, seja adicionado. No algoritmo o método *treinaClassificador()* utiliza o algoritmo C4.5 para treinar uma árvore de decisão. A classificação de novos exemplos de dados é feita aplicando-se o novo exemplo a todos os classificadores base no comitê, resultando em ε classificações, onde a classe de $\mathbf{z}_j$ atribuída pelo classificador h_i é atribuída à variável λ que é usada como índice para o vetor que representa as classes ϱ. O comitê é resolvido por simples votação, i.e. a classe mais frequente atribuída à $\mathbf{z}_j$.

2.4.2.2 O algortimo DWM

O Algoritmo (*Dinamic Weighted Majority*(DWM)) (Kolter e Maloof, 2007), assim como o algoritmo SEA, também consiste na construção de um comitê. Entretanto, o algoritmo DWM cria um comitê que, segundo os autores, virtualmente pode ser composto por qualquer classificador - com a ressalva de que os classificadores base suportem aprendizado incremental. Kolter e Maloof (2007), todavia, propõem o uso de dois algoritmos em especial. Um deles consiste em uma versão incremental do C4.5 proposto por Utgoff *et al.* (1997); o outro, em uma versão incremental do algoritmo Naive Bayes, utilizado neste trabalho e apresentado adiante nesta seção. Basicamente, esses algoritmos são modificações de suas versões originais, capazes de adquirir conhecimento à medida que novos dados são disponibilizados.

O algoritmo DWM atualiza o comitê a cada ρ iterações; a atualização corresponde à remoção de classificadores, atualização de pesos e adição de novo classificador. Se, no momento em que a atualização é considerada o comitê atual comete um erro, um novo classificador incremental é adicionado ao comitê. O novo classificador tem como treino, possivelmente, apenas uma instância, sendo que esta também é adicionada a todos os classificadores do comitê. Cada classificador base tem um peso associado, w, que é decrementado, por um fator pré-determinado, β, toda vez que o classificador base comete um erro. A remoção retira do comitê os classificadores cujo peso é menor que um limiar θ ($\theta = 0,01$ - como sugerido pelos autores). O comitê no algoritmo DWM, diferente do SEA, não tem limite de classificadores, o que significa que seu tamanho pode aumentar substancialmente, especialmente na presença de ruído. O controle do tamanho do comitê é feito pelo parâmetro ρ que define a frequência de adição e remoção de classificadores base. A cada ρ intervalos de iterações além da adição e remoção também ocorre a normalização dos pesos dos classificadores remanescentes, para que os classificadores recém adicionados não levem vantagem na solução do comitê.

O processo de resolução do comitê envolve a predição individual dos classificadores bases e seus pesos. Para definir a saída do comitê somam-se os pesos dos classificadores que predisseram a mesma classe, obtendo um resultado para cada classe. A classe atribuída ao exemplo de teste $\mathbf{z}$ será a que obteve a maior soma dos pesos. Apesar do uso de pesos, e de poder controlar a inserção e remoção de classificadores, a principal característica do DWM é a possibilidade de atualizar todos os classificadores do comitê com novos dados. O Algoritmo 2.2 apresenta em detalhes o algoritmo DWM.

No algoritmo, o método $treinaClassificador()$ corresponde a algum método de classificação em tempo real, uma vez que o algoritmo exige que os classificadores base aprendam cada nova instância de dado individualmente. Nos testes realizados neste trabalho, o algoritmo escolhido é o Naive Bayes incremental, por essa razão o algoritmo DWM será notado como DWM-NB. O método $normalizaPesos()$ normaliza os pesos dos classificadores no comitê. Os pesos são normalizados atribuindo-se ao maior peso, o valor 1 e aos demais, somando-se a diferença entre o maior peso e 1. O método $removeClassificador()$, por

Algoritmo 2.2 Algoritmo DWM

Entrada: $S = \{X_1, Z_1, ..., X_T, Z_T\}$ {Fluxo de dados, conjuntos rotulados e não rotulados}
$X_t = \{\mathbf{x}_1, ..., \mathbf{x}_N)\}$ {Conjuntos rotulados apresentados ao longo do tempo}
$Z_t = \{\mathbf{z}_1, ..., \mathbf{z}_M\}$ {Conjuntos não rotulados apresentados ao longo do tempo}
ρ {Período entre atualizações do comitê}
$\beta = 0,05$ {fator de decremento dos pesos}
Saída: $\varphi_H(\mathbf{z})$ {classe para os dados não rotulados}

```
repita
    se X_t então
        para todo x_i ∈ X faça
            ϱ ⇐ 0
            para todo h_k ∈ H faça
                λ ⇐ h_k(x_i)
                se λ ≠ y_i então
                    w_k ⇐ βw_k
                fim se
                ϱ_λ ⇐ ϱ_λ + w_j
            fim para
            φ_H(x_i) ⇐ {ω_λ|argmax(ϱ_λ)}
        fim para
        se k mod ρ = 0 então
            normalizaPesos(H)
            removeClassificador(H)
            se φ_H(x_i) ≠ y_i então
                ε ⇐ ε + 1
                H(h_ε) ⇐ treinaClassificador(x_i)
                w_ε ⇐ 1
            fim se
        fim se
    senão
        para todo z_j ∈ Z_t faça
            ϱ ⇐ 0
            para todo h_k ∈ H faça
                λ ⇐ h_k(z_j)
                ϱ_λ ⇐ ϱ_λ + w_k
            fim para
            φ_H(z_j) ⇐ {ω_λ|argmax(ϱ_λ)}
        fim para
    fim se
até que S = ∅
```

sua vez, retira do comitê os classificadores que cometeram muitos erros. Note que o algoritmo utiliza-se da classificação de dados rotulados para a atualização dos pesos e do comitê.

O Naive Bayes incremental (ver (Witten e Frank, 2005) para maiores detalhes) assim como na versão original, utiliza-se da contagem das classes no de treinamento para estimar as probabilidades *a priori* de cada classe $P(\omega_i)$. Bem como, utiliza-se da contagem, para atributos nominais, para estimar a probabilidade condicional $P(x_j|\omega_i)$ para o valor de atributo x_j dado a classe ω_i. A principal diferença considerando o algoritmo incremental, é que ao invés de utilizar um conjunto de treinamento estático, o mesmo atualiza a contagem das classes e atributos toda vez que uma nova instância é apresentada. Dessa forma, a classe da nova instância é determinada de acordo o algoritmo Naive Bayes, como

mostra a Equação (2.39).

$$\varphi(\mathbf{z}) = arg\max_{\omega_i} P(\omega_i) \prod_i P(z_j|\omega_i) \tag{2.39}$$

Para atributos numéricos, o algoritmo armazena a soma de cada atributo bem como a soma do quadrado de cada atributo - para o cálculo do desvio padrão. Dessa forma, o cálculo da probabilidade condicional é feito de acordo com a Equação (2.40).

$$P(x_j|\omega_i) = \frac{1}{\sigma_{ij}\sqrt{2\pi}} e^{-(x_j-\nu_{ij})^2/2\sigma_{ij}^2} \tag{2.40}$$

Na equação, ν_{ij} corresponde a média do atributo x_j para os dados pertencentes à classe ω_i, e σ_{ij} o desvio padrão.

2.4.2.3 O algoritmo WCEA

Wang *et al.* (2003) propuseram um classificador baseado em comitê similar ao algoritmo SEA. O algoritmo, conhecido por *Wang's Classifier Ensemble Approach* (WCEA), no entanto, utiliza pesos para resolver o comitê. O peso para o classificador h_i é estimado pela sua porcentagem de acertos em um conjunto de teste. A ideia é que o último conjunto de dados reflete melhor a distribuição atual dos dados. De forma que, os pesos dos classificadores podem ser aproximados calculando-se o erro no conjunto rotulado mais recente. O peso para o classificador h_k, w_k é determinado como $w_k = MSE_r - MSE_i$, onde MSE_i corresponde ao erro atual e pode ser obtido por meio da validação cruzada no último conjunto rotulado X_t, e MSE_r é o e erro estimado, dado as instâncias armazenadas, podendo ser definido como na Equação (2.41).

$$MSE_r = \sum_{\omega_j \in \Omega} p(\omega_j)(1 - p(\omega_j))^2 \tag{2.41}$$

Toda vez que um conjunto rotulado é disponibilizado, o algoritmo treina um novo classificador, no caso, utilizando o algoritmo C4.5 e o adiciona ao comitê. Assim como no algoritmo SEA, o comitê tem um tamanho pré-definido ε, e uma vez que todo o espaço esteja ocupado, a atualização procede substituindo o pior classificador base (aquele com o menor peso) no comitê pelo classificador recém-treinado, como mostra o Algoritmo 2.3.

A classificação é realizada apresentando o exemplo a ser classificado, $\mathbf{z}$, para os algoritmos base do comitê. Esses o classificam de acordo com uma das μ classes. O comitê é então resolvido somando os pesos dos classificadores para as diferentes classes, a classe que resultar em maior peso é atribuída ao novo exemplo.

No algoritmo o método $atualizaPeso()$ determina as duas quantidades, MSE_r e MSE_i para calcular o peso do classificador passado como parâmetro. O erro MSE_r é determinado de acordo com a Equação 2.41 e depende da distribuição das classes no novo conjunto de treinamento. Já o erro MSE_i é estimado por meio de validação cruzada, o que além de invibializar o algoritmo de processar conjuntos de dados pequenos, também consome

Algoritmo 2.3 Algoritmo WCEA

Entrada: $S = \{X_1, Z_1, ..., X_T, Z_T\}$ {Fluxo de dados, conjuntos rotulados e não rotulados}
$X_t = \{\mathbf{x}_1, ..., \mathbf{x}_N)\}$ {Conjuntos rotulados apresentados ao longo do tempo}
$Z_t = \{\mathbf{z}_1, ..., \mathbf{z}_M\}$ {Conjuntos não rotulados apresentados ao longo do tempo}
ε {Número de classificadores no comitê}
Saída: $\varphi_H(\mathbf{z})$ {classe para os dados não rotulados}

```
repita
    se X_t então
        para todo h_k ∈ H faça
            atualizaPesos(h_k, X_t)
        fim para
        se t < ε então
            H(h_t) ⇐ treinaClassificador(X_t)
            atualizaPesos(h_t, X_t)
        senão
            h_k ⇐ encontraPior(H)
            H(h_k) ⇐ treinaClassificador(X_t)
            atualizaPesos(h_k, X_t)
        fim se
    senão
        para todo z_j ∈ Z_t faça
            σ ⇐ 0
            para k = 1 até ε faça
                λ ⇐ h_k(z_j)
                ϱ_λ ⇐ ϱ_λ + w_k
            fim para
            φ_H(z_j) ⇐ {ω_λ | argmax(ϱ_λ)}
        fim para
    fim se
até que S = ∅
```

tempo na estimativa do erro - realizado para todo classificador do comitê.

2.4.2.4 O algoritmo OnlineTree2

Diferente dos algoritmos previamente descritos, o algoritmo OnlineTree2 (Núnez *et al.*, 2007) não é baseado em comitê. Ao invés disso, o algoritmo baseia-se em uma única árvore de decisão que se ajusta automaticamente durante o processamento do fluxo de dados. O algoritmo, basicamente, consiste em atualizar a árvore de decisão, podendo esta ser treinada inicialmente com qualquer algoritmo que produza uma árvore binária de decisão.

Para a atualização da árvore, cada nó possui alguns atributos, por exemplo tipo de nó (decisão ou folha), estatística de desempenho, tamanho do conjunto de dados que deve ser mantido, entre outros. Durante o processamento, cada instância de entrada é processada individualmente, tornado o algoritmo apto para aprender em tempo real. Cada nova instância é processada pela raiz e conduzida pelos nós filhos, de acordo com seus valores de atributo, até o nó folha correspondente. A cada nó intermediário em que a nova instância é apresentada, este é checado quanto a sua coerência. Um nó é dito coerente se a divisão dos dados realizada por ele contribui na indução do conceito corrente, a coerência é medida por meio da distribuição χ^2 (Esposito *et al.*, 1997), com grau de significância $0,05$.

Se o nó atingido for folha, então o tratamento de folha entra em processo, inicial-

mente a instância é armazenada no conjunto da folha e suas estatísticas são atualizadas, incluindo a performance local. Se a performance permanece estável ou é incrementada, o algoritmo tenta dividir esta folha com o objetivo de melhor adaptar ao subconceito, novamente baseando-se na distribuição χ^2. Se o nó folha tiver sua performance reduzida, o algoritmo tenta reduzir sua janela de tempo. O Algoritmo 2.4 detalha o processo de atualização da árvore de decisão. O algoritmo pode ser dividido em três estágios, revisão das estatísticas no sentido raiz-folhas, tratamento de folha ou nó não coerente, e atualização das estatísticas no sentido folha-raiz.

Algoritmo 2.4 Algoritmo OnlineTree2

Entrada: $S = \{X_1, Z_1, ..., X_T, Z_T\}$ {Fluxo de dados, conjuntos rotulados e não rotulados}
$X_t = \{\mathbf{x}_1, ..., \mathbf{x}_N)\}$ {Conjuntos rotulados apresentados ao longo do tempo}
$Z_t = \{\mathbf{z}_1, ..., \mathbf{z}_M\}$ {Conjuntos não rotulados apresentados ao longo do tempo}
Saída: $\varphi_H(\mathbf{z})$ {classe para os dados não rotulados}

```
se X_t então
    se nó não é folha e sua divisão é util então
        nó ⇐ OnlineTree2(proximoNo(No, x_i), x_i)
    senão
        se nó não é folha então
            proximoNo(x_i)
            chamadaRecursiva(AjustJanelaEstDegradado(nó))
            chamadaRecursiva(Poda(nó))
            atualizaEstatisticas(nó)
            tentaExpandir(nó)
        senão
            armazenaEmFolha(x_i)
            atualizaEstatistica(folha)
            se folha em estado de melhora então
                tentaExpandir(folha)
                se folha então
                    ajustaJanelaEstMelhorado(folha)
                fim se
            senão
                ajustJanelaEstDegradado(folha)
            fim se
        fim se
        retorne nó ou folha
    fim se
    AtualizaEstatistica()
senão
    para todo z_j ∈ Z_t faça
        φ(z_j) ⇐ percorreArvore(z_j)
    fim para
fim se
```

O primeiro estágio é recursivo e chega ao fim quando a instância atinge um nó não coerente ou uma folha. Se o primeiro caso se confirma, inicia-se o processo de tratamento de nó não coerente e o algoritmo passa para o segundo estágio. Neste estágio, primeiramente, a instância é deixada na folha referente a ela ($proximoNo()$). Depois, todas as folhas abaixo do nó não coerente e cuja performance está piorando, apagam os exemplos marcados para serem apagados e diminuem o conjunto dados. Então o nó não coerente é transformado em folha com os exemplos restantes da subárvore removida ($poda()$), finalmente, uma nova tentativa de dividir o nó é realizada ($tentaExpandir()$). Se, ao invés

disso, a instância atingir um nó folha passando por nós intermediários coerentes, o algoritmo verifica a possibilidade de expansão da folha ($tentaExpandir()$) e atualiza suas estatísticas de desempenho.

A tentativa de expansão considera o ganho de informação normalizado, assim como no C4.5, para escolher qual atributo será usado para dividir o nó. Mas diferente do C4.5, os autores propõem utilizar o algoritmo *k-means* (MacQueen, 1967) para dividir o atributo escolhido em dois, com o objetivo de reduzir a ordem de complexidade. Os métodos $ajustaJanelaEstMelhorado()$ e $ajustaJanelaEstDegradado()$ ajustam o tamanho do conjunto de dados do nó passado como parâmetro, aumentanto, no primeiro caso, e diminuindo, de acordo com a performance, no segundo. O método $chamadaRecursiva()$ corresponde à chamada do método indicado para toda a subárvore do nó referente à primeira chamada.

Como mencionado o algoritmo OnlineTree2 utiliza-se da performance local de cada folha para atualizar a árvore e calcular o tamanho do conjunto de dados em cada folha. O cálculo da performance usa a taxa entre o número de exemplos classificados corretamente e o número total de exemplos em cada folha. No entanto, esta taxa pode variar muito e torna-se inapropriada para tomar decisões. Ao invés disso é utilizado um método de suavização exponencial, como na Equação (2.42).

$$perf(t) = \frac{7}{8} perf(t-1) + \frac{1}{8} di \tag{2.42}$$

Na qual $perf(t)$ é a performance no tempo t e di é o desempenho instantâneo da folha, que corresponde à taxa de instâncias da classe que a folha representa em relação ao número de instâncias na folha. De maneira sucinta, quando a folha apresenta piora na performance, os exemplos mais antigos são apagados dando espaço a novos exemplos de um possível novo conceito. Uma vez terminado o segundo estágio, tratando folha ou nó não coerente, o algoritmo volta pelo mesmo caminho atualizando as estatísticas dos nós visitados, completando o terceiro estágio.

O processo de classificação é realizado pelo método $percorreArvore()$ que percorre a árvore da raiz à alguma folha, que determina a classe, como em uma simples árvore de decisão.

Capítulo 3

Classificação de dados baseado no grafo K-associado ótimo

Neste capítulo é apresentado a construção do classificador baseado no grafo K-associado ótimo, *K-Associated Optimal Graph* (*KAOG*), proposto para tratar problemas de classificação de dados com distribuição estacionária (Lopes *et al.*, 2009) (Bertini Jr. *et al.*, 2011b). A base da classificação está em representar o conjunto de treinamento como um grafo (ou rede), referenciado como grafo K-associado. Essa estrutura não apenas pode armazenar relações de similaridade entre os dados, mas também, por meio do grafo K-associado, é possível derivar algumas propriedades interessantes que relacionam dado e ruído, como a medida de pureza para componentes. A pureza, que será definida adiante, utiliza a estrutura do grafo para determinar localmente o nível de mistura dos dados com relação a suas classes.

Um grafo K-associado é construído a partir de um conjunto de dados vetoriais do tipo atributo-valor abstraindo as instâncias de dados em vértices e as similaridades de K-vizinhança, entre eles, em arestas. De forma que, a estrutura do grafo depende da distribuição dos dados e do valor de K. Portanto, diferentes valores de K produzem diferentes grafos com componentes de tamanho e pureza diferentes. Seja o grafo resultante da seleção dos componentes de maior pureza, pertencentes à vários grafos K-associados, nomeado grafo K-associado ótimo, por conter os componentes de maior pureza. Esta estrutura em grafo é usada pelo classificador para estimar a classe de um novo dado. A classificação consiste em estimar a probabilidade de pertinência para todo componente no grafo, então a classe é atribuída verificando-se a máxima probabilidade *a posteriori*. Portanto, o treinamento do classificador consiste na construção do grafo K-associado ótimo e na estimativa da pureza para os seus diversos componentes.

No restante do capítulo serão apresentadas em detalhes as três etapas na construção do

grafo ótimo[1], a saber, 1) o grafo K-associado - como construí-lo a partir de um conjunto de dados; 2) a medida de pureza - como calcular a pureza de cada componente e; 3) o grafo ótimo - como obter o grafo ótimo a partir dos grafos K-associados. Também é apresentado a construção do classificador que se baseia na estrutura do grafo para estimar a classe de novos dados, bem como um exemplo completo de classificação; seguido pela seção que determina a complexidade computacional de todo o processo. Cabe ressaltar que a construção do grafo, bem como a do classificador, não requer parâmetros e, dessa forma, não exige seleção de modelos. O capítulo também apresenta alguns experimentos em dados artificiais mostrando as principais propriedades dos grafos, K-associados e K-associado ótimo, bem como, da medida de pureza. Por fim, o algoritmo KAOG é comparado com cinco outros algoritmos em quinze domínios multiclasse, obtidos do repositório UCI (Asuncion e Newman, 2007), seguidos pela análise estatística dos resultados.

3.1 O grafo K-associado

Um grafo, gerado a partir de um conjunto de dados, cujo objetivo seja representar os padrões e as relações entre os dados, deve herdar as principais características presentes nos dados. Usualmente, cada instância no conjunto de treinamento é mapeado para um vértice no grafo e as conexões são estabelecias de maneira a preservar as relações de similaridades desejadas.

Considere o conjunto de treinamento $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ no qual cada instância $\mathbf{x}_i$ possui um rótulos representado por y_i e associado a uma das classes $\Omega = \{\omega_1, \omega_2, ..., \omega_\mu\}$, sendo μ o número de classes. O grafo K-associado pode ser construído como segue. Inicialmente, cada instância de entrada $\mathbf{x}_i$ é representado por um vértice v_i, e de maneira similar y_i representa a classe de v_i. O passo seguinte resume em conectar todo vértice v_i à todos os seus K vizinhos mais próximos que pertençam à mesma classe, ou seja, qualquer outro vértice v_j será conectado ao vértice v_i, se e semente se, ambas condições forem satisfeitas: 1) a instância representada por v_j é um dos K vizinhos mais próximos da instância representada por v_i, segundo uma determinada medida de similaridade; 2) as instâncias representadas por v_i e v_j pertencem à mesma classe.

O grafo K-associado pode ser visto como uma variante do grafo KNN, pelo fato de ambos considerarem relações de vizinhança do tipo vizinho mais próximo. No grafo K-associado, entretanto, dentre os K vizinhos de um vértice v_i, somente serão conectados os que possuem a mesma classe que v_i, e, além disso, no grafo K-associado as conexões duplas são mantidas. Considerando v_i e v_j, vértices de uma mesma classe, uma conexão dupla ocorre quando o vértice v_j seleciona o vértice v_i para conectar e, todavia, v_i já se encontra conectado a v_j, neste caso, a nova conexão é estabelecida resultando em duas conexões não direcionadas entre os vértices v_i e v_j. A Figura 3.1 ilustra as diferenças entre os grafos KNN e K-associado para o mesmo valor de K ($K = 2$).

[1] No texto, 'grafo ótimo' é usado em alusão ao termo 'grafo K-associado ótimo'.

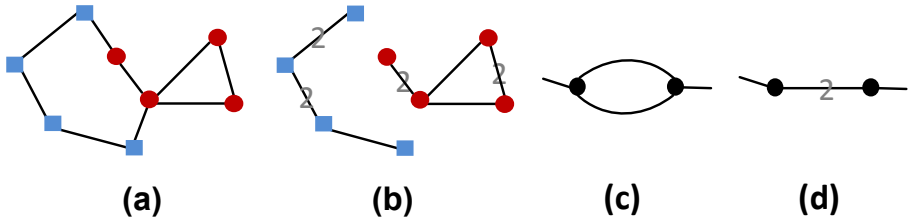

Figura 3.1: Diferenças no padrão de conexão entre vértices considerando os algoritmos: (a) grafo KNN e (b) grafo K-associado. As figuras (c) e (d) representam conexões duplas.

Nas Figuras 3.1(a) e (b) os vértices representam duas classes (retângulos azuis e círculos vermelhos), na Figura 3.1(a) é ilustrado o grafo 2NN, que não considera informação de classe nem conexões duplas. A Figura 3.1(b) mostra o grafo 2-associado formado pelo mesmo conjunto de vértices. Note que, o último considera diferenças entre classes e ligações duplas. Esta maneira em particular de conectar os vértices produz um multigrafo com arestas não direcionadas e é fundamental para a definição da pureza, como será mostrado a frente. A Figura 3.1(c) ilustra uma notação comumente usada para conexões duplas e a Figura 3.1(d) a notação, para o mesmo tipo de conexão, que será usada ao longo do texto.

Assim que todos os vértices forem considerados para conectar seus K vizinhos mais próximos, a construção do grafo K-associado chega ao fim. Neste estágio, cada classe é representada por um ou mais componentes, em outras palavras, o número de componentes será sempre maior ou igual ao número de classes. Note que conforme o valor de K aumenta, o número de componentes decresce monotonicamente até convergir para o número de classes.

De maneira formal, o grafo K-associado $G^{(K)} = (V, E)$ consiste de um conjunto de vértices $V = \{v_1, \ldots, v_N\}$, sendo N o número de instâncias no conjunto de treinamento, e um conjunto de arestas E. O vértice v_i está associado à instância $\mathbf{x}_i$, no qual uma aresta e_{ij} conecta o vértice v_i com o vértice v_j se $y_i = y_j$ e $v_j \in \Lambda_{v_i,K}$, onde $\Lambda_{v_i,K}$ representa o conjunto dos K vizinhos mais próximos de v_i segundo uma medida de similaridade.

Considerando as definições prévias, nota-se que:

1. Um grafo K-associado pode ser visto como um conjunto de componentes $C_\alpha \in C = \{C_1, ..., C_R\}$ e $\alpha = \{1, 2, ..., R\}$, onde α é índice de componente e R é o número de componentes no grafo.

2. Para cada par de vértices v_i e v_j, se $y_i = y_j$ e $v_j \in \Lambda_{v_i,K}$ ou $v_i \in \Lambda_{v_j,K}$ ambos os vértices estarão no mesmo componente.

3. A média máxima do grau, para qualquer componente C_α, sempre será $2K_\alpha$ (Figura 3.2(a)), isso ocorre quando todos os K_α vizinhos mais próximos de $v_i \in C_\alpha$ possuem a mesma classe.

4. Apenas vértices que pertencem à mesma classe podem ser conectados. Portanto, cada componente é associado a uma única classe e o número de componentes é no

máximo igual ao número de classes ($R \geq \mu$).

5. O número de arestas entre vértices de um mesmo componente C_α, é proporcional a K_α e, pode ser no máximo igual a $K_\alpha N_\alpha$, sendo N_α o número de vértices no componente. Note que a soma do grau conta as arestas duas vezes e corresponde à $2K_\alpha N_\alpha$.

O Algoritmo 3.1 detalha a construção do grafo, K-associado, tendo como entrada, o tamanho da vizinhança considerada K, e o conjunto de dados com classe associada. O algoritmo retorna o grafo K-associado, para o valor de K passado como parâmetro, como uma lista de componentes com suas respectivas purezas.

Algoritmo 3.1 Algoritmo para a construção de um grafo K-associado (Kac)

Entrada: K e $X = \{(\mathbf{x}_i, y_i), ..., (\mathbf{x}_N, y_N)\}$ {Conjunto de dados com classe associada}
Saída: $G^{(K)} = \{C_1, \ldots, C_N\}$ onde $C_\alpha = (G'_K(V', E'); \Phi_\alpha)$ {Grafo K-associado}
  $C \Leftarrow \emptyset$
  $G^{(K)} \Leftarrow \emptyset$
  para todo $v_i \in V$ **faça**
    para $j = 1$ to K **faça**
      $\Delta_{v_i,K} \Leftarrow \{v_j | v_j \in \Lambda_{v_i,K}$ **e** $y_j = y_i\}$
      $E \Leftarrow E \cup \{e_{ij} | v_j \in \Delta_{v_i,K}\}$
    fim para
  fim para
  $C \Leftarrow encontraComponentes(V,E)$
  para todo $C_\alpha \in C$ **faça**
    $\Phi_\alpha \Leftarrow pureza(C_\alpha)$
    $G^{(K)} \Leftarrow G^{(K)} \cup \{(C_\alpha(V', E'); \Phi_\alpha)\}$
  fim para
  retorne $G^{(K)}$

Como mencionado, o vértice v_i representa a instância $\mathbf{x}_i$. O algoritmo define o conjunto $\Delta_{v_i,K}$ que representa os vértices pertencentes à K-vizinhança do vértice v_i que também pertencem à mesma classe que v_i. Estes conjuntos serão usados para conectar o grafo, como mostrado no Algoritmo 3.1. Uma vez construído, a função $encontraComponentes()$ encontra os componentes no grafo K-associado atual, por meio da busca em largura ou em profundidade. O método $pureza()$ calcula a medida de pureza para cada componente, como será detalhada na próxima seção.

Note que cada valor de K está associado a um grafo K-associado e, como ficará claro mais adiante, diferentes valores para o parâmetro K resultará em grafos com diferentes características como, números de componentes e seus valores de pureza. Entretanto, no contexto de classificação, é desejável que cada componente apresente o mínimo de ruído, ou pureza máxima. A ideia é encontrar um grafo que consista em vários componentes sendo que cada componente mantenha a maior pureza possível, sem que dependam de um único valor de K.

3.1.1 A medida de pureza

A maneira em que o grafo K-associado é construído permite o cálculo de uma medida que reflete o quão puro ou livre de ruído é um componente. O cálculo da pureza utiliza-se da topologia do grafo para quantificar o grau de mistura em relação às diferentes classes.

Seja g_i o grau do vértice v_i, N o número de instâncias no conjunto de treinamento e K o tamanho da vizinhança considerada na construção do grafo. A taxa $g_i/2K$ corresponde à fração de arestas entre o vértice v_i e os vértices do mesmo componente. Assim, o total de conexões $|E_\alpha|$ entre vértices no componente C_α é dado pela Equação (3.1).

$$|E_\alpha| = \frac{1}{2}\sum_{i=1}^{N_\alpha} g_i = \frac{N_\alpha}{2}\sum_{i=1}^{N_\alpha}\frac{g_i}{N_\alpha} = \frac{N_\alpha}{2}\langle G_\alpha\rangle \tag{3.1}$$

Na qual, N_α é o número de vértices no componente C_α e $\langle G_\alpha\rangle$ corresponde à média do grau no componente C_α. O número máximo de arestas entre os N_α vértices do componente C_α é KN_α. Dessa forma, pode-se definir uma medida de pureza ou de quão misturado estão os componentes com relação às suas classes. Considere a medida de pureza definida pela Equação (3.2).

$$\Phi_\alpha = \frac{\frac{N_\alpha\langle G_\alpha\rangle}{2}}{KN_\alpha} = \frac{\langle G_\alpha\rangle}{2K} \tag{3.2}$$

Na equação, Φ_α é definida como a *pureza* do componente C_α. A Figura 3.2 ilustra alguns exemplos do cálculo da pureza.

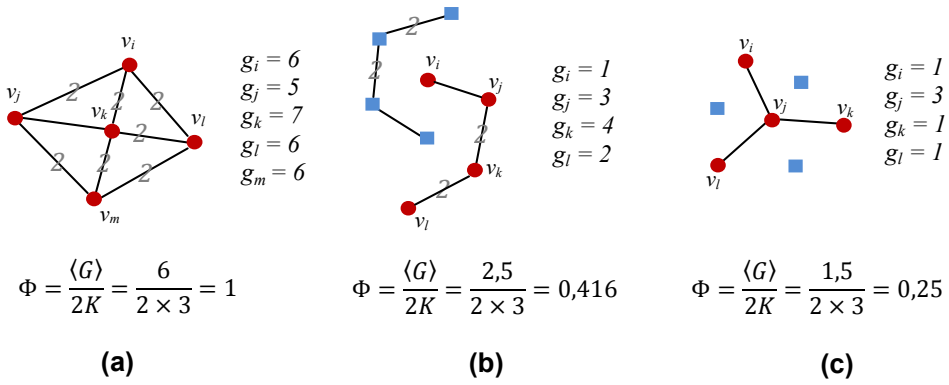

Figura 3.2: Exemplo do cálculo da pureza para $K = 3$, considerando (a) componente puro, (b) componentes com pureza intermediária e (c) componente em volta a ruído.

Para um componente relativamente isolado com N instâncias de dados pertencentes a uma mesma classe, como na Figura 3.2(a), todo vértice terá conexões com todos os K outros vértices (formando um clique), o que resulta em um total de KN arestas ou $2KN$ na soma do grau. De modo que, o grau médio neste caso é $\langle G\rangle = 2K$, resultando no valor máximo para a pureza, $\Phi_\alpha = 1$. Em contrapartida, quando existem ruídos ou quando vértices de classes diferentes encontram-se misturados, os vértices não estabelecerão K conexões devido à presença de vértices de diferentes classes em suas K-vizinhanças. Nesses casos, quanto maior é a mistura dos dados, menor será o valor da pureza, como na Figura

3.2(b), onde os componentes encontram-se entrelaçados diminuindo a pureza um do outro (neste caso ambos têm pureza identica). O caso extremo, $\Phi_\alpha = 0$, refere-se a vértices isolados no grafo que estão relacionados a dados ruidosos ou à *outliers* (Hodge e Austin, 2004), como por exemplo, os elementos da classe representada por retângulos azuis na Figura 3.2(c). Note que, a pureza igual a zero desconsidera o componente para ser usado na classificação de novos dados - o que significa que, por meio da pureza é possível detectar ruídos e *outliers*. De modo geral, Φ_α é definida no intervalo $[0, 1]$, no qual, valores próximos a 1 indicam maior interconectividade em um componente enquanto que valores menores indicam mistura entre componentes de classes diferentes ou ruído. Portanto, a relação $\Phi_\alpha = \langle G_\alpha \rangle / 2K$ pode ser considerada como uma medida de pureza para o componente C_α. Em última análise, a pureza pode ser vista como a probabilidade *a priori* de conexão na região delimitada pelo componente. Esta propriedade será explorada pelo classificador para definir as classes dos novos dados.

3.1.2 O grafo K-associado ótimo

No processo descrito para a construção de um grafo K-associado, note que diferentes grafos podem ser gerados usando diferentes valores de K, e que, de acordo com a pureza, alguns dos grafos podem conter melhores componentes do que outros. No geral, raramente um grafo obtido com um único valor de K possuirá os melhores componentes dentre todos os possíveis componentes gerados, considerando os diversos valores possíveis para K. O que se observa é que, diferentes regiões do espaço de dados são melhor representadas por diferentes componentes - gerados com diferentes valores de K. Análogo à classificação realizada pelo algoritmo KNN, onde o valor de K que resulta em melhor desempenho de classificação varia de acordo com o domínio de dados. Consequentemente, uma ideia intuitiva é obter um grafo com a melhor organização dos vértices em componentes e maximizar a pureza sem depender de um único valor de K. Dessa forma, o componente C_α tem seu próprio K_α, que é o valor de K para o qual a pureza do componente é máxima.

Portanto, para obter o grafo ótimo, aumenta-se o valor de K enquanto os melhores componentes, até então encontrados, são armazenados. Para comparar componentes obtidos com valores diferentes de K, considere os grafos, K-associado e $(K+q)$-associado, para qualquer inteiro $q > 1$. Sejam α e β índices de componentes, um componente $C_\beta^{(K+q)}$ pertencente ao grafo $(K+q)$-associado, com pureza $\Phi_\beta^{(K+q)}$, deve ser comparado, em relação à pureza, a todo componente C_α, cuja pureza é $\Phi_\alpha^{(K)}$, de forma que, para que o componente $C_\beta^{(K+q)}$ seja considerado mais puro que os componentes $C_\alpha^{(K)} \subseteq C_\beta^{(K+q)}$, a Equação (3.3) deve ser satisfeita.

$$\Phi_\beta^{(K+q)} \geq \Phi_\alpha^{(K)} \text{ para todo } C_\alpha^{(K)} \subseteq C_\beta^{(K+q)} \tag{3.3}$$

O grafo ótimo é a estrutura final e única obtida por meio do aumento de K e pela seleção dos melhores componentes ao longo de diferentes valores de K, de acordo com a

Equação (3.3). No Algoritmo 3.2 é detalhado a construção do grafo K-associado ótimo tendo como entrada um conjunto de treinamento. Note que este usa o Algoritmo 3.1 para a construção dos grafos K-associados. O grafo ótimo será usado na classificação de novos dados, como será exposto na próxima seção.

Algoritmo 3.2 Algoritmo da construção do grafo K-associado ótimo (KAOG)

Entrada: $X = \{(\mathbf{x}_i, y_i), ..., (\mathbf{x}_N, y_N)\}$ {Conjunto de dados vetoriais com classe associada}
Saída: $G^{(ot)} = \{(G'(V, E); \Phi)\}$; {Grafo com os melhores componentes}

$K \Leftarrow 1$
$G^{(ot)} \Leftarrow Kac(X, K)$
repita
  $ultimaTaxa \Leftarrow G^{(K)}/K$
  $K \Leftarrow K + 1$
  $G^{(K)} \Leftarrow Kac(X, K)$
  para todo $C_\beta^{(K)} \subset G^{(K)}$ **faça**
    se ($\Phi_\beta^{(K)} \geq \Phi_\alpha^{(ot)}$ para todo $C_\alpha^{(ot)} \subseteq C_\beta^{(K)}$) **então**
      $$G^{(ot)} \Leftarrow G^{(ot)} - \bigcup_{C_\alpha^{(ot)} \subseteq C_\beta^{(K)}} C_\alpha^{(ot)}$$
      $$G^{(ot)} \Leftarrow G^{(ot)} \cup \{C_\beta^{(K)}\}$$
    fim se
  fim para
até que $\langle G^{(K)} \rangle / K < ultimaTaxa$;
retorne $G^{(ot)}$

No Algoritmo 3.2, a construção do grafo ótimo é detalhada. O algoritmo tem como única entrada o conjunto de treinamento, que é representado por um conjunto de vértices, cada qual com uma classe associada. O grafo ótimo, notado por $G^{(ot)}$, pode ser visto como uma lista dos melhores componentes encontrados durante este processo. Inicialmente a lista de componentes $G^{(ot)}$ é inicializada com o grafo 1-associado ($K = 1$), pois até então todos os componentes são os melhores encontrados. O algoritmo procede incrementando K de um e, para cada K, constrói-se o grafo K-associado correspondente (por meio da função Kac, Algoritmo 3.1), representado por $G^{(K)}$ no algoritmo. Conforme K aumenta, componentes de uma mesma classe tendem a unirem-se para formar componentes maiores, dado que o aumento de K pode resultar na união de alguns componentes. Cada componente recém-formado, obtido da união de um ou mais componentes do grafo K-associado anterior, tem um valor de pureza próprio que depende da estrutura do componente atual. Para cada grafo K-associado, a pureza de cada componente novo é comparada com os componentes que o formaram. Portanto, se o novo componente tiver pureza maior então trocam-se os componentes correspondentes, armazenados em $G^{(ot)}$, pelo componente recém-formados.

A substituição de componentes ocorre para todo $K > 1$, e é realizada se a pureza do componente atual for maior do que a dos componentes, armazenados no grafo ótimo, que possuem os mesmos vértices. Seja o componente $C_\beta^{(K)}$, com pureza $\Phi_\beta^{(K)}$, um componente do grafo K-associado formado na iteração atual K. Para que este componente seja aceito e adicionado no grafo ótimo, é preciso que a pureza $\Phi_\beta^{(K)}$ seja maior ou igual à pureza de todos os componentes $C_\alpha^{(ot)}$, pertencentes ao grafo ótimo, que estejam contidos

no componente $C_\beta^{(K)}$. A propriedade de crescimento monotônico dos componentes, em resposta ao aumento no valor de K, garante que um componente do grafo K-associado contenha todos os vértices dos componentes do grafo $(K-1)$-associado que o formaram, i.e. $C_\beta^{(K-1)} \subseteq C_\beta^{(K)}$. O que significa que as conexões estabelecidas não são desfeitas ao longo do aumento de K, e que componentes do grafo K-associado são provenientes da adição de arestas que eventualmente unem componentes do grafo (*K-1*)-associado. Dessa forma, um componente do grafo K-associado irá conter todos os componentes previamente obtidos a partir dos mesmos vértices.

A ideia é que variando K e mantendo os melhores componentes encontrados durante esse processo, é possível determinar o melhor valor de K, juntamente com a pureza, do componente que representa a distribuição local dos dados. Podendo, esses valores, variarem para as diferentes regiões do espaço de dados. De forma que, cada componente C_α, além de estar associado à pureza Φ_α, também é relacionado ao valor de K em que foi formado, notado por K_α. O processo segue enquanto a taxa de crescimento $\langle G \rangle / K$ continuar crescendo, ou seja, o algoritmo termina quando a taxa de crescimento, correspondente ao grafo K-associado, for menor do que a taxa do último grafo obtido (grafo $(K-1)$-associado), pela primeira vez. Para justificar esse critério de parada, considere encontrar o valor para o grau médio de um componente do grafo K-associado, de modo que, a pureza permaneça inalterada, como mostrado na Equação (3.4). Seja $\Phi^{(K-1)}$ e $\Phi^{(K)}$ a pureza deste componente nos grafos $(K-1)$-associado e K-associado, respectivamente e, $\langle G^{(K)} \rangle$ a média do grau dos componentes do grafo K-associado.

$$
\begin{aligned}
\Phi^{(K)} = \Phi^{(K-1)} \quad &\Leftrightarrow \quad \frac{\langle G^{(K)} \rangle}{2K} = \frac{\langle G^{(K-1)} \rangle}{2(K-1)} \\
&\Leftrightarrow \quad \frac{\langle G^{(K)} \rangle}{2K} = \frac{\langle G^{(K-1)} \rangle}{2K-2} \\
&\Leftrightarrow \quad \frac{2K\langle G^{(K)} \rangle}{2K} - \frac{2\langle G^{(K)} \rangle}{2K} = \frac{2K\langle G^{(K-1)} \rangle}{2K} \\
&\Leftrightarrow \quad \langle G^{(K)} \rangle - \frac{\langle G^{(K)} \rangle}{K} = \langle G^{(K-1)} \rangle \\
&\Leftrightarrow \quad \langle G^{(K)} \rangle = \langle G^{(K-1)} \rangle + \frac{\langle G^{(K)} \rangle}{K}
\end{aligned}
\tag{3.4}
$$

Note que, para que o valor da pureza de um componente do grafo $(K-1)$-associado não sofra alterações, quando recalculada para o mesmo componente no grafo K-associado, a média do grau neste componente deve ser ao menos $\langle G^{(K)} \rangle / K$ maior que a média do grau no grafo anterior. Como o critério depende somente da média do grau, pode-se generalizar esta análise para o grafo todo e não apenas um único componente. Dessa forma, qualquer aumento na média do grau, do grafo K-associado atual, maior que a essa taxa $\langle G^{(K)} \rangle / K$, resultante do incremento de K, significa que o grafo K-associado terá algum(s) componente(s) com pureza maior do que em relação ao grafo K-associado. Caso contrário, pode-se dizer que, intuitivamente, o crescimento do grafo atingiu um ponto de

saturação, onde não é mais possível manter a taxa em crescimento e como consequência a pureza máxima para cada componente já foi atingida.

O critério de parada apresentado constitui uma heurística que utiliza a taxa de crescimento da média do grau para interromper o processo de construção do grafo ótimo assim que esta decresce. O critério de parada que considera todos os componentes possíveis corresponde à convergência do número de componentes no grafo K-associado atual, para o número de classes do problema. No entanto, para certos domínios de dados, este critério pode levar muitas iterações para ser satisfeito, ao passo que, os componentes de maior pureza são obtidos em poucas iterações. Nos experimentos considerados neste trabalho, o uso da heurística como critério de parada diminuiu o esforço computacional sem comprometer o desempenho do algoritmo.

Uma consequência da definição da pureza, Equação (3.2), é que vértices isolados tem pureza igual a zero, o que significa que esses vértices não serão considerados no processo de classificação. Vértices isolados podem ser encontrados no grafo ótimo como reflexo de exemplos ruidosos ou *outliers*. Intuitivamente, quando uma instância de dado encontra-se cercado por instâncias de classes diferente da sua na distribuição dos dados; o vértice que o representa no grafo terá dificuldade para conectar-se à vértices da mesma classe. Mesmo quando o valor de K é suficientemente grande e esse tipo de vértice conecta-se a algum componente mais longe, esta nova configuração possivelmente não será aceita, pois esta conexão provavelmente degradará a pureza do componente que o vértice isolado se conectou tardamente.

3.2 Classificação não paramétrica baseada no grafo K-associado ótimo

O objetivo desta seção é apresentar o algoritmo para a construção do classificador que utiliza o grafo K-associado ótimo como um modelo construído a partir dos dados para inferir a classe de novos dados de entrada. Uma vez que cada componente é formado por vértices (instâncias) de uma única classe, pode-se calcular a probabilidade de um novo vértice pertencer a uma determinada classe, através do cálculo da probabilidade do novo vértice pertencer a um dos componentes do grafo ótimo. Antes da apresentação do classificador, considere as seguintes notações.

Tipicamente uma instância de treinamento é representada por $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{ip}, y_i)$, sendo x_{ij} o j-ésimo atributo da i-ésima instância do conjunto de treinamento e y_i a classe associada à instância $\mathbf{x}_i$; considerando um problema com μ classes tem-se, $y_i \in \Omega = \{\omega_1, \omega_2, ..., \omega_\mu\}$. A tarefa de classificação resume-se em determinar a classe, ω, associada a um novo exemplo $\mathbf{z} = (z_1, z_2, ..., z_p)$. O vértice correspondente ao exemplo de dado $\mathbf{z}$ é denotado por v_z.

3.2.1 Descrição do método de classificação proposto

De acordo com a teoria de Bayes (Hastie *et al.*, 2009), a probilidade *a posteriori* de um novo vértice v_z pertencer ao componente C_α, dado o número de vizinhos de v_z que pertencem ao componente C_α, denotado por N_{v_z,C_α}, é definida na Equação (3.5).

$$P\left(v_z \in C_\alpha | N_{v_z,C_\alpha}\right) = \frac{P\left(N_{v_z,C_\alpha}|C_\alpha\right) P\left(C_\alpha\right)}{P\left(N_{v_z}\right)} \tag{3.5}$$

Como cada componente C_α teve origem em determinado grafo K-associado, o número de vizinhos de v_z, pertencentes ao componente C_α, N_{v_z,C_α}, deve considerar K_α como o valor de K, no qual o componente C_α foi formado. Portanto, a probabilidade do vértice v_z estabelecer N_{v_z,C_α} conexões com o componente C_α, considerando as K_α possíveis conexões, é definida na Equação (3.6).

$$P\left(N_{v_z,C_\alpha}|C_\alpha\right) = \frac{|\{\Lambda_{v_z,K_\alpha} \cap C_\alpha\}|}{K_\alpha} \tag{3.6}$$

Na equação o conjunto Λ_{v_z,K_α} representa o conjunto dos K_α vizinhos mais próximos de v_z. O termo de normalização, $P(N_{v_z})$, na Equação (3.5), pode ser determinado considerando todos os componentes, C_β, que v_z tenha conectado no processo de definição dos vizinhos mais próximos, ou seja, todos os componentes para os quais $N_{v_z,C_\beta} \neq 0$, como definido na Equação (3.7).

$$P\left(N_{v_z}\right) = \sum_{N_{v_z,C_\beta} \neq 0} P\left(N_{v_z,C_\beta}|C_\beta\right) P\left(C_\beta\right) \tag{3.7}$$

Como mencionado na Seção 3.1.1, a pureza, além de refletir o nível de mistura, pode ser vista como a probabilidade de interconexão em um componente. Dessa forma, a pureza normalizada pode agir como a probabilidade *a priori* de cada componente. De modo que, para ser usada como probabilidade, a normalização da pureza considera somente os componentes que v_z foi conectado, como mostra a Equação (3.8).

$$P\left(C_\alpha\right) = \frac{\Phi_\alpha}{\sum\limits_{N_{v_z,C_\beta} \neq 0} \Phi_\beta} \tag{3.8}$$

Na maioria dos casos o número de componentes excede o número de classes, de acordo o classificador ótimo de Bayes, como as probabilidades *a posteriori* são calculados para cada componente, é necessário combinar as probabilidades *a posteriori* de todos os componentes que pertençam à mesma classe para extrair a verdadeira probabilidade *a posteriori* do novo exemplo pertencer a uma determinada classe. Dessa forma a probabilidade *a posteriori* de um novo exemplo pertencer a uma classe será a soma das probabilidades *a posteriori* obtidas dos componentes com a mesma classe. Dessa forma, a probabilidade

de v_z pertencer à classe ω_i, é determinada como na Equação (3.9).

$$P\left(v_z|\omega_i\right) = \sum_{\hat{C}_\alpha=\omega_i} P\left(v_z \in C_\alpha|N_{v_z,C_\alpha}\right) \tag{3.9}$$

Onde $\hat{C}_\alpha$ representa a classe dos vértices no componente C_α. Finalmente o maior valor entre as probabilidades *a posteriori* (*Maximum A Posteriori*) reflete a classe mais provável para o novo exemplo, de acordo com a Equação (3.10).

$$\varphi\left(v_z\right) = arg\ max\{P\left(v_z|\omega_1\right), ..., P\left(v_z|\omega_\mu\right)\} \tag{3.10}$$

Na qual, $\varphi(v_z)$ identifica a classe atribuída ao vértice v_z e, por consequência ao exemplo $\mathbf{z}$. O classificador é recomendado para tratar problemas multiclasses ($\mu > 2$) devido à organização esparsa do grafo ótimo aliada à medida de pureza que, como visto anteriormente, pode ser vista como uma probabilidade *a priori* para o classificador Bayesiano. O Algoritmo 3.3 ilustra, em detalhes, os passos na classificação de um novo exemplo $\mathbf{z}$, representado como vértice v_z.

Algoritmo 3.3 Algoritmo do processo de classificação - classificador estático

Entrada: $X = \{(\mathbf{x}_i, y_i), ..., (\mathbf{x}_N, y_N)\}$ {Conjunto de dados com classe associada},
$G^{(ot)}$ {Grafo K-associado ótimo} e $\mathbf{z}$ {Novo exemplo a ser classificado}
Saída: $\varphi(v_z)$ {Classe atribuída ao exemplo $\mathbf{z}$}
$K_{max} \Leftarrow maiorK(G^{(ot)})$
$\Lambda_{v_z,K_{max}} \Leftarrow encontraVizinhos(X, \mathbf{z}, K_{max})$
para todo $v_i \in \Lambda_{v_z,K_{max}}$ **faça**
  $C_\beta \Leftarrow componente(G^{(ot)}, v_i)$
  se $v_i \in \Lambda_{v_z,K_\beta}$ **então**
    $N_{v_z,C_\beta} \Leftarrow N_{v_z,C_\beta} + 1$
  fim se
fim para
para todo $C_\beta \in G^{(ot)}$ **e** $N_{v_z,C_\beta} \neq 0$ **faça**
  $P(v_z \in C_\beta) \Leftarrow (N_{v_z,C_\beta}/K_\beta)P(C_\beta)$
  $P(N) \Leftarrow P(N) + P(v_z \in C_\beta)$
fim para
para $j = 1$ **até** μ **faça**
  se $\hat{C}_\beta = \omega_j$ **então**
    $P(\omega_j) \Leftarrow P(\omega_j) + P(v_z \in C_\beta)/P(N)$
  fim se
fim para
retorne $\varphi(v_z) = argmax\{P(\omega_1), \ldots, P(\omega_\mu)\}$

No algoritmo, o método $maiorK()$ recupera o maior valor de K obtidos durante a fase de construção do grafo ótimo, o método $encontraVizinho()$ encontra os K_{max} vizinhos mais próximos do exemplo que deve ser classificado $\mathbf{z}$, note que este método utiliza o conjunto de entrada para determinar os vizinhos. O conjunto dos vizinhos do vértice v_z, $\Lambda_{v_z,K_{max}}$, é usado para determinar a qual componente do grafo ótimo cada um deles pertence, por meio do método $componente()$. Uma vez definido a qual componente um determinado vértice pertence, verifica-se se v_i está na K_β-vizinhança de v_z, i.e. se v_i é um dos K_β vizinhos mais próximos de v_z. Em caso afirmativo, o componente C_β possui uma ligação válida com o novo vértice v_z. O número de conexão válidas, a cada componente

C_β, é armazenado em N_β, que será usado para estimar a probabilidade de v_z pertencer ao componente C_β, notada por $P(v_z \in C_\beta)$. A probabilidade $P(v_z \in C_\beta)$ depende do valor de K_β e da pureza normalizada do componente C_β, notada por $P(C_\beta)$; $P(N)$ é o termo normalizador e pode ser calculado como a soma das pertinências do novo exemplo aos diversos componentes. Uma vez obtida as pertinências por componentes, somam-se as de mesma classe, como mostrado na Equação (3.9). A classe atribuída ao novo dado **z** corresponde à classe com maior probabilidade *a posteriori*.

3.2.2 Exemplo de classificação

Com o objetivo de deixar claro o funcionamento do classificador considere o seguinte exemplo de classificação em um domínio artificial. A Figura 3.3 ilustra um grafo ótimo composto por cinco componentes (C_1, C_2, C_3, C_4 e C_5), no qual C_1 pertence à classe ω_1, C_2 e C_3 pertencem à classe ω_2, enquanto que C_4 e C_5 pertencem à classe ω_3. Na figura também são mostrados a pureza (Φ) de cada componente e o valor de K em que cada componente foi construído. O objetivo neste exemplo é ilustrar todos os passos na classificação de um novo exemplo de dado **z**, para isso o primeiro passo é inseri-lo no grafo como um novo vértice v_z. Uma vez abstraído para vértice, é necessário encontrar os vizinhos mais próximos. Como cada componente possui diferentes valores de K, é preciso determinar os K vizinhos mais próximos para cada componente. O que equivale a encontrar o número de vizinhos determinado pelo valor do maior K ($K = 4$ neste exemplo). Note na Figura 3.3 que o vértice a ser classificado v_z possui conexões (em linha pontilhada) com os quatro vértices mais próximos, numeradas de acordo com a ordem de similaridade. Na verdade estas conexões podem ser vistas como conexões imaginárias, pois somente servirão para classificar o novo vértice, uma vez que o novo vértice não integrará a rede - considerando o classificador estático. Considerando o componente C_1, com $K = 4$, todas as quatro ligações são consideradas, já no caso onde $K = 3$, como ocorre com os componentes C_2 e C_5, apenas as ligações 1, 2 e 3 são consideradas, enquanto que a ligação 4 é ignorada. O mesmo aplica-se ao caso $K = 2$, que ocorre nos componentes C_3 e C_4, apenas as ligações 1 e 2 são levadas em consideração.

Como já mencionado a tarefa é predizer a classe do novo vértice v_z (representado como estrela na figura). De acordo com a Equação (3.10), a classe do novo vértice é determinada por meio do cálculo da probabilidade *a posteriori* máxima de cada classe. Dessa forma, é necessário calcular a probabilidade *a posteriori* do novo vértice pertencer a cada um dos componentes, segundo a Equação (3.5), e depois combiná-las de acordo com cada classe como na Equação (3.9). Portanto, inicialmente deve-se determinar $P(v_z|\omega_1)$, $P(v_z|\omega_2)$, e $P(v_z|\omega_3)$.

No exemplo, a pureza de cada componente já foi calculada, no entanto para que a mesma possa se comportar como probabilidade *a priori* de cada componente, a soma entre as purezas de todos os componentes utilizados na classificação deve ser 1. Considere a normalização da pureza, como definida na Equação (3.8), feita para este estudo de caso.

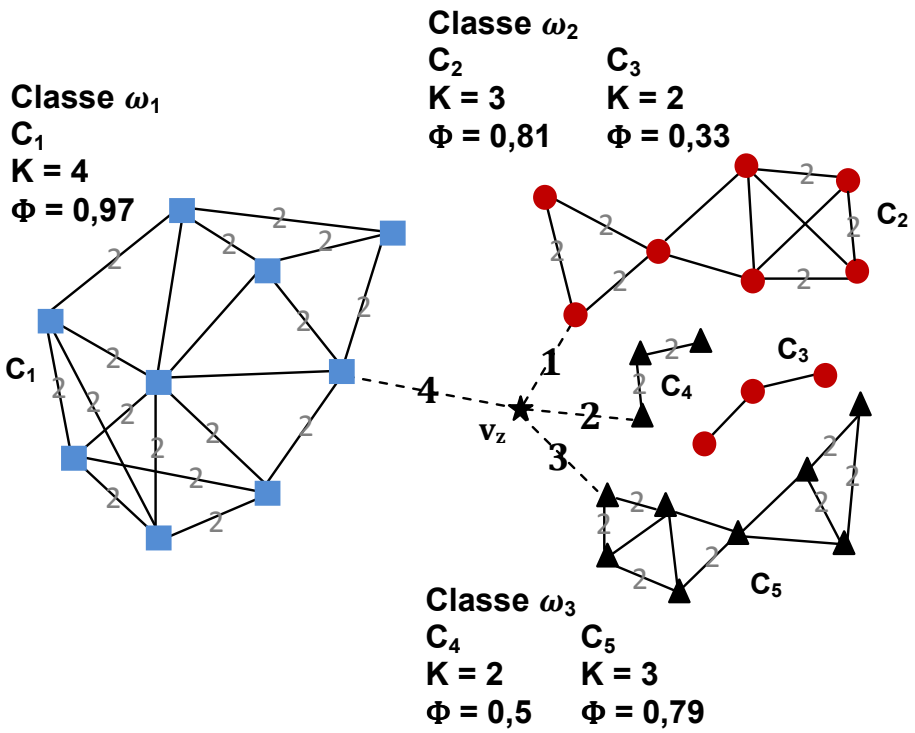

Figura 3.3: Exemplo de classificação do algoritmo KAOG em um domínio artificial com três classes.

$$P(C_1) = \frac{\Phi_1}{\Phi_1 + \Phi_2 + \Phi_4 + \Phi_5} = \frac{0,97}{0,97 + 0,81 + 0,5 + 0,79} \approx 0,32 \tag{3.11}$$

$$P(C_2) = \frac{\Phi_2}{\Phi_1 + \Phi_2 + \Phi_4 + \Phi_5} = \frac{0,81}{0,97 + 0,81 + 0,5 + 0,79} \approx 0,26 \tag{3.12}$$

$$P(C_4) = \frac{\Phi_4}{\Phi_1 + \Phi_2 + \Phi_4 + \Phi_5} = \frac{0,5}{0,97 + 0,81 + 0,5 + 0,79} \approx 0,17 \tag{3.13}$$

$$P(C_5) = \frac{\Phi_5}{\Phi_1 + \Phi_2 + \Phi_4 + \Phi_5} = \frac{0,79}{0,97 + 0,81 + 0,5 + 0,79} \approx 0,25 \tag{3.14}$$

O próximo passo é determinar a probabilidade de v_z conectar-se ao componente C_α dado K_α, denotada por $P(N_{v_z,C_\alpha}|C_\alpha)$, e definida na Equação (3.6), para $\alpha = \{1, 2, 3, 4, 5\}$. Para o componente C_1 com $K_1 = 4$, como na Figura 3.3, apenas a conexão 4 de fato conecta o novo vértice v_z ao componente C_1. Isto significa que a probabilidade de v_z ser conectado ao componente C_1, dado o número de ligações permitidas $K_1 = 4$, é de $1/4$, dessa forma tem-se $P(N_{v_z,C_1}|C_1) = 1/4$. Considere agora o cálculo para o componente C_2 que apresenta $K_2 = 3$, dessa vez apenas a ligação 1 conecta o novo vértice v_z ao componente em questão C_2, o que resulta em $P(N_{v_z,C_2}|C_2) = 1/3$. Considerando o componente C_3, com $K_3 = 2$, note que dentre os dois vizinhos mais próximos de v_z, nenhum pertence ao componente C_3, portanto $P(N_{v_z,C_3}|C_3) = 0$. Já o componente C_4, com $K_4 = 2$, recebe a conexão 2 como única conexão, dentre as duas possíveis, então $P(N_{v_z,C_4}|C_4) = 1/2$. Por último, um dos vértices do componente C_5, que permite três ligações, é o terceiro vizinho mais próximo de v_z, então $P(N_{v_z,C_5}|C_5) = 1/3$.

Finalmente, calcula-se o termo de normalização, $P\left(N_{v_z}\right)$, como definido na Equação (3.7), que corresponde à observação completa das conexões de v_z na classificação corrente. Na Figura 3.3, nota-se que as conexões de v_z utilizadas na classificação foram: uma com o componente C_1, uma com C_2, uma com C_4 e uma com C_5. Dessa forma, $P(N_{v_z,C_1}|C_1) = 1/4$, $P(N_{v_z,C_2}|C_2) = 1/3$, $P(N_{v_z,C_3}|C_3) = 0$, $P(N_{v_z,C_4}|C_4) = 1/2$ e, $P(N_{v_z,C_5}|C_5) = 1/3$. Assim, o termo normalizador é dado pela somatória das probabilidades de pertencer a todos os componentes envolvidos, como mostra a Equação (3.15).

$$\begin{aligned} P\left(N_{v_z}\right) &= \sum_{N_{v_z,C_\beta}\neq 0} P\left(N_{v_z,C_\beta}|C_\beta\right) P\left(C_\beta\right) \\ &= \frac{1}{4}\times 0,32 + \frac{1}{3}\times 0,26 + \frac{1}{3}\times 0,25 \\ &+ \frac{1}{2}\times 0,17 = 0,3343 \end{aligned} \tag{3.15}$$

Juntando os termos previamente calculados na Equação (3.5), obtém-se:

$$P\left(v_z \in C_1|N_{v_z,C_1}\right) = \frac{\frac{1}{4}\times 0,32}{0,3343} \approx 0,24 \tag{3.16}$$

$$P\left(v_z \in C_2|N_{v_z,C_2}\right) = \frac{\frac{1}{3}\times 0,26}{0,3343} \approx 0,26 \tag{3.17}$$

$$P\left(v_z \in C_4|N_{v_z,C_4}\right) = \frac{\frac{1}{2}\times 0,17}{0,3343} \approx 0,25 \tag{3.18}$$

$$P\left(v_z \in C_5|N_{v_z,C_5}\right) = \frac{\frac{1}{3}\times 0,25}{0,3343} \approx 0,25 \tag{3.19}$$

Uma vez encontradas as probabilidades de pertinência de v_z a cada um dos componentes, o passo seguinte é somar as probabilidades referentes aos diferentes componentes de uma mesma classe, como na Equação (3.9), assim tem-se:

$$P\left(v_z|\omega_1\right) = P\left(v_z \in C_1|N_{v_z,C_1}\right) = 0,24 \tag{3.20}$$

$$P\left(v_z|\omega_2\right) = P\left(v_z \in C_2|N_{v_z,C_2}\right) = 0,26 \tag{3.21}$$

$$\begin{aligned} P\left(v_z|\omega_3\right) &= P\left(v_z \in C_4|N_{v_z,C_4}\right) \\ &+ P\left(v_z \in C_5|N_{v_z,C_5}\right) = 0,5 \end{aligned} \tag{3.22}$$

Por fim, aplica-se a Equação (3.10) que corresponde em encontrar a máxima probabilidade *a posteriori* e como resultado verifica-se que v_z é classificado como pertencente à classe ω_3.

3.2.3 Superfície de decisão

Como o algoritmo proposto utiliza relações de vizinhança na classificação de novos dados, o próximo exemplo explora as diferenças entre os algoritmos KAOG e KNN. Considere o conjunto com três classes gerado artificialmente e apresentado na Figura 3.4(a). As superfícies de decisão relativa ao conjunto de dados apresentado na Figura 3.4(a) são apresentadas na Figura 3.4(b) para 1NN, Figura 3.4(c) para 15NN e na Figura 3.4(d) para o algoritmos proposto KAOG. Apesar das limitações em revelar todas as características dos algoritmos, considerando uma representação em duas dimensões com poucos dados, ainda assim é possível notar algumas das maiores diferenças entre os algoritmos.

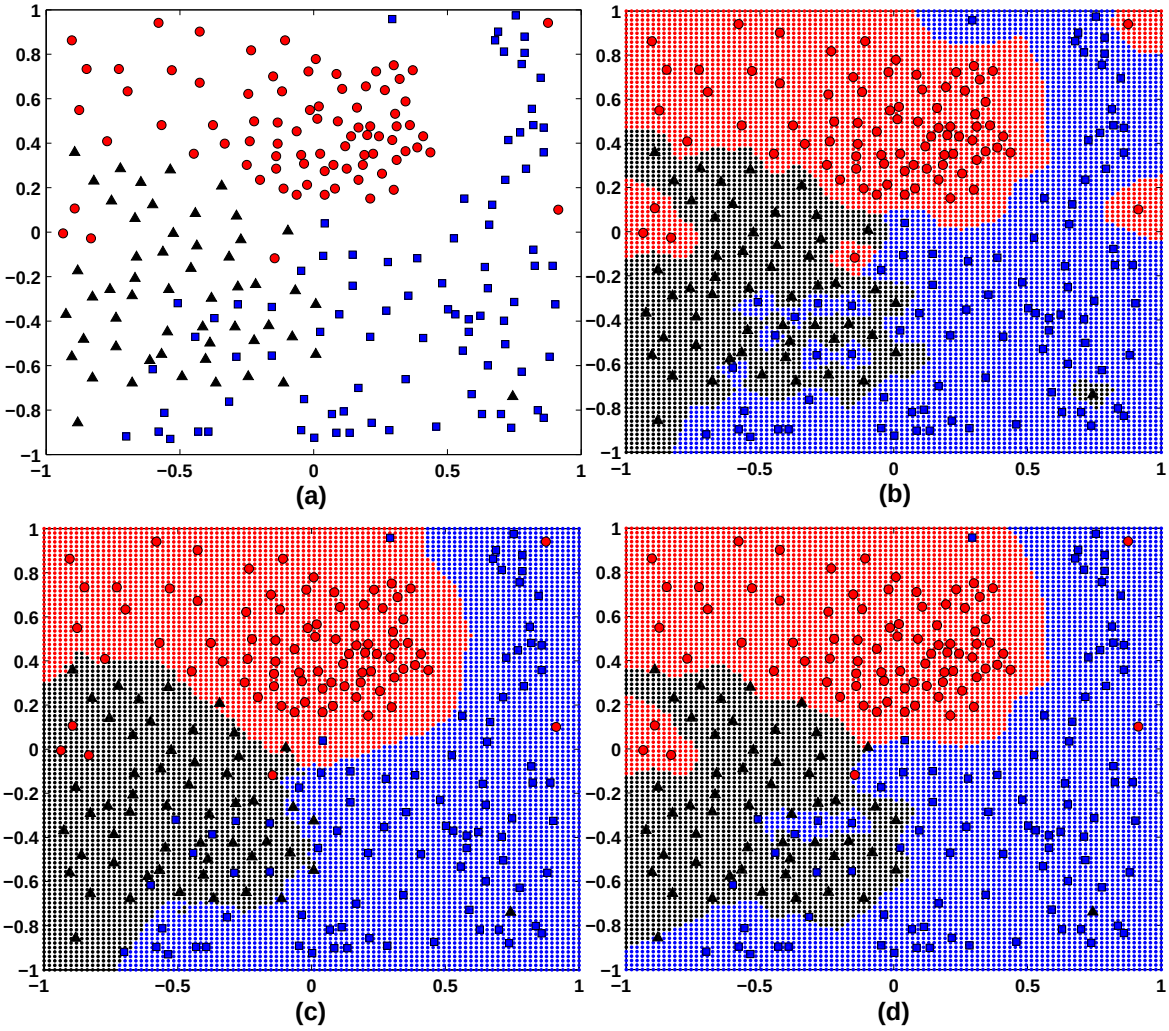

Figura 3.4: Comparação entre superficies de decisão entre KNN e KAOG. A Figura (a) representa um conjunto com três classes gerado artificialmente; nas demais figuras é apresentado a superficie de decisão gerada pelos algoritmos (b) 1-NN, (c) 15-NN e, (d) KAOG.

Como pode ser visto nas Figuras 3.4(b)-(d), métodos baseados em vizinhos mais próximos são por natureza não linear e multiclasse. Além de que, o algoritmo KAOG é capaz de ajustar localmente à distribuição dos dados, devido à maneira que os componentes são usados na classificação. Como resultado do processo de treinamento, o grafo ótimo é composto pelos componentes de maior valor de pureza, dentre os diversos componentes

obtidos na geração dos grafos K-associados. Dessa forma, o grafo ótimo pode ser visto como uma estrutura otimizada localmente que permite ao classificador agir de acordo com a distribuição local das instâncias. Uma analogia direta entre o algoritmo KAOG e o algoritmo KNN pode ser estabelecida por meio da comparação de suas respectivas superfícies de decisão. No algoritmo KNN, pequenos valores de K produzem superfícies de decisão bastante ajustada aos exemplos de dados, ao passo que, valores altos para K produzem superfícies de decisão mais gerais. Como o algoritmo proposto pode armazenar componentes obtidos através de diferentes valores de K, é consequência que, a superfície de decisão apresentará localmente a característica de certo componente. Por exemplo, em áreas com ruído o algoritmo tentará encontrar pequenos grupos que possam formar um componente, evitando a criação de vários componentes de um único vértice - o que seria semelhante à classificação feita pelo algoritmo 1NN. Enquanto que em áreas relativamente puras, o algoritmo tentará incrementar K com o objetivo de aumentar a influência desta área na classificação. Como já mencionado, o algoritmo proposto, KAOG, também possui mecanismos para a detecção e remoção de *outliers*, que consiste em remover do grafo ótimo os vértices isolados. A razão para que essas instâncias permaneçam desconectadas deve-se a sua localização em relação a outras instâncias no espaço de dados, possivelmente envoltas a instâncias de outra classe ou longe da distribuição dos dados.

3.3 Complexidade computacional

De acordo com o Algoritmo 3.2, o treinamento do classificador consiste na construção de vários grafos K-associados sequencialmente, incrementando K de um, até o ponto em que a taxa $\langle G \rangle / K$ do grafo K-associado for menor que a do grafo $(K-1)$-associado. Portanto, o critério de parada depende do conjunto de treinamento, seu nível de ruído, número de classes e distribuição das classes. Assim, para determinar a complexidade do algoritmo, considere K_{max} como o valor máximo de K quando o critério de parada fora atingido. Note que K_{max} não se configura como um parâmetro, trata-se apenas de um valor estimado do valor de parada usado para determinar a complexidade computacional do algoritmo proposto. No geral, trata-se de um valor pequeno, por exemplo, em todos os experimentos realizados neste trabalho $K_{max} < 20$. De acordo com o algoritmo KAOG, a construção do grafo, bem como a do classificador, consiste nos seguintes passos:

1. Calcular a matriz de distância a partir do conjunto de treinamento;
2. Encontrar K_{max} vizinhos mais próximos para o exemplo a ser classificado;
3. Para $K = 1$ até K_{max}
   (a) Construir o grafo K-associado correspondente;
   (b) Encontrar todos os componentes no grafo K-associado atual;
   (c) Calcular a pureza para todos os componentes;

(d) Unir componentes, referentes aos grafos (K-1)-associados e ao grafo K-associado como mostrado na Equação (3.3).

Dessa forma, a ordem de complexidade dos passos mencionados corresponde à complexidade de construção do classificador. A complexidade para classificar um conjunto com M novos exemplos usando o classificador será analisado separadamente. Apesar de muitos dos passos poderem ser otimizados, a analise é conduzida considerando o uso de métodos básicos. Ao final desta seção, serão discutidas algumas abordagens e alguns métodos avançados que podem ser usados para melhorar a performance do algoritmo proposto.

Primeiramente, considere a análise da ordem de complexidade relativa à fase de construção do classificador. Dado um conjunto de treinamento no formato *atributo-valor* com N instâncias e p atributos, a complexidade para calcular a matriz de distância correspondente é $N(N-1)p$ que implica em uma ordem de complexidade de $O(N^2p)$. Para a construção do grafo ótimo, é necessário determinar até K_{max} vizinhos mais próximos para cada uma das N instâncias de entrada. Considerando o uso de um simples algoritmo de ordenação como a ordenação por seleção (*selection sort*) (Cormen *et al.*, 2009) e o fato de que $K_{max} << N$, pode-se dizer que são requeridas N operações para encontrar um único vizinho, consequentemente são necessárias $K_{max}N$ operações para encontrar K_{max} vizinhos. Sendo assim, a ordem de complexidade para encontrar K_{max} vizinhos mais próximos para as N instâncias é $K_{max}N^2$. O passo seguinte consiste em construir cada um dos grafos K-associados. São necessárias KN operações, pois existem K tentativas de conexões para cada um dos N vértices (Algoritmo 2). Assumindo o uso de busca em largura para encontrar todos os componentes (função *encontraComponentes*() no Algoritmo 2), o algoritmo tem ordem de complexidade de $O(|V| + |E|)$, no qual o número de vértices é igual ao número de instâncias no conjunto de treinamento $|V| = N$, no entanto o número de arestas $|E|$ aumenta de acordo com K. Uma vez que $K_{max} << N$, tem-se que o grafo K-associado será esparso, ou seja $|E| = O(N)$, dessa forma a ordem de complexidade para encontrar todos os componentes em um grafo K-associado é $O(N)$. Uma vez encontrados os componentes, a medida de pureza pode ser calculada simplesmente através da soma do grau dos N vértices, o que tem ordem de complexidade de $O(N)$. Prosseguindo com a formação da rede, o próximo passo é decidir se componente atual - possivelmente resultado de união entre dois ou mais componentes do componentes do grafo (K-1)-associado, é preferível ao(s) componente(s) anterior(es). Para isso é necessário comparar a pureza do componente atual (da rede K-associado) com os componentes do último grafo ($(K-1)$-associado) que o formaram. Sendo assim a ordem de complexidade é menor ou igual a $O(N)$ - pois N é o número máximo de componentes. Para a construção do grafo ótimo varia-se os valores de K, de 1 a K_{max}, portanto, a complexidade do passo 3 é $K_{max}N$. Juntando os três passos, a ordem de complexidade é $O(N^2p + K_{max}N^2 + K_{max}N)$. Considere $p << N$ e $K_{max} << N$, a ordem de complexidade para a construção do classificador é $O(N^2)$.

Com o objetivo de verificar a ordem de complexidade previamente estabelecida do algoritmo KAOG, considere o seguinte experimento de tempo de execução apresentado na Figura 3.5. No experimento foram usados conjuntos de dados com diferentes tamanhos, de 100 a 10.000 instâncias. Cada resultado representa o tempo (em segundos) gasto para construir o grafo ótimo a partir de um conjunto de dados de entrada no formato vetorial. Os resultados consistem na média de 100 execuções, e também são apresentados os valores extremos (mínimo e máximo) dentre as 100 para cada conjunto. Os conjuntos de dados foram gerados artificialmente para toda execução e consistem em dois atributos cujos valores pertencem ao intervalo $[0, 1]$ e com classe atribuída aleatoriamente considerando os possíveis valores $\{1, 2\}$. O experimento foi conduzido usando Java em um ambiente Windows, com um equipamento de configuração Core 2 Duo 1.83 Ghz com 2 Gb de memória.

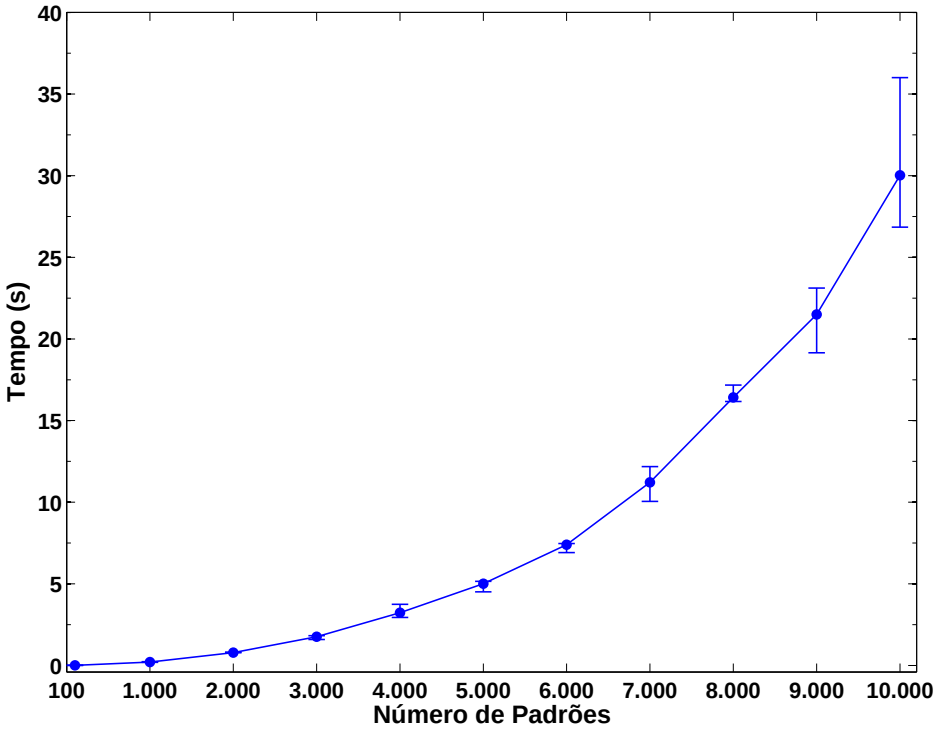

Figura 3.5: Tempo médio de execução seguido dos valores extremos na construção do grafo ótimo, a partir de conjuntos de dados com diferentes tamanhos.

Considere, agora, a análise da ordem de complexidade referente ao processo de classificação. Considerando M exemplos de teste, o cálculo da matriz de distância entre o conjunto de treino e o de teste requer MNp operações. Encontrar os K_{max} vizinhos mais próximos necessita de $K_{max}MN$ operações. Dessa forma, tem-se $O(MNp + K_{max}MN)$. Assumindo que $M << N$, $p << N$ e $K_{max} << N$, a ordem de complexidade para classificar M exemplos, é de $O(MN)$. Se $M << N$, a ordem para classificação é de $O(N)$. Como caso especial, a complexidade para classificar um novo dado é de $O(N)$.

Repare que o processo mais caro computacionalmente é o cálculo da distância entre o novo exemplo e todas as instâncias de treinamento, assim como nos métodos baseados em vizinhos mais próximos, dessa forma qualquer variação que possa ser implementado por

um método de vizinhos mais próximos com o objetivo de reduzir o tempo de execução, também pode ser implementado pelo algoritmo KAOG. Por exemplo, o uso de métodos baseados em árvores possibilita encontrar os vizinhos mais próximos sem ter que calcular explicitamente a distância para todas as instâncias (Hernandez-Rodriguez *et al.*, 2010), ou de métodos baseados em estimativas probabilísticas (Toyama *et al.*, 2010). Outro trabalho propõe um método baseado na estratégia de dividir para conquistar, e utiliza a Bissecção de Lanczos (Chen *et al.*, 2009b) para construir um grafo KNN com limite superior para a complexidade da ordem de $O(Np)$. Considerando este algoritmo na construção de cada grafo K-associado, obtém-se $O(K^2Np + Nc) = O(N)$. Considerando essas melhorias não apenas tempo computacional pode ser economizado, mas também o uso de memória pode ser otimizado no algoritmo KAOG por meio do uso de técnicas de redução baseadas em instâncias (Wilson e Martinez, 2000).

A principal vantagem do algoritmo KAOG está no fato de não necessitar de seleção de modelo, o que, em muitos casos é computacionalmente mais caro que o próprio algoritmo de aprendizado. Por exemplo, uma execução do método de validação cruzada com repetição para P-partições (ver Seção 2.3.2), demanda P vezes D vezes N_P, onde D é o número de modelos considerados e N_P o número de repetições consideradas. Portanto, considerando $N_P = P$, e.g. 10 repetições da validação cruzada de 10-conjuntos; a seleção de modelo, neste caso requer P^2D operações, ou seja tem ordem de complexidade $O(P^2)$, sem contar a complexidade de treino de cada modelo. Comparado ao algoritmo KAOG, que não exige seleção de modelo, portanto, complexidade de $O(1)$ para a fase de seleção for modelos, o que implica em uma grande vantagem no ajuste do classificador.

3.4 Exemplos ilustrativos e resultados experimentais

Nesta seção, alguns exemplos ilustrativos são apresentados com o objetivo de melhor explicar os conceitos introduzidos, tais como, a medida de pureza, o grafo K-associado, a construção do grafo ótimo e as diferenças entre os grafos K-associado e K-associado ótimo. Também é apresentado os resultado de classificação em domínios do repositório UCI (Asuncion e Newman, 2007) para o algoritmo proposto, bem como, uma análise comparativa considerando cinco outros classificadores multiclasse.

3.4.1 Construindo o grafo ótimo: um exemplo ilustrativo

Com o objetivo de ilustrar a formação do grafo ótimo, considere o conjunto de dados formado a partir da base de dados Iris (Asuncion e Newman, 2007) considerando apenas os atributos 1 e 4 bem como sua classe correspondente. Os dados de mesma classe repetidos devido à seleção de somente dois atributos foram removidos, resultando em uma base de dados com 114 instâncias de dados. As Figuras 3.6(a)-(d) mostram os grafos 1-associado, 2-associados, 3-associados e 4-associados, respectivamente. Em cada uma das figuras, as conexões pontilhadas (de cor magenta) representam o grafo K-associado e as conexões

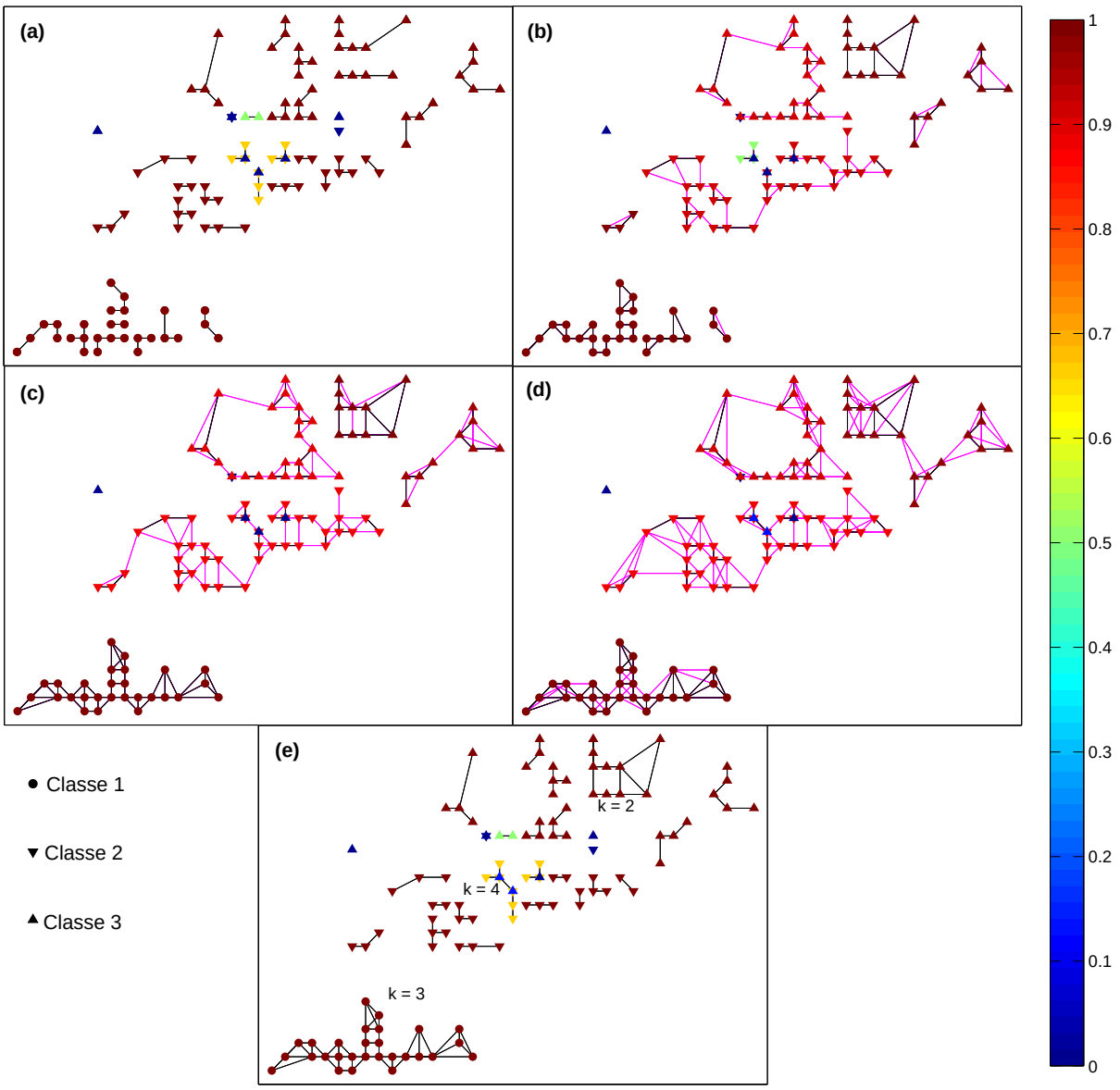

Figura 3.6: Formção do grafo ótimo para o conjunto formado com o atributos 1 e 4 da base de dados Iris. Conexões duplas entre vértices são mostradas como simples por motivos de visualização. As figuras corresponde a, (a) grafo 1-associado; (b) grafo 2-associado; (c) grafo 3-associado; (d) grafo 4-associado e (e) grafo ótimo.

contínuas (de cor preta) representam os componentes de melhor pureza que serão inseridos no grafo ótimo. A Figura 3.6(e) mostra o grafo ótimo; a barra de cor indica a pureza de cada componente.

A Figura 3.6(a) mostra o grafo 1-associado, no qual a maioria dos componentes tem pureza igual a 1, exceto os componentes que apresentam-se muito misturados com componentes de outras classes ou estão muito longe de qualquer outro exemplo da mesma classe, o que acaba resultando em um componente formado por um único vértice isolado. A Figura 3.6(b), mostra o grafo 2-associado, até então vários componentes pequenos foram unidos em componentes maiores. Observe também que os componentes que estão em uma área de sobreposição de classes têm suas purezas diminuídas em comparação à pureza desses componentes no grafo 1-associado, portanto, a união desses componentes

não é aceita. A Figura 3.6(c) representa o grafo 3-associado. Neste caso, as purezas dos componentes que estão em área de sobreposição continuam a diminuir, ao mesmo tempo em que a classe ω_1 torna-se um único componente. Na Figura 3.6(d) a adição de conexões já não aumenta a pureza do componente referente à classe ω_1. Entretanto, não é possível saber quando o grafo ótimo será obtido, por isso o processo segue até que o critério de parada seja satisfeito. Finalmente, a Figura 3.6(e) ilustra o grafo ótimo construído ao final desse processo. Note que os vértices que permaneceram isolados até o final do processo de construção do grafo ótimo (considerados componentes de único vértice), estão rodeados por vértices de outras classes e, dessa forma são considerados ruídos (*outliers*) e consequentemente deixados de fora pelo algoritmo KAOG.

Note que, de acordo com a regra descrita na Equação (3.3), a união de dois ou mais componentes será aceita, se e somente se, a pureza do novo componente for maior ou igual às purezas de cada um dos componentes. Desta forma, o grafo ótimo é, na maioria das vezes, diferente do grafo K-associado, como pode ser visto na Figura 3.6.

3.4.2 Comparações entre o grafo K-associado e o grafo K-associado ótimo

Com o objetivo de visualizar a construção do grafo K-associado e do grafo K-associado ótimo, a partir de um conjunto de dados, e para elucidar as diferenças entre eles, esta seção apresenta um exemplo artificial e extrai algumas medidas de ambos os grafos. Os experimentos foram conduzidos com o fim de mostrar algumas características importantes pertinentes a esses grafos, tais como, média do grau, número de componentes, coeficiente de agrupamento e pureza do grafo. Como a pureza é uma medida de componentes, considere a pureza do grafo como a média das purezas dos componentes ponderada pelo seu tamanho, denotada por $\langle\Phi\rangle$ e definida na Equação (3.23).

$$\langle\Phi\rangle = \frac{\sum_{s=1}^{R} \frac{N_\alpha \Phi_\alpha}{R}}{N} \tag{3.23}$$

Na equação, R é o número de componentes no grafo ótimo, N_α e Φ_α correspondem ao número de vértices e a pureza do componente C_α, respectivamente, e N é o número total de vértices no grafo. A ideia é capturar o comportamento dos grafos em um cenário em que a sobreposição entre as classes aumenta de maneira contralada. Para isso foi considerado 3 classes com distribuições Gaussiana e 100 instâncias cada uma, geradas em 10 distribuições diferentes, variando entre um cenário em que se encontram totalmente separadas para o cenário onde as três classes dividem a mesma distribuição. Sejam as três Gaussianas, $\mathcal{N}_1(\mu_1, 1)$, $\mathcal{N}_2(\mu_2, 1)$ e $\mathcal{N}_3(\mu_3, 1)$, com instâncias das classes ω_1, ω_2 e ω_3 respectivamente. Dessa forma, as 10 distribuições são obtidas variando os centros (x_1, x_2) de acordo com a Equação (3.24), onde μ^j corresponde à j-ésima distribuição.

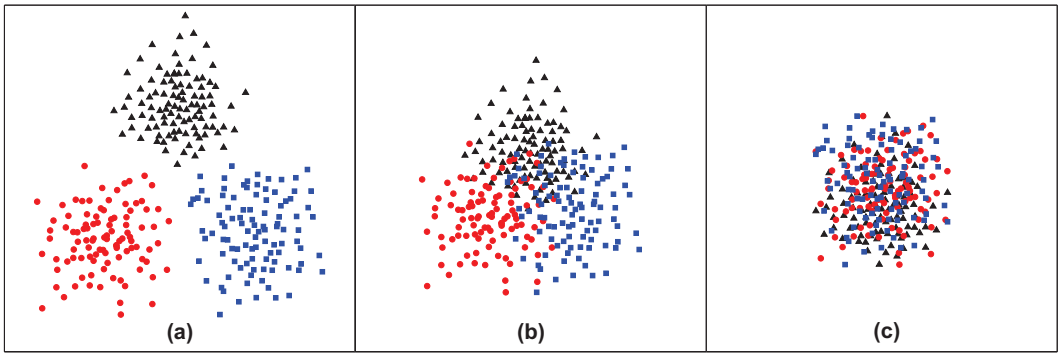

Figura 3.7: Distribuições Gaussianas com diferentes níveis de sobreposição.

$$\begin{aligned} \mu_1^j &= (9-0,5j;0+0,5j) \\ \mu_2^j &= (0+0,5j;0+0,5j) \quad \text{para} \quad j=0,\ldots,9 \\ \mu_3^j &= (4,5;9-0,5j) \end{aligned} \tag{3.24}$$

De maneira que, as três Gaussianas aproximam-se uma das outras, em dez passos, até convergirem para um centro único (4,5;4,5). A Figura 3.7 ilustra três instâncias desses conjuntos. Para cada conjunto, o grau de mistura entre as classes varia entre totalmente separadas, como na Figura 3.7(a), passando por estágios intermediários, como na Figura 3.7(b), a totalmente misturadas Figura 3.7(c).

A Figura 3.8 mostra os resultados das medidas mencionadas. Cada resultado é a média de 10 execuções para cada grafo (3-associado, 5-associado e grafo ótimo) considerando cada um dos 10 conjuntos. Na Figura 3.8 o eixo das abcissas (x) representa cada um dos conjuntos de dados com diferentes sobreposições entre classes, ordenados de forma ascendente quanto ao nível de sobreposição, ou seja, a coordenada $x = 1$ representa o conceito que apresenta as classes totalmente separadas, ao passo que a coordenada $x = 10$ representa o conceito no qual as classes dividem a mesma distribuição.

Considere o grafo K-associado, o aumento no valor de K resulta na diminuição do número de componentes e, dado um valor razoavelmente grande para K, o número de componentes converge para o número de classes. Note que quanto mais misturados estão as classes, no conjunto de dados, maior será o valor de K para que o número de componentes se iguale ao número de classes. Tendo esses fatos em mente, a pureza pode ser entendida como uma troca entre o número de vizinhos e o inverso do tamanho da vizinhança. Portanto, para manter um alto valor da pureza, um conjunto de dados com distribuição de classes altamente sobrepostas tende a formar pequenos componentes (grafo esparso localmente conectado), enquanto conjuntos que apresentam distribuição de classes totalmente disjuntas formarão grandes componentes densamente conectados. Esses fatos podem ser verificados na Figura 3.8(a), i.e., os grafos 3-associado, 5-associado e o grafo ótimo resultam em grafos mais esparsos conforme a sobreposição aumenta, elevando o número de componentes no grafo. Na Figura 3.8(b), o baixo valor da média do

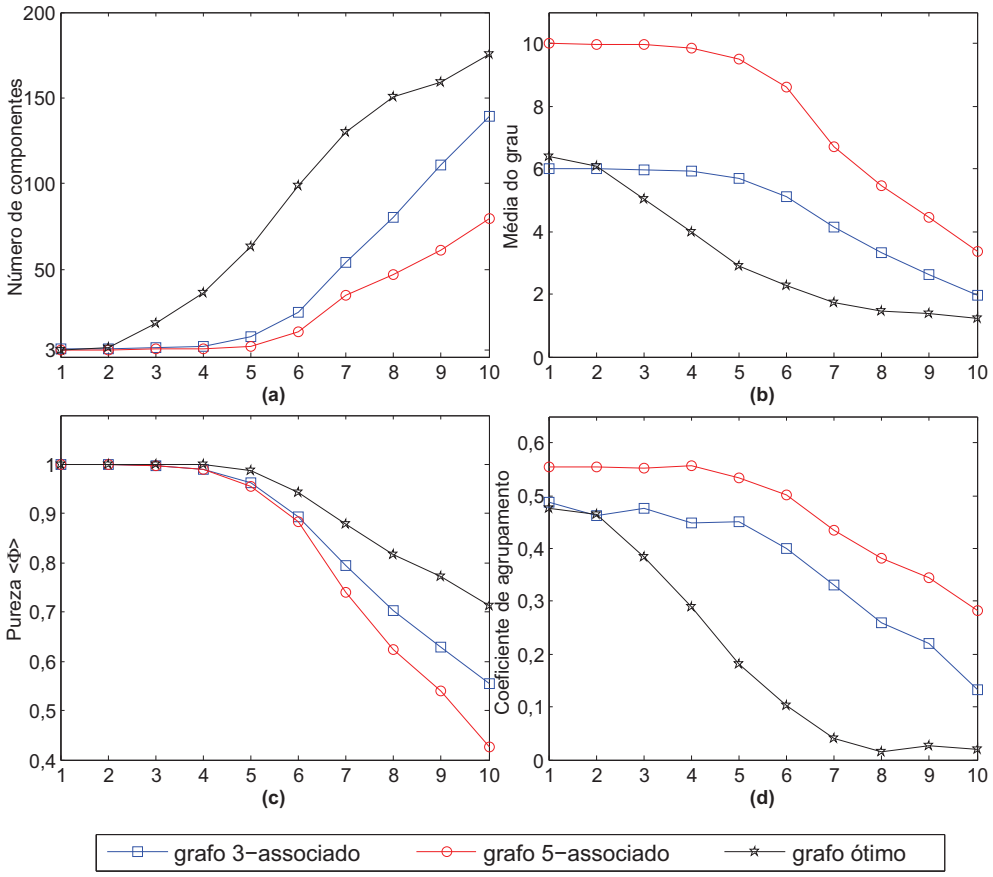

Figura 3.8: Resultados da aplicação dos grafos 3-associado, 5-associado e ótimo em 10 conjuntos de dados com diferentes graus de sobreposição entre classes.

grau evidencia a eficiência do grafo ótimo em produzir componentes conectados. Ainda com relação à Figura 3.8(b), pode-se notar que a média do grau diminui enquanto a sobreposição das classes aumenta, devido ao fato de cada vértice (instância) em um conjunto com classes misturadas tem menos vizinhos da mesma classe, enquanto que cada vértice em um conjunto de dados com pouca mistura entre classes terá mais vizinhos da mesma classe.

A Figura 3.8(c) apresenta o comportamento da medida de pureza para os grafos K-associado, bem como para o grafo ótimo, à medida que a sobreposição entre classes aumenta. Note que a pureza diminui enquanto a sobreposição entre as classes aumenta. Geralmente, a pureza pode aumentar com o crescimento de K e atingir o valor ótimo para algum K, entretanto, continuar aumentando o valor de K resultará em decréscimo da pureza. Esta característica pode ser observada na Figura 3.8(c), na qual o grafo 3-associado tem maior pureza que o grafo 5-associado. Como o grafo ótimo é construído com o objetivo de maximizar a pureza, este sempre apresentará pureza alta e baixo número de arestas, se comparado aos grafos K-associados. Para analisar a Figura 3.8(d), considere a medida de coeficiente de agrupamento definida por Watts e Strogatz (1998), como mostrada na Equação (3.25).

$$cc_i = \frac{2N_{e_i}}{N_{v_i}(N_{v_i}-1)} \quad \text{e} \quad cc_G = \frac{1}{N}\sum_i cc_i \tag{3.25}$$

Na qual, o coeficiente é medido para cada vértice, dessa forma o vértice v_i tem coeficiente de agrupamento cc_i. O coeficiente do grafo cc_G é a média dos coeficientes dos vértices. Na equação, N_{v_i} é o número de vizinhos de v_i e N_{e_i} corresponde ao número de vizinhos de v_i que tambem são vizinhos entre si, i.e. formam um triângulo. O coeficiente de agrupamento varia no intervalo $[0, 1]$ e quantifica o nível de conectividade na região do vértice v_i, i.e. corresponde ao número de arestas que existem entre os vizinhos de um vértice pelo número máximo de vizinhos que poderiam existir. Dessa forma, como esperado, o coeficiente de agrupamento aumenta com o aumento de K, para os grafos K-associados, e diminui com o aumento da sobreposição entre as classes. Um fato interessante é que o coeficiente de agrupamento no grafo ótimo decresce mais rápido do que nos grafos 3-associado e 5-associado, conforme a sobreposição entre as classes aumenta. Isto se deve à propriedade do grafo ótimo de preservar estruturas localmente conectadas com o objetivo de manter a pureza alta.

3.4.3 Resultados comparativos e análise estatística

Esta seção apresenta os resultados de simulações comparando o classificador baseado no grafo K-associado ótimo (KAOG) e cinco outros algoritmos de classificação multiclasse. Três deles são algoritmos que se baseiam nos vizinhos mais próximos, o KNN, o KNN com peso e o KNN baseado em protótipos (Hastie *et al.*, 2009). Para todos os algoritmos que utilizam o conceito de vizinhos mais próximos, inclusive o algoritmo KAOG, a distância Euclidiana foi usada como medida de similaridade. Os demais algoritmos são o C4.5 (Quinlan, 1993) e uma versão multiclasse do SVM (M-SVM) (Vapnik, 1999). Os testes foram conduzidos considerando 15 base de dados multiclasse, ou seja com $\mu > 2$, onde μ corresponde ao número de classes; obtidas do repositório de dados UCI (Asuncion e Newman, 2007) e especificados na Tabela 3.1. Note que, exceto pelo algoritmo KAOG, todos são paramétricos, dessa forma, o método de validação cruzada com repetição (Filzmoser *et al.*, 2009) foi usado para ajustar os parâmetros para cada conjunto de dados. No caso dos algoritmos KNN e KNN com peso o único parâmetro é o número de vizinhos K a serem considerados. Para o ajuste do modelo foram usados todos os valores de K no intervalo de 1 ao número de instâncias na maior classe do conjunto de treinamento. Para o algoritmo KNN baseado em protótipo, o parâmetro consiste no número de protótipos por classes que deverão ser utilizados, para esta simulação foram considerados todos os protótipos no intervalo $[1, 30]$. Um protótipo pode ser visto como um padrão representante de um agrupamento local de dados com características similares e podem ser obtidos, por exemplo, pelo uso do algoritmo *k-means* (Hastie *et al.*, 2009). No caso do algoritmo SVM, a versão multiclasse escolhida foi a versão um-contra-um (*one-against-one*), na qual $\mu(\mu - 1)/2$ classificadores binários (SVM) distinguem entre todo par de classe através de algum esquema de votação. A versão multiclasse um-contra-um é preferível sobre outras versões porque no geral obtém melhores resultados (Hsu e Lin, 2002). Com o objetivo de diminuir o espaço de busca dentre os possíveis modelos de

Tabela 3.1: Especificações dos domínios de dados utilizados.

Domínio	**# Instâncias**	**# Atributos**	**# Classes**
Yeast	1484	8	10
Teaching	151	5	3
Zoo	101	16	7
Image	210	19	7
Wine	178	13	3
Iris	150	4	3
Glass	214	9	6
E.coli	336	8	8
Balance	625	4	3
Vowel	990	13	11
Libras	360	91	15
Hayes-Roth	132	5	3
Segment	2310	19	7
Vehicle	846	18	4
Wine Q. (Red)	1599	12	6

SVM, algums dos parâmetros foram fixados, tais como, o *kernel* utilizado foi o RBF $K(x_i, x_j) = e^{-\gamma\|x_i - x_j\|^2}$, e o critério de parada utilizado para o método de otimização *Karush-Kuhn-Tucker* considerou violação inferior a 10^{-3}, como utilizado por Hsu e Lin (2002). Para cada base de dados a seleção de modelo é realizada considerando o parâmetro do *kernel* $\gamma \in \{2^4, 2^3, 2^2, \ldots, 2^{-10}\}$ e o custo $C \in \{2^{12}, 2^{11}, 2^{10}, \ldots, 2^{-2}\}$, o que resulta em 125 possíveis modelos (classificadores) para cada base de dados. Para o algoritmo C4.5, também dois parâmetros foram considerados, o fator de confiança (*confidence factor*) que pode assumir os valores $cf \in \{0, 0.1, 0.25, 0.5, 0.8, 1\}$, onde os valores menores refletem maior poda (portanto, para $cf = 1$ não ocorre poda). O outro parâmetro refere-se ao número mínimo de instâncias de dados que um subconjunto deve ter para que este possa ser particionado $m \in \{0, 1, 2, 3, 4, 5, 10, 15, 20, 50\}$. Os parâmetros resultantes da seleção de modelos para cada algoritmo em cada base de dados são notados na Tabela 3.2.

Os resultados do uso de cada algoritmo consistem na média de 100 processos de validações cruzadas estratificado de 10 conjuntos (10-*fold stratified cross-validation*). A Tabela 3.3 apresenta os resultados de classificação de conjunto de teste seguido pelo desvio-padrão. Para cada resultado também são mostrados os parâmetros ajustados na fase de seleção de modelo pelo método de validação cruzada dupla repetida, bem como a posição do algoritmo em relação aos demais para cada base de dados. Para cada uma das 15 bases de dados, foram gerados dois novos conjuntos com diferentes níveis de ruído. Os conjuntos foram gerados a partir dos dados originais por meio da alteração aleatória

Tabela 3.2: Parâmetros resultantes da seleção de modelo. Parâmetros ajustados por algoritmo, KNN e KNN com peso (K), KNN protótipo (p), C4.5 (cf;m) e M-SVM (C; γ).

Domínio	**KNN**	**KNN c/ peso**	**KNN Prot.**	**C4.5**	**M-SVM**
Yeast	(k=15)	(k=16)	(p=1)	(0,1;5)	$(2^{11};2^{0})$
Yeast (5%)	(k=19)	(k=16)	(p=1)	(0,1;5)	$(2^{5};2^{2})$
Yeast (10%)	(k=11)	(k=11)	(p=1)	(0,1;15)	$(2^{10};2^{-1})$
Teaching	(k=1)	(k=9)	(p=29)	(1;0)	$(2^{6};2^{3})$
Teaching (5%)	(k=19)	(k=19)	(p=1)	(1;15)	$92^{5};2^{2})$
Teaching (10%)	(k=1)	(k=13)	(p=28)	(1;1)	$(2^{9};2^{-2})$
Zoo	(k=1)	(k=1)	(p=2)	(0,1;0)	$(2^{1};2^{1})$
Zoo (5%)	(k=1)	(k=3)	(p=2)	(1;0)	$(2^{1};2^{-2})$
Zoo (10%)	(k=5)	(k=3)	(p=2)	(0,5;5)	$(2^{3};2^{-2})$
Image	(k=3)	(k=3)	(p=16)	(0,8;3)	$(2^{10};2^{-3})$
Image (5%)	(k=3)	(k=3)	(p=15)	(1;0)	$(2^{9};2^{2})$
Image (10%)	(k=4)	(k=6)	(p=20)	(0,5;5)	$(2^{5};2^{-2})$
Wine	(k=1)	(k=1)	(p=28)	(0,5;1)	$(2^{11};2^{2})$
Wine (5%)	(k=5)	(k=9)	(p=17)	(1;3)	$(2^{10};2^{-3})$
Wine (10%)	(k=5)	(k=7)	(p=25)	(0,1; 5)	$(2^{9};2^{0})$
Iris	(k=19)	(k=19)	(p=3)	(0,25;2)	$(2^{-2};2^{3})$
Iris (5%)	(k=19)	(k=19)	(p=1)	(1;0)	$(2^{0};2^{2})$
Iris (10%)	(k=20)	(k=19)	(p=1)	(0,5;5)	$(2^{-1};2^{3})$
Glass	(k=1)	(k=1)	(p=6)	(0,1;3)	$(2^{10};2^{4})$
Glass (5%)	(k=7)	(k=5)	(p=9)	(0,1;5)	$(2^{8};2^{3})$
Glass (10%)	(k=5)	(k=5)	(k=9)	(0,1;4)	$(2^{10};2^{3})(2)$
E.coli	(k=9)	(k=9)	(p=2)	(0,25;3)	$(2^{12};2^{-9})$
E.coli (5%)	(k=13)	(k=11)	(p=1)	(0,1;4)	$(2^{1};2^{2})$
E.coli (10%)	(k=7)	(k=13)	(p=1)	(0,25;15)	$(2^{2};2^{2})$
Balance	(k=1)	(k=11)	(p=8)	(0,5;1)	$(2^{7};2^{0})$
Balance (5%)	(k=13)	(k=13)	(p=1)	(0,25;1)	$(2^{-2};2^{3})$
Balance (10%)	(k=14)	(k=12)	(p=1)	(0,25;15)	$(2^{-2};2^{3})$
Vowel	(k=1)	(k=11)	(p=8)	(0,5;0)	$(2^{7};2^{0})$
Vowel (5%)	(k=3)	(k=4)	(p=28)	(0,5;0)	$(2^{5};2^{2})$
Vowel (10%)	(k=4)	(k=5)	(p=29)	(0,25;2)	$(2^{5};2^{3})$
Libras	(k=1)	(k=1)	(p=13)	(0,8;1)	$(2^{7};2^{2})$
Libras (5%)	(k=1)	(k=5)	(p=8)	(1;0)	$(2^{7};2^{3})$
Libras (10%)	(k=5)	(k=5)	(p=15)	(0,1;3)	$(2^{8};2^{3})$
Hayes-Roth	(k=1)	(k=3)	(p=8)	(1;0)	$(2^{12};2^{3})$
Hayes-Roth (5%)	(k=1)	(k=4)	(p=9)	(0,8;5)	$(2^{12};2^{-10})$
Hayes-Roth (10%)	(k=3)	(k=3)	(p=9)	(0,1;50)	$(2^{5};2^{3})$
Segment	(k=1)	(k=5)	(p=25)	(1;0)	$(2^{11};2^{0})$
Segment (5%)	(k=4)	(k=6)	(p=29)	(1;0)	$(2^{8};2^{1})$
Segment (10%)	(k=6)	(k=7)	(p=28)	(0,1;5)	$(2^{9};2^{1})$
Vehicle	(k=3)	(k=5)	(p=10)	(0,5;2)	$(2^{10};2^{3})$
Vehicle (5%)	(k=6)	(k=6)	(p=9)	(0,1;3)	$(2^{10};2^{3})$
Vehicle (10%)	(k=7)	(k=7)	(p=10)	(0,1;10)	$(2^{12};2^{3})$
Wine Q. Red	(k=1)	(k=19)	(p=27)	(1;0)	$(2^{9};2^{1})$
Wine Q. Red (5%)	(k=5)	(k=19)	(p=29)	(1;0)	$(2^{11};2^{-1})$
Wine Q. Red (10%)	(k=1)	(k=19)	(p=26)	(0,1;2)	$(2^{10};2^{-2})$

5 e 10% das classes originais. Os resultados de simulação conduzidos nesses novos dados também são apresentados na Tabela 3.3. Como pré-processamento, cada instância foi normalizada para ter magnitude 1, dividindo cada atributo pela distância da instância à origem. Ou seja, a instância $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{ip}, y_i)$ é normalizado dividindo cada atributo x_{ij} por $(\sum_{j=1}^{p} x_{ij}^2)^{\frac{1}{2}}$.

Para comparar múltiplos classificadores em múltiplos domínios de dados será usado um método descrito por Demšar (2006) (bem como as referências mencionadas no trabalho). Nesta metodologia, ordena-se, de forma descendente, as médias de acertos de classificação para cada conjunto e, atribui-se a posição de cada algoritmo nesta ordem. A média das posições de cada algoritmo é então calculada e usada no teste de Friedman para verificar se as médias são significativamente diferentes da média das posições ($3,5$ no caso estudado). O teste de Friedman usa a distribuição F para calcular o valor crítico que será usado para rejeitar a hipótese nula, definida segundo um grau de confiança. Considere N_a o número de algoritmos testados e N_d o número de bases de dados utilizados, o valor crítico é definido por meio do cálculo do valor da distribuição F para os graus de liberdade $(N_a - 1)$ e $(N_a - 1) \times (N_d - 1)$. No caso estudado $N_a = 6$, $N_d = 15$ e considerando grau de confiança de $0,05$, o valor crítico para o experimento é $F(5; 70) \approx 2,35$. Dessa forma, qualquer resultado para o teste de Friedman maior que o valor crítico rejeita a hipótese nula. No caso em questão, o teste de Friedman resulta em $F_F \approx 7,49$, o que confirma a rejeição da hipótese nula, portanto os algoritmos estudados não são equivalentes.

Até agora o que se pode dizer sobre os algoritmos estudados é que eles não são equivalentes nos resultados de teste, para as bases de dados consideradas. Com o objetivo de estabelecer alguma conclusão considerando a performance do algoritmo proposto em relação aos demais, é necessário prosseguir com um teste *post-hoc*. Como comentado por Demšar (2006), não se deve comparar todos os algoritmos par a par quando deseja-se comparar um novo algoritmo com algoritmos já existentes. Para tal propósito, existem métodos de teste como o de Bonferroni-Dunn, que consiste em comparar um algoritmo controle, neste caso o KAOG, com os demais, evitando comparações desnecessárias. Considere a aplicação do teste de Bonferroni-Dunn, a diferença critica (DC) é calculada para um determinado nível de significância. Qualquer diferença maior que DC entre a média das posições, para um dado par de algoritmos, confirma que eles são significativamente diferentes. De volta ao caso em estudo o valor crítico para nível de significância de $0,05$, para o teste de Bonferroni-Dunn é de $DC = 1,76$. Portanto, qualquer diferença na média das posições (entre dois algoritmos) maior que este valor justifica a superioridade do algoritmo com menor valor para a média.

Para o nível de significância de $0,05$, e enquanto esses algoritmos e bases de dados sejam considerados, pode-se concluir que o algoritmo proposto (KAOG) apresentou resultados significativamente melhores que o KNN baseado em protótipos. As comparações par a par entre o algoritmo KAOG e os demais algoritmos resultaram em diferença menor

Tabela 3.3: Resultados de comparação considerando quinze domínios, cada qual com três niveis ruído. Cada resultado corresponde à média de teste medida em 100 resultados de validação cruzada, seguida pelo respectivo desvio padrão e a posição comparativa com relação aos demais algoritmos.

Domínio	**KAOG**	**KNN**	**KNN c.p.**	**KNN Prot.**	**C4.5**	**M-SVM**
Yeast	53,6±3,8 (5)	58,7±3,4(3)	**60,9±3,6**(1)	48,0±2,7(6)	55,8±3,6(4)	58,9±4,8(2)
Yeast (5%)	50,7±3,6(5)	53,9±3,9(3)	54,3±4,1(2)	44,9±2,8(6)	52,2±3,7(4)	**54,5±5,3**(1)
Yeast (10%)	46,8±3,9(5)	49,8±3,6(2)	**50,6±3,7**(1)	38,8±3,2(6)	47,9±4,3(3)	47,3±3,9(4)
Teaching	62,5±11,6(2)	59,6±10,2(3)	**63,0±12,3**(1)	58,3±9,0(4)	58,2±14,9(5)	52,5±7,9(6)
Teaching (5%)	**59,5±16,3**(1)	53,9±3,9(4)	56,0±4,0(2)	44,9±2,8(6)	50,8±3,3(5)	54,5±5,3(3)
Teaching (10%)	**55,1±7,5**(1)	55,0±12,1(2)	47,9±12,6(4)	46,0±12,5(6)	52,3±14,8(3)	46,8±16,7(5)
Zoo	**97,0±5,2**(1)	96,1±5,9(4)	96,2±5,8(3)	93,6±7,1(6)	95,8±5,3(5)	96,3±6,4(2)
Zoo (5%)	**84,1±14,2**(1)	81,6±10,8(5)	82,5±10,2(3)	81,8±14,0(4)	83,1±11,1(2)	74,3±9,3(6)
Zoo (10%)	**76,3±11,6**(1)	72,1±13,9(4)	69,1±13,0(6)	75,8±6,9(2)	75,2±14,9(3)	71,9±11,0(5)
Image	75,3±7,5(5,5)	75,3±8,2(5,5)	75,4±8,2(4)	75,5±10,2(3)	80,7±7,6(2)	**86,7±7,4**(1)
Image (5%)	64,3±11,3(6)	64,8±9,3(4)	67,1±8,4(3)	64,7±8,5(5)	**74,7±9,0**(1)	71,4±7,0(2)
Image (10%)	56,2±8,0(5)	56,1±10,0(6)	62,6±10,3(3)	56,9±8,5(4)	62,7±10,9(2)	**69,0±7,1**(1)
Wine	85,3±8,5(3)	84,1±8,5(4,5)	84,1±8,2(4,5)	81,4±11,9(6)	91,7±6,7(2)	**94,4±5,8**(1)
Wine (5%)	76,2±10,2(3)	74,0±10,0(5)	74,5±10,1(4)	69,0±9,7(6)	82,5±4,1(2)	**85,5±7,0**(1)
Wine (10%)	69,9±10,4(3)	68,8±10,4(4)	67,0±10,7(5)	61,9±9,3(6)	76,6±10,0(2)	**78,3±12,4**(1)
Iris	97,4±3,2(2)	**97,9±3,4**(1,5)	**97,9±3,3**(1,5)	97,3±3,2(4)	95,0±5,8(6)	97,0±4,6(5)
Iris (5%)	**88,6±8,8**(1)	88,5±7,7(2)	88,4±8,0(3)	84,0±7,3(6)	86,0±7,6(5)	88,0±6,8(4)
Iris (10%)	81,4±9,4(3)	**84,2±9,2**(1)	79,4±10,1(5)	80,0±14,4(4)	78,8±10,1(6)	82,6±9,5(2)
Glass	**72,5±8,1**(1)	71,9±8,6(2)	71,8±9,0(3)	67,3±11,8(5)	66,9±9,4(6)	69,5±5,6(4)
Glass (5%)	62,5±11,6(3)	**64,6±9,0**(1)	62,4±10,1(4)	58,5±13,1(6)	64,0±8,6(2)	61,3±6,5(5)
Glass (10%)	57,9±9,5(4)	53,7±9,3(5)	59,9±10,3(3)	52,8±8,0(6)	**60,8±11,3**(1)	60,0±4,7(2)
E.coli	85,8±6,4(4)	86,5±5,2(3)	**87,4±5,4**(1)	80,4±5,2(6)	83,6±6,1(5)	86,7±8,2(2)
E.coli (5%)	79,0±9,2(2)	78,5±6,4(3)	**79,4±6,3**(1)	69,4±6,8(6)	76,2±7,1(5)	78,2±7,2(4)
E.coli (10%)	65,8±8,2(5)	71,9±6,9(2)	**72,4±6,9**(1)	59,3±12,3(6)	67,4±7,1(4)	70,2±7,3(3)
Balance	94,9±2,5(3)	94,7±2,6(4)	96,7±2,1(2)	76,6±5,7(6)	89,6±3,7(5)	**98,2±0,9**(1)
Balance (5%)	79,5±4,6(4)	83,1±4,5(2)	**86,6±4,0**(1)	71,7±4,3(6)	79,4±5,0(5)	82,3±3,1(3)
Balance (10%)	72,0±5,2(5)	75,8±4,6(3)	**78,3±4,2**(1)	64,9±5,7(6)	73,3±5,4(4)	76,9±3,6(2)
Vowel	**98,9±0,7**(1)	97,8±1,0(3)	98,8±0,9(2)	96,6±1,8(5)	78,6±4,3(6)	97,5±1,9(4)
Vowel (5%)	**85,7±3,9**(1)	83,9±3,4(2)	72,7±6,7(5)	78,5±3,8(4)	65,7±4,7(6)	83,1±3,8(3)
Vowel (10%)	**74,5±6,4**(1)	73,7±3,4(2)	63,7±8,0(5)	66,9±4,3(4)	54,7±4,7(6)	72,1±3,8(3)
Libras	85,4±5,3(2)	84,8±5,5(3,5)	84,8±5,4(3,5)	55,7±41,2(6)	71,6±7,5(5)	**86,6±5,0**(1)
Libras (5%)	71,9±6,9(2)	69,4±6,7(4)	**72,8±6,7**(1)	44,6±33,2(6)	58,6±8,5 (5)	70,0±7,6(3)
Libras (10%)	59,9±5,9(2)	59,7±7,2(4)	**61,8±7,2**(1)	37,5±28(6)	50,7±8,2(5)	58,1±8,5(3)
Hayes-Roth	55,7±12,6(2)	54,9±12,4(3)	**56,8±13,2**(1)	44,6±11,3(6)	46,7±10,5(4)	45,4±13,1(5)
Hayes-Roth (5%)	**52,4±12,4**(1)	51,3±12,7(2)	47,4±13,6(3)	44,0±11,7(4)	46,6±10,5(5)	42,9±17,2(6)
Hayes-Roth (10%)	**49,9±13,5**(1)	49,5±12,1(2)	45,0±13,2(4)	47,7±11,6(3)	46,8±10,5(5)	33,4±11,6(6)
Segment	93,7±1,5(3)	93,6±1,6(4,5)	93,6±1,4(4,5)	60,9±3,2(6)	94,5±1,2(2)	**96,6±1,2**(1)
Segment (5%)	80,3±2,3(5)	82,7±2,3(3)	82,3±2,4(4)	53,3±3,1(6)	84,7±2,3(2)	**86,5±2,4**(1)
Segment (10%)	74,9±2,8(3)	74,7±2,4(4)	72,4±2,7(5)	46,3±3,2(6)	75,6±2,5(2)	**77,6±3,7**(1)
Vehicle	67,9±4,4(3)	67,3±4,0(5)	67,6±4,1(4)	60,0±5,5(6)	70,7±3,5(2)	**84,4±3,4**(1)
Vehicle (5%)	58,8±4,5(5)	60,1±4,4(3)	60,9±4,3(4)	51,6±5,0(6)	61,9±4,6(2)	**75,0±4,0**(1)
Vehicle (10%)	52,5±4,7(5)	53,7±4,0(4)	56,8±4,5(3)	47,2±4,5(6)	58,5±4,5(2)	**67,1±5,6**(1)
Wine Q. Red	61,8±3,6(2)	61,3±3,4(3)	**64,0±3,8**(1)	38,9±3,1(6)	59,8±2,4(5)	60,4±3,2(4)
Wine Q. Red (5%)	**57,9±3,9**(1)	55,5±3,2(4)	57,5±4,0(2)	38,4±3,5(6)	53,4±1,9(5)	55,6±2,1(3)
Wine Q. Red (10%)	50,8±3,1(3)	50,4±3,5(4)	**54,1±3,6**(1)	33,3±3,3(6)	50,2±3,2(5)	52,3±3,2(2)
Média das posições nos dados originais	2,63	3,50	2,46	5,40	4,26	2,66
Média das posições nos dados com ruído	2,93	3,20	3,00	5,30	3,63	2,90
Média das posições em todos os dados	2,83	3,30	2,82	5,35	3,84	2,82

que o valor crítico DC, o que significa que a diferença entre eles é estatisticamente insignificante dado as condições de análise. Portanto, apesar do algoritmo KNN com peso apresentar desempenho ligeiramente superior ao do algoritmo proposto, KAOG, nas bases de dados originais, a diferença na média das posições é muito pequena para afirmar sua superioridade. Estes resultados conduzem à conclusão estatística: enquanto considerado esses domínios de dados, o algoritmo proposto não apresenta diferença significativa em comparação com os algoritmos KNN, KNN com peso, M-SVM e C4.5. Em outras palavras, o algoritmo KAOG apresenta desempenho similar aos algoritmos testados, com exceção ao KNN baseado em protótipo. Este resultado é bastante interessante para o algoritmo proposto, pois o KAOG não exige seleção de modelo. O que significa que o classificador proposto apresenta precisão de classificação comparável a bons classificadores, sem a necessidade de ajustar qualquer parâmetro, o que torna o algoritmo KAOG uma opção eficiente na classificação multiclasse.

Avançando com a análise, considere os domínios com ruído ($N_d = 30$) bem como o resultado para todos os domínios considerados ($N_d = 45$) para o teste de Friedman. Como feito anteriormente para as bases de dados originais, o primeiro passo é determinar o valor de F para avaliar a hipótese nula. Considerando as novas configurações, os valores para os domínios com ruído e para todos os domínios são respectivamente $F(5; 145) \approx 2,27$ e $F(5; 220) \approx 2,25$. O cálculo para o teste de Friedman para ambos os casos resulta em $F_F = 15,52$ e $F_F = 23,55$, resultando na negação da hipótese nula para ambos os casos. O próximo passo consiste no cálculo da diferença crítica DC; para o caso dos domínios com ruído, a diferença resulta em $DC = 1,24$. Novamente, qualquer diferença maior que DC é considerado estatisticamente significante, o que leva a conclusão similar à conclusão sobre os conjuntos originais. O último grupo a ser analisado é o grupo composto por todos os domínios, neste grupo a diferença crítica encontrada é $DC = 0,4$. A comparação entre o algoritmo KAOG e os demais algoritmos para todos os domínios mostra que o algoritmo proposto tem melhor performance que os algoritmos KNN, KNN baseado em protótipo e C4.5, ao passo que possui performance comparável aos algoritmos KNN com peso e M-SVM.

Capítulo

4

Classificação incremental usando o grafo K-associado ótimo

Devido principalmente à mudança de conceito, não é possível para um classificador estático, treinado somente com um conjunto inicial, ser utilizado na classificação de dados não estacionários mantendo desempenho aceitável. Para tal propósito, é necessário que o classificador seja atualizado constantemente durante o processamento do fluxo de dados, o que caracteriza o aprendizado incremental. Este termo identifica, de maneira geral, todo tipo de sistema de aprendizado capaz de aprender durante o processo de classificação. O que significa que o sistema de aprendizado deve incorporar conhecimento de novos exemplos rotulados e classificar novos exemplos não rotulados, em qualquer ordem de chegada ou independente de serem apresentados em conjuntos ou individualmente. O caso específico em que as instâncias de dados rotuladas ou não rotuladas de um fluxo de dados são apresentadas individualmente consiste em um problema de aprendizado em tempo real. Em ambos os casos, o tempo gasto no treinamento, ou atualização, do classificador é uma restrição importante para esse tipo de sistema. O que, além de inviabilizar o uso de classificadores com alta complexidade computacional, também dificulta o uso de classificadores que possuem muitos parâmetros - dado o grande volume de dados neste tipo de aplicação, o custo associado à seleção de modelos é alto.

O bom desempenho do algoritmo estático, KAOG, bem como algumas de suas características, como a ausência de parâmetro e a baixa complexidade computacional, inspiraram a extensão de sua pesquisa para domínios de dados com distribuição não estacionária (Bertini Jr. *et al.*, 2011a). Intuitivamente, as propriedades do classificador estático podem ser adaptadas para satisfazer os objetivos e as restrições inerentes à classificação incremental. A continuação da pesquisa neste sentido resultou no desenvolvimento da versão incremental do algoritmo KAOG, chamada KAOGINC. Basicamente, a maior

diferença do algoritmo incremental em relação ao algoritmo estático, está no grafo usado na classificação de novos exemplos - dado que, no contexto incremental, o grafo é uma estrutura dinâmica que precisa ser atualizada com novos conhecimentos e livrar-se de conhecimentos antigos. De acordo com as categorizações de algoritmos incrementais, apresentadas na Seção 2.4.1, o classificador proposto enquadra-se nas categorias de e algoritmos com armazenamento parcial de instâncias e de conceitos. Como ficará claro adiante nesta seção, o algoritmo KAOGINC utiliza instâncias de dados armazenadas para o cálculo da distância para a classificação de novos exemplos de dados, mas também se desfaz de padrões antigos ao longo do tempo. Já a segunda categorização, armazenamento parcial de conceitos, deve-se à constante troca de componentes no grafo - dado que cada componente representa parte da descrição de um conceito. Cada componente é parametrizado com o fim de mensurar sua atividade e performance, de modo que possam ser retirados do grafo devido a alterações nestes parâmetros - causados por eventuais mudanças de conceito.

No que segue, será apresentado em detalhes o classificador incremental KAOGINC seguido pela complexidade computacional do algoritmo. Também é apresentada uma comparação com o algoritmo estático, KAOG, em domínios não estacionários, com o objetivo de validar a proposta incremental. Como o algoritmo incremental possui um parâmetro, este é análisado quanto ao seu comportamento, na mesma seção em que a poda do grafo é apresentada. Por último, algumas comparações com outros algoritmos de aprendizado incremental, em domínios reais e artificiais, são apresentados, seguidos pela análise estatística dos resultados obtidos.

4.1 O método incremental para dados com distribuição não estacionária

Considerando um cenário no qual, dados rotulados e não rotulados são apresentados em um fluxo de dados $S = \{X_1, Z_1, \ldots, X_t, Z_t, \ldots, X_T, Z_T\}$, podendo ser processado como uma sequência de diferentes conjuntos com dados rotulados X_t e não rotulados e Z_t. Os objetivos de um classificador indicado para processar fluxo de dados são, manter um desempenho estável e aceitável em relação à porcentagem de acertos na classificação e, ao tempo gasto no processamento de novos dados. De acordo com Giraud-Carrier (2000), para satisfazer esses objetivos um classificador deve implementar as seguintes tarefas durante a fase de aplicação: (i) incorporar novos conhecimentos, provindos de dados rotulados ao longo do tempo, (ii) predizer as classes dos dados não rotulados e (iii) desfazer-se de conhecimento antigo que não será mais utilizado.

O algoritmo de aprendizado incremental proposto, KAOGINC, análogo ao algoritmo estático, KAOG, utiliza um grafo gerado a partir dos dados para estimar as probabilidades pertinência de um novo exemplo em relação aos vários componentes do grafo. No entanto, para que possa ser usada na classificação incremental, a estrutura do grafo não pode ser

estática como o grafo ótimo - usado na classificação estacionária. Por essa razão, o algoritmo KAOGINC utiliza uma versão dinâmica do grafo ótimo, chamada *grafo principal*. O grafo principal é formado por meio da adição contínua dos componentes pertencentes aos vários grafos ótimos, construídos a partir de cada conjunto rotulado apresentado durante a aplicação do classificador. O esquema geral de como as tarefas, previamente descritas, são desempenhadas pelo algoritmo proposto, KAOGINC, por meio de alterações no grafo principal, é mostrado na Figura 4.1.

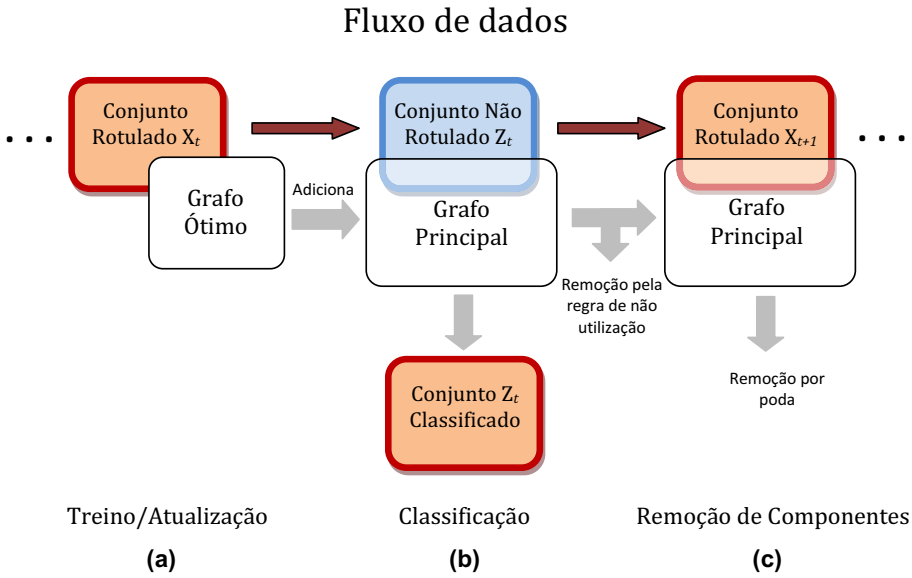

Figura 4.1: Esquema de processamento de fluxo de dados, apresentados em conjuntos rotulados e não rotulados, realizado pelo algoritmo KAOGINC.

Na Figura 4.1, duas das três tarefas representadas, dizem respeito à manutenção do classificador e correspondem às tarefas que o tornam incremental, a saber atualização do grafo principal, com novo conhecimento (Figura 4.1(a)), e remoção de conhecimento antigo Figura (4.1(c)). A tarefa (i) refere-se à atualização (ou treinamento) do classificador com novos conceitos. No algoritmo proposto a atualização do grafo principal acontece toda vez que um novo conjunto rotulado é apresentado. O procedimento consiste em criar um grafo ótimo a partir do novo conjunto rotulado atual, X_t, e adicioná-lo ao grafo principal, como mostrado na Figura 4.1(a). A tarefa (ii), como mostrado na Figura 4.1(b), corresponde à classificação de dados não rotulados, e é realizada sempre que um conjunto com dados não rotulados é apresentado ao classificador. O processo de classificação incremental é semelhante ao processo realizado pelo algoritmo estático, com a única resalva de que, no caso incremental, usa-se o grafo principal como estrutura para inferir a classe dos novos exemplos de dados. A tarefa (iii) está relacionada diretamente ao tamanho do grafo principal, bem como ao desempenho do classificador, e corresponde à eliminação de componentes do grafo. Como mostrado na Figura 4.1(c), os componente podem ser retirados do grafo por meio da poda ou da regra de não utilização, explicadas adiante. A retirada de componentes controla o tamanho do grafo principal, e interfere diretamente no tempo e no desempenho de classificação. O algoritmo KAOGINC é apresentado em

detalhes no Algoritmo 4.1.

Algoritmo 4.1 Classificador incremental KAOGINC

Entrada: $S = \{X_1, Z_1, ..., X_T, Z_T\}$ {Fluxo de dados, conjuntos rotulados e não rotulados}
$X_t = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$ {Conjuntos rotulados apresentados ao longo do tempo}
$Z_t = \{\mathbf{z}_1, ..., \mathbf{z}_M\}$ {Conjuntos não rotulados apresentados ao longo do tempo}
τ {Parâmetro de esquecimento}
Saída: $\varphi(\mathbf{z})$ {classe para os padrões não rotulados}

```
repita
    se X_t então
        poda(G_P, X_t)
        G_P ⇐ G_P ∪ KAOG(X_t)
    senão
        para todo z_j ∈ Z_t faça
            φ(z_j) ⇐ classificadorIncremental(G_P, z_j)
        fim para
        para todo C_α ⊂ G_P faça
            se t_α > τ então
                G_P ⇐ G_P − C_α
            fim se
        fim para
    fim se
até que S = ∅
```

Inicialmente o algoritmo $KAOG()$ (Seção 3.1.2) cria o grafo K-associado ótimo a partir do conjunto de treinamento X_1, idêntico ao que seria produzido no contexto do classificador estático. Então o grafo ótimo é adicionado ao grafo principal, G_P, para que seja usado na classificação de novos padrões não rotulados. Como mostrado no Algoritmo 4.1, sempre que um conjunto rotulado é disponibilizado, durante a fase de classificação, o algoritmo KAOGINC cria um grafo ótimo, utilizando o algoritmo KAOG, e o adiciona no grafo principal. Lembre-se que o grafo ótimo é formado por componentes, variando no intervalo $N \geq N_c \geq \mu$, sendo N o número de vértices no grafo, N_c o número de componentes e μ o número de classes.

A adição de novos componentes no grafo principal atualiza o grafo com dados dos novos conceitos, mas também promove o crescimento do grafo. Para evitar o crescimento descontrolado do grafo principal, o algoritmo KAOGINC possui duas formas para retirar do grafo os componentes indesejados, a regra de não utilização e a poda. A regra de não utilização de componentes retira do grafo principal todo componente que não esteja sendo usado na classificação de novos exemplos. Esta é definida por meio do parâmetro τ que determina o número de classificações de novos exemplos, feitas pelo algoritmo, para as quais, qualquer componente pode permanecer no grafo principal sem que seja utilizado. Considere que o componente C_α tenha um parâmetro de esquecimento, notado por t_α, cujo objetivo seja armazenar o número de classificações consecutivas em que o componente C_α não foi utilização. Este parâmetro é utilizado para indicar que um determinado componente não é mais útil e pode ser retirado do grafo principal. Para exemplificar o mecanismo de retirada de componentes do grafo principal, considere a vida útil do componente C_α. No momento em que o componente C_α é criado, o parâmetro de esquecimento é inicializado, $t_\alpha = 0$ e, é incrementado de um, toda vez que este não for utilizado na

classificação de um novo exemplo. Entretanto, sempre que o componente C_α for utilizado, t_α é zerado. O componente C_α é retirado do grafo principal quando o parâmetro de esquecimento for maior que o limiar pré-determinado, $t_\alpha > \tau$. Note, no algoritmo 4.1, que a retirada dos componentes é feita depois da classificação do conjunto, ao invés de verificar ao final de cada classificação. Dessa forma, além de economizar tempo computacional não atrapalha no processo de classificação e permite um relaxamento na regra de remoção de componentes, ou seja, o componente será retirado do grafo se este não for utilizado durante τ classificações ou mais em um conjunto não rotulado.

Como visto a retirada de componentes pela regra de não utilização é realizada durante a fase de classificação de dados ainda não vistos, portanto, não é possível saber se a classificação resultou em acerto ou erro - o único objetivo é verificar se todos os componentes estão sendo usados. Dessa forma, imagine que um componente não mais represente o conceito corrente, ainda assim este pode permanecer no grafo somente cometendo erros, dado que este não será removido do grafo pela regra de não utilização. Isso ocorre devido a ruídos presentes nos dados que possam ser confundido com o conceito passado, representado pelo componente em questão. Esse tipo de situação se agrava quando ocorre uma mudança no conceito, principalmente se a mudança ocorrer somente na distribuição das classes. Neste caso, como somente as classes serão alteradas, os novos dados localizar-se-ão no mesmo espaço de atributos que os dados do conceito anterior e serão classificados erroneamente por esses componentes. Já, se a mudança de conceito ocorrer devido à alteração na distribuição dos dados, a retirada de componentes de conceitos anteriores dependerá do espaço de atributos para o qual os novos dados mudaram-se, ou seja, se os espaços de atributos dos dados antigos não se inter-relacionarem com o espaço dos novos dados, eventualmente os componentes antigos serão retirados do grafo em τ iterações. Consequentemente, conclui-se que o parâmetro de esquecimento, apesar de medir a utilização dos componentes no grafo, não é suficiente para retirar do grafo os componentes de conceitos passados que permanecem no grafo provocando erro, especialmente durante as mudanças de conceitos.

O outro artifício, mais rigoroso, para remoção de componentes do grafo principal é a poda, implementada pelo método *poda*() no Algoritmo 4.1. Basicamente, a poda consiste em retirar do grafo principal todo componente cuja utilização resultou em erro. Para isso o método de poda utiliza o método de classificação em um conjunto rotulado para verificar os erros e eliminar os componentes associados a eles. Portanto, a poda comporta-se de acordo com o erro do classificador, quando o classificador mantém porcentagem de acertos alta, por exemplo, durante conceito estável, a poda é sutil retirando do grafo principal somente alguns componentes devido a ruído ou sobreposição entre os dados. Já se o classificador apresenta muitos erros, a poda é agressiva, por exemplo, quando ocorre uma mudança de conceito a queda no desempenho do classificador faz com que a poda elimine muitos dos componentes do conceito antigo. A poda é útil principalmente quando ocorre mudanças de conceito, pelo fato desta remover rapidamente os componentes de conceitos

antigos. O que permite ao classificador recuperar rapidamente o desempenho de classificação deteriorado pela sobreposição de componentes de diferentes classes, ocasionado pela mudança de conceito. Durante as fases de conceito estático, a poda ajuda a manter o tamanho do grafo principal estável, retirando do grafo principalmente os componentes que, somente pela regra de não utilização, poderiam permanecer no grafo cometendo erro.

Quando um conjunto não rotulado é apresentado, o algoritmo classifica todo exemplo do conjunto utilizando o classificador apresentado no Algoritmo 4.2, representado pelo método $classificadorIncremental()$ no Algoritmo 4.1. As únicas diferenças entre os métodos $classificadorIncremental()$ utilizado no KAOGINC, e o método de classificação utilizado pelo algoritmo KAOG, é que no caso incremental, o algoritmo controla o parâmetro t_α dos componentes. Bem como a procura por vizinhos mais próximos, como explicada adiante, é realizada utilizando informações dos componentes para amenizar o esforço computacional no cálculo das distâncias, devido ao grande volume de dados envolvido neste tipo de aplicação. Os detalhes do processo de classificação do classificador incremental são mostrados no Algoritmo 4.2.

Algoritmo 4.2 Algoritmo do processo de classificação - classificador incremental

Entrada: G_P {Grafo principal} e
  z {novo exemplo a ser classificado}
Saída: $\varphi(v_z)$ {Classe atribuída ao exemplo **z**}
  repita
    $temVizinho \Leftarrow$ **falso**
    $C_\alpha \Leftarrow encontraProximoVizinho(\mathbf{c}, \mathbf{z})$
    $\Lambda_{v_z,K_\alpha} \Leftarrow encontraVizinhos(X_\alpha, \mathbf{z}, K_\alpha)$
    para todo $v_i \in C_\alpha$ **faça**
      se $v_i \in \Lambda_{v_z,K_\alpha}$ **então**
        $N_\alpha \Leftarrow N_\alpha + 1$
        $temVizinho \Leftarrow$ **verdadeiro**
      fim se
    fim para
  até que não $temVizinho$
  para todo $C_\beta \in G_P$ **faça**
    se $N_{v_z,C_\beta} \neq 0$ **então**
      $t_\beta \Leftarrow 0$
      $P(v_z \in C_\beta) \Leftarrow (N_{v_z,C_\beta}/K_\beta)P(C_\beta)$
      $P(N) \Leftarrow P(N) + P(v_z \in C_\beta)$
    senão
      $t_\beta \Leftarrow t_\beta + 1$
    fim se
  fim para
  para $j = 1$ **até** μ **faça**
    se $\hat{C_\beta} = \omega_j$ **então**
      $P(\omega_j) \Leftarrow P(\omega_j) + P(v_z \in C_\beta)/P(N)$
    fim se
  fim para
  retorne $\varphi(v_z) = argmax\{P(\omega_1), \ldots, P(\omega_\mu)\}$

No processo de classificação do KAOGINC, diferente do classificador estático, a classificação depende do número de vértices no grafo principal que, em geral, possui muito mais vértices que nos grafos ótimos. Considerando a grande quantidade de vértices, comparar todo novo vértice com todos os vértices do grafo, em busca dos vizinhos mais próximos,

torna-se custoso no contexto de aprendizado incremental. Para suavizar o custo computacional envolvido no cálculo da distância entre todos os vértices do grafo principal, considere que cada componente C_α possua um centro associado, c_α, calculado como na Equação (4.1).

$$c_\alpha = (c_1, \ldots, c_j, \ldots, c_p) \quad \text{tal} \quad \text{que} \quad c_j = \frac{\sum_{v_i \in C_\beta} x_j}{N_\beta} \tag{4.1}$$

Na equação cada atributo c_j corresponde à média dos valores dos atributos dos dados cujos vértices pertencem ao componente C_β, e N_β é o número de vértices no componente C_β. Dessa forma, no processo de classificação, ao invés de calcular a distância entre uma nova instância e todas as instâncias que compõem o grafo principal, calcula-se inicialmente as distâncias entre a nova instância e os centros dos componentes. Uma vez calculadas as distâncias, os componentes são considerados em ordem crescente de proximidade para o novo vértice e, então, procura-se pelos vizinhos mais próximos dentre os vértices do componente em questão. No Algoritmo 4.2 o método $encontraProximoVizinho()$ retorna o próximo componente na ordem de próximidade, calculado entre $\mathbf{z}$ e o vetor $\mathbf{c}$, com os centros dos componentes, i.e. na primeira chamada retorna o componente de centro mais próximo, na segunda, o componente cujo centro é o segundo mais próximo, e assim por diante. Seja v_z o vértice referente ao exemplo $\mathbf{z}$ que deve ser classificado, e C_α o componente com centro mais próximo do vértice v_z. O próximo passo é encontrar, dentre os vértices pertences a C_α, os K_α vizinhos mais próximos de v_z, determinado pelo método $encontraVizinhos()$, que os procura no conjunto de dados X_α que formou o componente C_α. Depois disso, o componente com o segundo centro mais próximo é considerado, prosseguindo dessa forma até o ponto em que um componente C_β não possuir nenhum vértice dentre os K_β vizinhos mais próximos do vértice a ser classificado v_z. No Algoritmo este mecanismo de parada é controlado pela variável $temVizinho$.

Uma vez determinada as conexões, define-se as probabilidades de pertinência para cada componente, de maneira semelhante à feita no algoritmo estático. No entanto, no algoritmo incremental, para que um componente permaneça no grafo principal este deve ser usado com frequência, ou é retirado do grafo, de acordo com a regra de não utilização. De modo que, a cada classificação, todo componente C_β nos quais v_z não se conectou $N_{v_z,C_\beta} = 0$, têm seu parâmetro de tempo, t_β, incrementado de um. Ao passo que toda vez que um componente C_β for usado, i.e. $N_{v_z,C_\beta} > 0$, o parâmetro t_β é zerado.

Como visto o a Algoritmo KAOGINC processa conjuntos de dados apresentados em sequência, como um fluxo de dados. Por ser incremental, o algoritmo utiliza cada conjunto rotulado para construir um grafo ótimo e adiciona-lo ao grafo principal. Portanto, o bom desempenho do algoritmo depende da formação dos componentes a partir dos conjuntos rotulados. De modo que, se o conjunto de dados for muito pequeno, este provavelmente não formará bons componentes pelo fato de não representar bem o espaço de dados, i.e. é pequena a probabilidade dos exemplos, presentes no conjunto, serem próximos uns dos outros a ponto de formar um componente que represente bem o espaço de dados em que se

encontra. Ao passo que, conjuntos maiores apresentam uma amostragem maior do dado em questão, o que aumenta a probabilidade de formação de componentes que representem de maneira apropriada o espaço de dados em que se encontra.

4.2 Complexidade computacional

Considere obter a complexidade computacional para o classificador incremental proposto, KAOGINC. Para isso, considere a divisão do algoritmo em três partes, como ilustrado no esquema da Figura 4.1, 1) treinamento ou atualização do grafo principal, 2) classificação de novos dados e 3) poda e remoção de componentes do grafo principal pela regra de não utilização. Como o fluxo de dados é apresentado em conjuntos que não necessariamente têm o mesmo número de instâncias, para esta análise, considere que os conjuntos rotulados tenham em média N instâncias e os conjuntos não rotulados M. De maneira análoga, considere N_c o número de componentes do grafo ótimo, obtido a partir de um conjunto rotulado X_t.

A primeira parte corresponde à construção do grafo ótimo, sendo esta idêntica à análise feita na Seção 3.3, uma vez que o algoritmo a construção dos grafos ótimos é realizada pelo algoritmo KAOG. Dessa forma, a construção de cada grafo ótimo tem ordem de complexidade de $O(N^2)$, lembrando que o quadrado, neste caso, deve-se ao cálculo da matriz de distância. Vale ressaltar que o grafo principal não é construído diretamente, mas sim pela adição dos componentes formados nos grafo ótimos e, o processo de adição de componentes no grafo principal consome N_c iterações e é da ordem de $O(N_c)$. Portanto, a atualização do grafo principal tem complexidade de $O(N^2)$, devido a construção do grafo ótimo.

Considere agora obter a ordem de complexidade do processo de classificação para o classificador incremental proposto. Como visto, no Algoritmo 4.2, os vizinhos para o novo vértice são procurados em duas etapas, primeiro considera-se os centros dos componentes e, para cada componente considerado, determina-se os vértices vizinhos. Este artifício torna o processo computacionalmente mais eficiente porque considera o número de componentes no grafo principal, N_{cp}, ao invés do número de vértices N_{Xp}, que por sua vez é muito maior que o número de componentes $N_{Xp} >> N_{cp}$. Dessa forma, além do cálculo da distância entre o novo vértice e todo centro de componente, que é feita em N_{cp} iterações, também é necessário ordenar as distâncias e calcular as distâncias para os vértices de cada componente considerado. Considere encontrar os componentes mais próximos, um a um, à medida que for necessário; utilizando o métodos de ordenação por seleção (*selection sort*). Seja $\hat{N}_{v_z}$ o número de componentes considerados como vizinhos até que o critério de parada seja satisfeito e, note que $\hat{N}_{v_z} << N_{cp}$. Desse modo, pode-se encontrar os componentes em $\hat{N}_{v_z} N_{cp}$ iterações, e o cálculo das distâncias para encontrar os vizinhos mais próximos dentre os vértices pertences aos componentes considerados em $\hat{N}_{v_z} N_{vp}$ iterações, onde N_{vp} corresponde ao número médio de vértices por componentes no

grafo principal. O que significa que a classificação pode ser realizada em $N_{cp} + \hat{N}_{v_z} N_{cp} + \hat{N}_{v_z} N_{vp}$ iterações; como $N_{cp} >> \hat{N}_{v_z}$ e N_{vp} é pequeno, a complexidade depende do número de componentes no grafo, portanto tem ordem de complexidade $O(N_{cp})$.

A terceira e última parte, a poda e remoção por não utilização, refere-se a retirada de componentes do grafo. A retirada de componentes por não utilização pode ser feita simplesmente verificando o parâmetro de tempo de cada componente, portanto é da ordem de $O(N_{cp})$. A poda consiste em classificar um conjunto rotulado, de forma que, se o conjunto possui N instâncias, a ordem de complexidade corresponde à classificação destas instâncias, que ocorre em $N_L N_{cp}$ iterações. Considerando que $N << N_{cp}$, pode-se dizer que a ordem da poda também corresponde à $O(N_{cp})$.

Portanto, a complexidade do classificador incremental proposto depende do tamanho dos conjuntos de dados rotulados de forma quadrática, devido ao cálculo da matriz de distância, e do número de componentes no grafo principal de forma linear tanto para a classificação e remoção de componentes. O número de componentes no grafo principal, por sua vez, depende do parâmetro de esquecimento τ e da distribuição dos dados. Note que, como na Seção 3.3, a complexidade para o classificador incremental foi derivada considerando o uso de métodos clássicos. No entanto, assim como na versão estática, a versão incremental também se beneficia de métodos avançados, por exemplo, no cálculo das distâncias ou para encontrar os componentes mais próximos.

4.3 *Classificação incremental* vs. *estática em dados com mudança de conceitos*

Como visto no Capítulo 3, o classificador estático apresenta desempenho comparável à classificadores bem conhecidos e do estado da arte. No entanto, um classificador estático é designado a classificar exemplos de dados pertencentes à distribuição dos dados do conjunto em que foi treinado. Algoritmos de aprendizado estático não apresentam mecanismos para reconhecer e reagir à mudanças de conceito. Com o objetivo de analisar o comportamento de um classificador estático em um ambiente dinâmico e compará-lo à abordagem incremental, considere os seguintes experimentos com os classificadores propostos KAOG e KAOGINC. Sejam dois domínios de dados artificiais que apresentam mudanças de conceito, o primeiro com mudança de conceito gradual e o segundo com mudanças de conceito podendo variar entre gradual e abrupta.

Para o primeiro experimento, considere que em ambos os casos o classificador foi treinado inicialmente com uma instância do subconjunto apresentado na Figura 4.2(a). O restante do conjunto original foi dividido em 7 grupos de dados, representando uma mudança de conceito gradual (ver Figuras 4.2(b)-(h)) e apresentados sequencialmente aos classificadores. Um classificador estático, assim como o KAOG, presume que o conceito aprendido no treinamento original não será alterado e não dispõe de mecanismos para processar dados rotulados após a fase de treinamento. Por essa razão, somente os conjuntos

de teste são apresentados após o treinamento. Já o classificador incremental KAOGINC tem habilidade para continuar aprendendo depois do treinamento inicial, durante a fase de uso do classificador. Dessa forma, o classificador pode processar exemplos rotulados, que são apresentados ao longo do tempo.

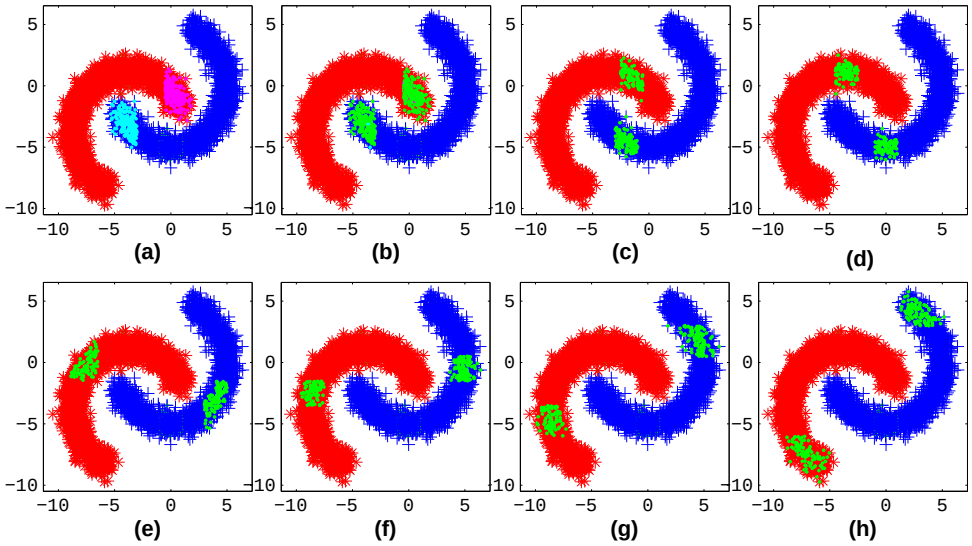

Figura 4.2: Domínio artificial dividido sequencialmente em grupos, simulando um domínio não estacionário com mudança de conceito gradual. Os padrões em evidência representam o subconjunto corrente.

A Figura 4.3 mostra o resultado da comparação para o domínio não estacionário, ilustrado na Figura 4.2, entre as abordagens de aprendizado estática e incremental do classificador baseado no grafo K-associado ótimo, respectivamente KAOG e KAOGINC. Os resultados são a média de 10 execuções, para cada execução um grafo ótimo foi construído considerando 400 instâncias de treinamento (200 de cada classe) gerados aleatoriamente dentro da distribuição do subconjunto da Figura 4.2(a). Os conjuntos de teste possuem 200 instâncias não rotuladas (100 para cada classe), também gerados aleatoriamente de acordo com a distribuição do subconjunto atual. O classificador incremental também recebe antes de cada conjunto de teste um conjunto rotulado com 100 instâncias rotuladas (50 cada classe).

A diferença significativa no desempenho entre as abordagens, como mostra a Figura 4.3, deve-se ao fato de que o classificador estático, uma vez treinado, não ter a capacidade de adquirir conhecimento, ao passo que o classificador incremental adquire conhecimento, continuamente - ao longo do processo de classificação. Para o experimento, a porcentagem de acertos foi calculada para cada dado de teste de forma cumulativa, ou seja, atualizada para cada novo resultado.

Note na Figura 4.3, que os primeiros 200 exemplos ambos classificadores mantém um desempenho perfeito, acertando todas os exemplos de teste que pertencem ao conceito aprendido. Já na primeira troca de conceito o desempenho do classificador estático começa a cair a uma determinada taxa de erro. Logo adiante, quando ocorre a segunda troca de conceito, o desempenho do classificador estático piora, aumentando a taxa de erro. Note,

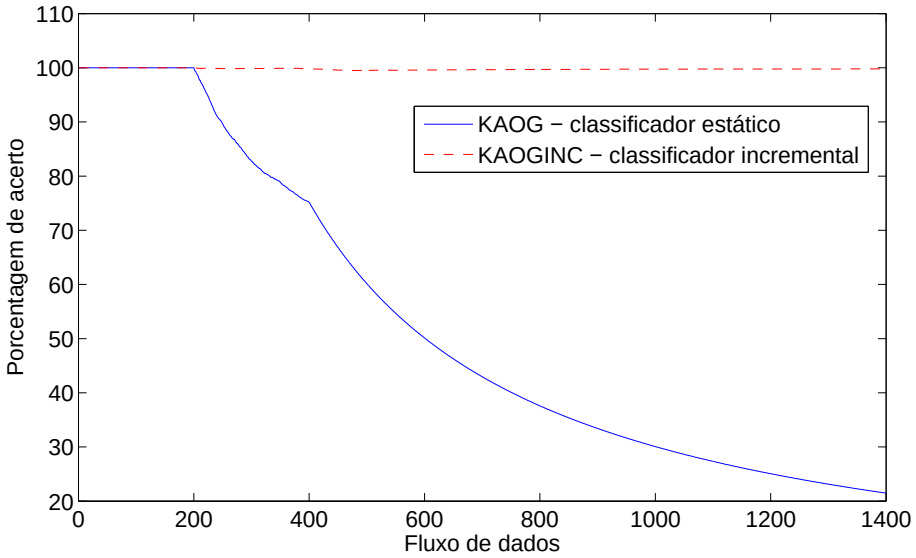

Figura 4.3: Resuldado de classificação utilizando os algoritmos KAOG e KAOGINC para o domínio artificial apresentado na Figura 4.2.

no entanto, que a curva de desempenho do classificador estático apresenta somente esses dois pontos de inflexão, relativos a primeira e segunda troca de conceito. O motivo pelo qual a taxa de erro não sofre alterações após a segunda troca de conceito é que nesta ocasião o classificador estático já não acerta mais nenhum exemplo de teste e a porcentagem de acertos é deteriorada. Esta queda brusca no desempenho do classificador estático pode ser facilmente entendida por meio da Figura 4.2. Sejam os subconjuntos apresentados na Figura 4.2(a), o conceito aprendido pelo classificador estático e, também o primeiro conceito de teste, neste caso os exemplos de teste não apresentam dúvida ao classificador. Já na sequência do experimento, na Figura 4.2(b), os conceitos de teste de ambas as classes apresenta uma pequena mudança, com a tendência de uma dirigir-se ao encontro da outra. Nesta situação, considerando a proximidade entre o conjunto de teste e o conjunto utilizado no treinamento, pode-se induzir que parte do conjunto de teste será classificada corretamente, por estar mais próxima ao conjunto correto. De maneira análoga, alguns erros deverão ocorrer, pelo fato de que o lado de cada conjunto de teste oposto ao conceito correto encontra-se mais próximo ao conceito que representa a classe oposta. Ainda seguindo esta ideia, nota-se que a partir do conceito estabelecido na Figura 4.2(c), os conjuntos de teste de determinada classe começam a dar a volta na outra classe, o que faz com que os dados de teste fiquem sempre mais próximos aos dados de treino da classe contraria. De forma que, o classificador estático treinado com o conceito inicial (Figura 4.2(a)) torna-se obsoleto, pois os dados pertencem a novos conceitos que não foram aprendidos pelo classificador estático.

Para o próximo experimento, considere o domínio artificial estabelecido como referência no estudo de mudança de conceito, proposto por Street e Kim (2001) e nomeado em alusão a este trabalho, como conceitos SEA (*SEA concepts*). Este domínio consiste na geração de um conjunto de dados, cada qual formado por três atributos, todos com valores gerados aleatoriamente no intervalo $[0, 10]$. O domínio, por definição, possui duas classes

que são atribuídas com base nos atributos x_1 e x_2. Se determinada instância apresenta a soma dos atributos x_1 e x_2 menor, ou igual, que um determinado limiar θ, $x_1 + x_2 \leq \theta$, então a este é atribuído a classe ω_1, caso a inequação não seja satisfeita atribui-se a classe ω_2. Para simular a mudança de conceito ao longo do tempo, considere o conjunto completo formado por 70.000 instâncias divididas em 4 subconjuntos com 17.500 instâncias cada, sendo que, no experimento 10.000 instâncias foram usadas como treino e 7.500 como teste. O que define cada subconjunto é justamente o limiar θ, que age como o viés na definição do hiperplano que divide as duas classes e tem o efeito de simular a mudança de conceito abrupta, quando este é alterado repentinamente.

Nos experimentos, o conjunto que exprime os conceitos SEA é dividido em 500 subconjuntos com 140 instâncias, no qual 80 são de treino e 60 de teste[1]. No experimento, o classificador incremental inicia treinando com o primeiro conjunto de treinamento de 80 instâncias. Já o classificador estático é treinado com um conjunto de 700 instâncias, o que equivale à quinta iteração do classificador incremental, considerando treino e teste. No que segue o conceito SEA é apresentado em conjuntos de treino com 80 instâncias e conjuntos de teste com 60 instâncias. O algoritmo incremental recebe os conjuntos de treino e teste intercaladamente, ao passo que o algoritmo estático KAOG recebe somente os conjuntos de teste.

A Figura 4.4 apresenta o resultado da comparação entre os algoritmos KAOG e KAOGINC no domínio SEA. Os conceitos foram gerados de acordo com os seguintes limiares, Figura 4.4(a) $\theta = \{7; 9; 6; 9, 5\}$, Figura 4.4(b) com $\theta = \{8; 6; 8; 7, 5\}$, Figura 4.4(c) $\theta = \{3; 5; 7; 9\}$ e Figura 4.4(d) com $\theta = \{9, 5; 8; 6; 4\}$. O resultado compreende a média de 20 execuções para cada algoritmo, considerando diferentes gerações do domínio SEA. Cada um dos 500 resultados, em cada experimento, compreende a média da porcentagem de erro nos conjuntos de teste.

Como esperado, nos quatro experimentos apresentados nas Figuras 4.4(a)-(d), o classificador estático apresentou bons resultados enquanto os exemplos de teste apresentados pertenciam ao conceito aprendido. Em outras palavras, até que ocorra a primeira troca de conceito, o problema apresentado corresponde a um problema estático e, dessa forma, é tratado naturalmente por um classificador estático. Por essa razão o desempenho de ambos os classificadores no primeiro conceito são semelhantes - inclusive na variação dos resultados. Há, no entanto, uma pequena vantagem por parte do classificador incremental sobre o estático em alguns pontos no primeiro conceito dos quatro experimentos. Esta vantagem deve-se ao fato de que o conjunto de treinamento usado pelo classificador estático, apesar de representar o conceito em questão, pode não abranger todo o espaço de atributos do conceito. O que não ocorre com o classificador incremental por consequência de sua constante atualização.

Ainda considerando os experimentos da Figura 4.4, verifica-se que toda vez que ocorre a primeira mudança de conceito, o classificador estático apresenta maior taxa de erros

[1] Esta proporção é mantida para todos os experimentos que envolvem o conceito SEA.

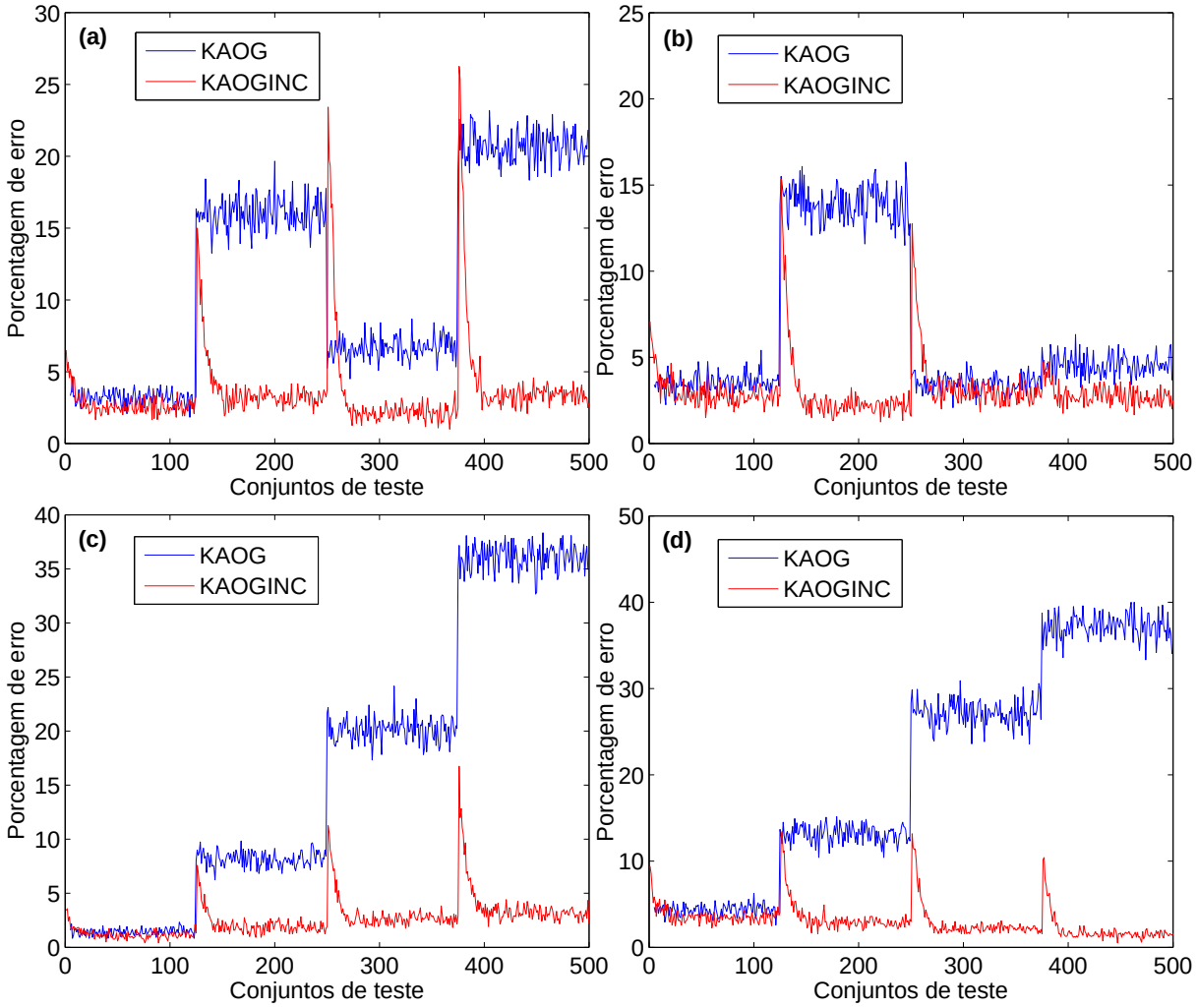

Figura 4.4: Comparações de desempenho entre o algoritmo estático KAOG e sua extenção incremental KAOGINC, considerando o domínio SEA gerado considerando diferentes conjuntos de limiares, a saber, (a) $\theta = \{7; 9; 6; 9, 5\}$; (b) $\theta = \{8; 6; 8; 7, 5\}$; (c) $\theta = \{3; 5; 7; 9\}$; (d) $\theta = \{9, 5; 8; 6; 4\}$ com o objetivo de simular diferentes mudanças de concetios.

e maior variação nos resultados, sendo esta variação, consequência da instabilidade do classificador ao lidar com um novo conceito. Por outro lado, o classificador incremental reage à mudança de conceito rapidamente. Nota-se que ocorre um pico na curva de erros no instante em que ocorre a mudança de conceito, mas passadas algumas iterações, o novo conceito é aprendido e a taxa de erro volta a níveis aceitáveis. Também é possível perceber que quanto mais abrupto a mudança de conceito maior o pico na taxa de erros. Nos casos estudados, a mudança mais abrupta ocorre na Figura 4.4(a) na terceira troca de conceitos[2], seguido pela segunda, o que implica nos maiores picos.

A Figura 4.4(b) mostra uma situação de mudança de conceito recorrente, em que um conceito passado volta a ocorrer. No caso em questão, o primeiro e o terceiro conceito são idênticos, portanto o classificador estático apresenta bom desempenho em ambos, uma vez que foi treinado com instâncias de dados do primeiro conceito. Já a mudança que separa o terceiro e quarto conceitos, é bem suave, mudando pouco os dados do quarto conceito -

[2] Para o do domínio SEA quanto maior a diferença entre os limiares, mais abrupta a mudança.

o que explica o pequeno aumento na taxa de erro por parte do classificador estático. As Figuras 4.4(c) e (d) mostram um cenário de mudança de conceito gradual, onde o limiar cresce ou decresce, respectivamente, de maneira gradual afastando o hiperplano corrente cada vez mais do hiperplano referente ao conceito original. Por essa razão a taxa de erros do classificador estático aumenta a cada vez que o novo conceito se afasta do conceito aprendido. Considerando agora o classificador incremental, repare que na Figura 4.4(c), a taxa de erro tem início por volta de 1 a 2% e tende a crescer ao longo do tempo até chegar a 3 a 4%. Situação oposta à apresentada na Figura 4.4(d), onde a taxa de erro tem início em 4 ou 5% e termina em 1 ou 2%. Isto se deve ao fato de como os limiares influenciam no conjunto gerado. Como as classes são definidas de acordo com a soma de dois atributos e, como esses atributos variam no intervalo $[0, 10]$, a situação em que se espera igual probabilidade para ambas as classes, ocorre quando o limiar está próximo de 10. Assim, limiares pequenos favorecem o crescimento da classe ω_1 sobre a classe ω_2, o que interfere na taxa de erros do classificador. Este fato também pode ser verificado nos resultados do classificador estático que apresenta maior taxa de erros na Figura 4.4(d) devido ao problema no balanceamento das classes citado anteriormente.

Em resumo, classificadores estáticos não são desenvolvidos para classificar dados que possam sofrer mudanças de conceito. Esses classificadores assumem que o conceito, ou a distribuição dos dados em questão não será alterado. O que limita a atuação de classificadores estáticos a conjuntos de dados estacionários ou, eventualmente, a algum problema não estacionário muito específico, no qual vale a premissa de que mudanças de conceito não ocorram. Em uma abordagem incremental, no entanto, o sistema classificador é definido de maneira que este possa identificar mudanças no conceito e reagir a isso. Como mostrado nas Figuras 4.3 e 4.4, a abordagem incremental detectou todas as mudanças de conceito e foi capaz de recuperar o desempenho de classificação prontamente. Cabe ressaltar que nesta seção foram comparadas duas abordagens diferentes de aprendizado com o objetivo não de estabelecer o melhor, mas sim de ilustrar os problemas que um classificador estático apresenta em um contexto dinâmico. A Seção 4.5 compara e analisa o algoritmo KAOGINC com outros algoritmos incrementais, a fim de estabelecer a real contribuição da abordagem proposta.

4.4 Análise de parâmetros para o algoritmo KAOGINC

Como o classificador incremental KAOGINC é paramétrico, nesta seção será exposta uma breve análise da variação de seu parâmetro, τ. A seção também apresenta a análise do efeito do uso da poda nos desempenho do classificador, aliado à regra de não utilização. Considere, inicialmente, a análise da variação do parâmetro τ, que define o número máximo de classificações em que cada componente pode permanecer no grafo sem ser usado. O experimento ilustra duas implementações para cada valor de τ, uma em que a poda não é utilizada, que visa evidenciar o comportamento do parâmetro τ. E outra

combinado a poda com a regra de não utilização, com o objetivo de verificar o comportamento da poda para diferentes valores de τ. Para cobrir um intervalo razoável de valores, considere o parâmetro τ variando no intervalo $[1, 100]$ e assumindo valores intervalados de 5, $\tau \in \{1, 5, 10, \ldots, 100\}$, resultando em 21 modelos de classificadores. Para os experimentos foi usado o domínio SEA (Street e Kim, 2001), como descrito na Seção 4.3, com os mesmos conjuntos de limiares usados nos experimentos reportados na Figura 4.4, a saber, (a) $\theta = \{7; 9; 6; 9, 5\}$, (b) $\theta = \{8; 6; 8; 7, 5\}$, (c) $\theta = \{3; 5; 7; 9\}$ e (d) $\theta = \{9, 5; 8; 6; 4\}$. O algoritmo KAOGINC foi executado 20 vezes para cada valor do parâmetro τ, com diferentes instâncias do domínio SEA geradas a cada execução. Portanto, para cada valor de τ obteve-se a média do erro, corresponde a 20 execuções, para cada conjunto de teste ao longo do domínio SEA. A Figura 4.5 compreende à média do erro dos conjuntos de testes referentes a cada um dos conjuntos citados.

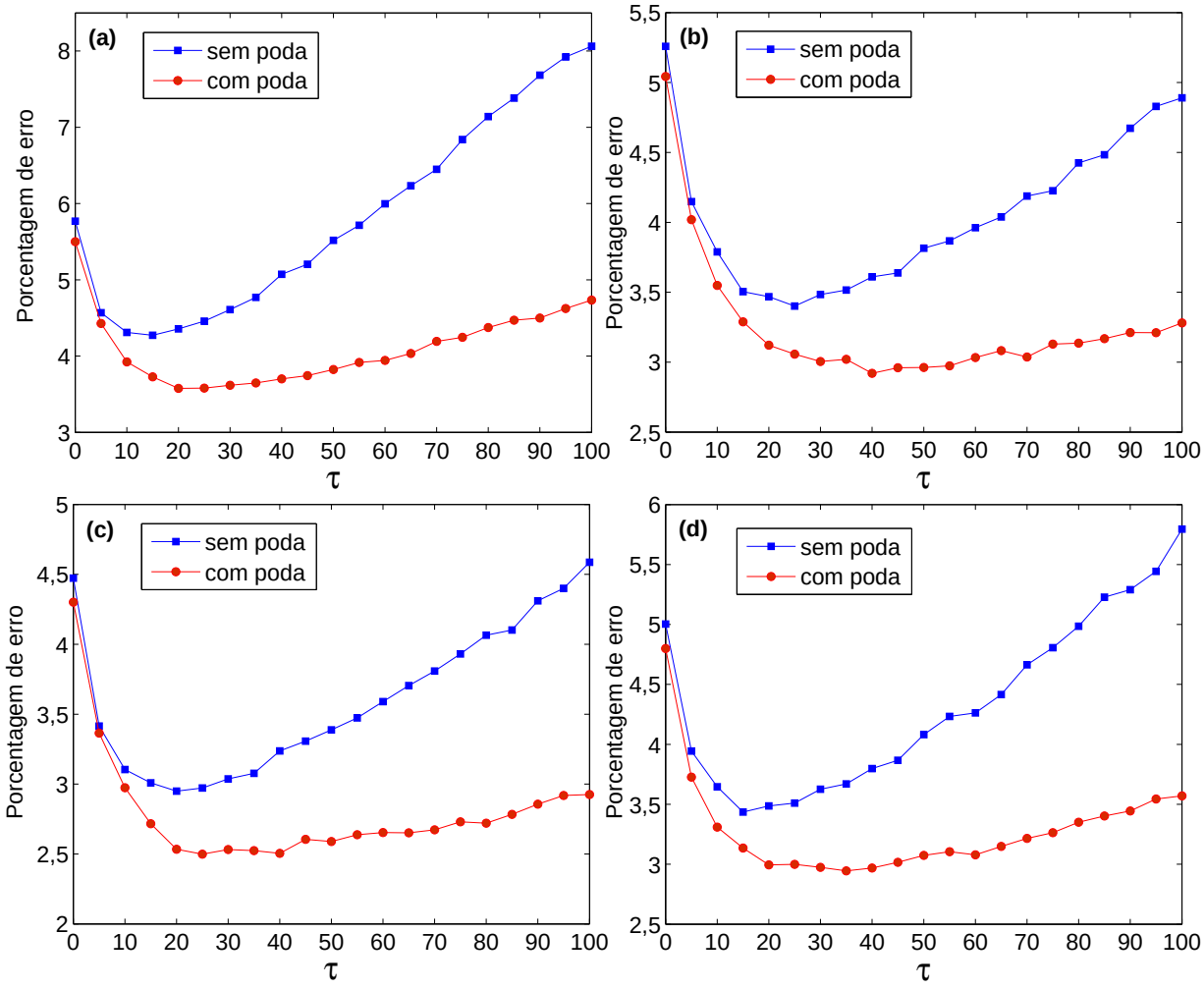

Figura 4.5: Média do erro para diferentes valores do parâmetro τ considerando o domínio SEA gerado com diferentes conjuntos de limiares, a saber (a) $\theta = \{7; 9; 6; 9, 5\}$; (b) $\theta = \{8; 6; 8; 7, 5\}$; (c) $\theta = \{3; 5; 7; 9\}$; (d) $\theta = \{9, 5; 8; 6; 4\}$.

A Figura 4.5 compreende quatro versões do domínio SEA, ligeiramente diferentes uma das outras. Note que, ainda assim, os valores para o parâmetro τ, que resultaram nos melhores desempenhos, foram diferentes para cada domínio. Isso mostra que mesmo para domínios relativamente parecidos, o valor de τ pode não ser o mesmo, sendo que para

cada domínio existe um intervalo de valores de τ que proporciona o melhor desempenho do algoritmo. Por exemplo, considerando os modelos dos classificadores que não utilizam a poda, na Figura 4.5(a), um bom intervalo para a escolha de τ é $[8, 15]$; já na Figura 4.5(b), o parâmetro τ pode ser escolhido no intervalo $[15, 30]$. De maneira similar, bons intervalos de valores para o parâmetro τ mostradas nas simulações das Figuras 4.5(c) e (d) seriam, $[15, 25]$ e $[12, 25]$, respectivamente. De modo que, qualquer valor escolhido nestes intervalos produzirão bons classificadores, uma vez que a curva de erros mostra a tendência no aumento do erro em relação à variação do parâmetro e não corresponde em determinar um único valor ótimo para este.

Considerando agora o uso da poda de forma concomitante à regra de não utilização. Note que com o uso da poda, o algoritmo é alterado, o que torna necessário selecionar novamente o modelo, fato que também explica as diferenças nos desempenhos dos algoritmos. Considere a seleção de modelo feita anteriormente, mas agora considerando classificadores que programem a eliminação de componentes que cometem erro. Pode-se dizer que, como esperado, quanto maior o valor de τ maior o efeito da poda, uma vez que o aumento de τ aumenta o número de componentes no grafo. De maneira geral, nota-se que a poda tende a diminuir a taxa de erro, bem como a influência do parâmetro τ, pois diminui a diferença na taxa de entre valores próximos de τ. Outro ponto interessante é que o uso da poda, nos quatro domínios, não só aumentou o tamanho dos intervalos de valores de τ que produzem bons resultados, mas também a magnitude dos valores em si, por exemplo, na Figura 4.5(a) passou de $[8, 15]$ sem o uso da poda, para $[15, 30]$, com o uso da poda, e assim também para os demais domínios.Vale ressaltar que, apesar da poda ter resultado em melhora para todos os modelos testados neste experimentos, pode acontecer que, para algum domínio o uso da poda resulte em piora no desempenho, de maneira que, esta pode ser retirada da implementação do algoritmo.

Considere agora verificar as diferenças no desempenho ao longo da apresentação do domínio, entre um valor de τ, pertencente ao intervalo de bons valores, obtidos anteriormente, e dois outros valores tomados ao acaso. Para evidenciar o comportamento do parâmetro τ considere o algoritmo sem o uso da poda. Para este experimento considere, a variante do domínio SEA como gerado na Figura 4.5(c), com os seguintes limiares $\theta = \{3, 5, 7, 9\}$, e os valores $\tau = 20$, como o valor escolhido dentro do intervalo estimado, e os valores $\tau = 45$ e $\tau = 65$, tomados ao acaso fora do intervalo em questão. Os resultados como mostrado na Figura 4.6.

Note que, o aumento de τ, invariavelmente, aumenta o tempo de recuperação do classificador nas mudanças de conceito. Quanto maior o valor de τ mais tempo um componente permanece no grafo sem ser usado, o que por um lado aumenta a amostragem de dados em conceitos estáticos, o que, até certo ponto, pode melhorar o desempenho do classificador. Por outro lado, quando ocorre uma mudança de conceito, os componentes obtidos em conceitos anteriores permanecerão no mínimo τ iterações no novo conceito, o que geralmente atrapalha a classificação de novos dados. Valores pequenos para τ, apesar de tornar

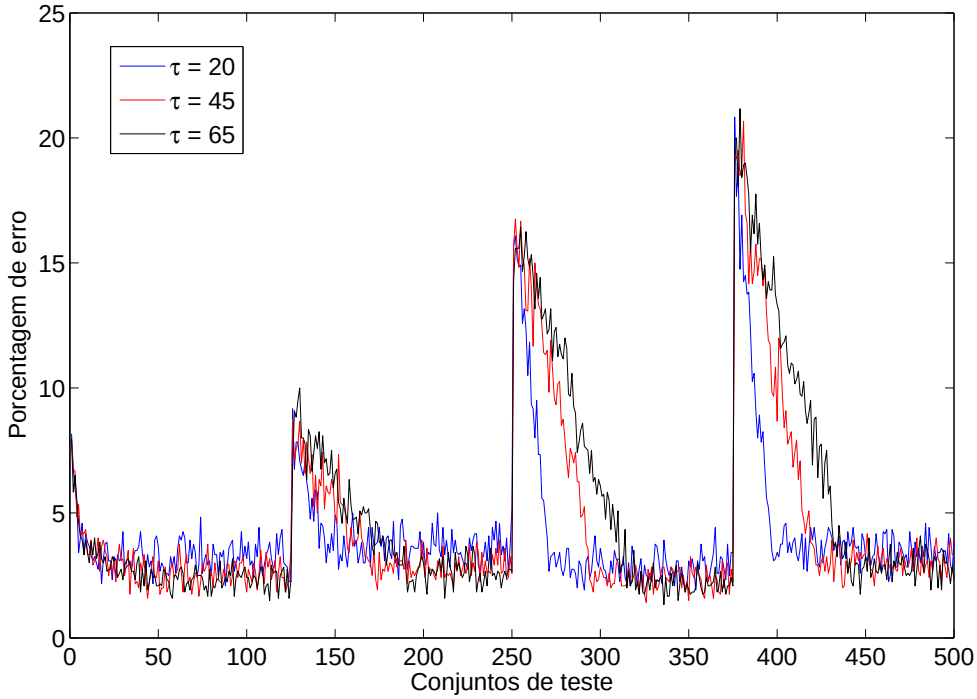

Figura 4.6: Comparação de desempenho entre os modelos do classificador KAOGINC referentes aos parâmetros $\tau = 20, \tau = 45, \tau = 65$ no domínio SEA.

o classificador mais flexível, possibilitando-o recuperar-se rapidamente em mudanças de conceito, também o torna menos abrangente em conceitos estáticos, fazendo com que seu desempenho piore. Os efeitos que a variação de τ provoca no classificador sugere que, o desempenho em conceito estático e a recuperação de desempenho nas mudanças de conceitos, são situações conflitantes. O que obriga o usuário a ponderar entre bom desempenho em conceito estático e rápida recuperação nas mudanças de conceitos. Essa ponderação pode ser feita com sucesso se houver algum conhecimento prévio, por exemplo, se o usuário, de alguma forma, sabe que não ocorrerá mudanças de conceito ou que ocorrerão poucas alterações, então τ pode ser ajustado de modo a otimizar o desempenho nos conceitos estáticos. Já, o caso oposto acontece quando se sabe que o domínio apresenta muitas mudanças de conceito, neste caso opta-se por τ com valores menores para facilitar a renovação de componentes e a recuperação rápida de desempenho. No entanto, considerando aplicações apresentadas em fluxo de dados, fica difícil obter conhecimento prévio sobre o acontecerá no futuro. Além disso, o que se deseja de um classificador incremental é que este mantenha bom desempenho em áreas de conceito estático e recupere-se com rapidez nas mudanças de conceitos. Felizmente, é possível aperfeiçoar o classificador proposto KAOGINC otimizando seu desempenho tanto em conceitos estático como na mudança de conceito, por meio da implementação da poda. Como o aumento no tempo de recuperação do classificador está relacionado a erros provenientes do uso de componentes de conceito(s) anterior(es) na classificação de novos dados; a solução é retirar do grafo todo componente que está envolvido em erro de classificação de exemplos do conjunto de teste. Como a escassez de dados não é problema em aplicações de classificação em fluxo de dados, a eliminação de vértices do grafo não resulta em perda de informação,

mas sim na melhora do classificador. A Figura 4.7 ilustra como o aumento nos valores de τ influencia negativamente a retomada do desempenho quando ocorre a mudança de conceito para o classificador que não usa poda, comparado com a versão do classificador que implementa a poda.

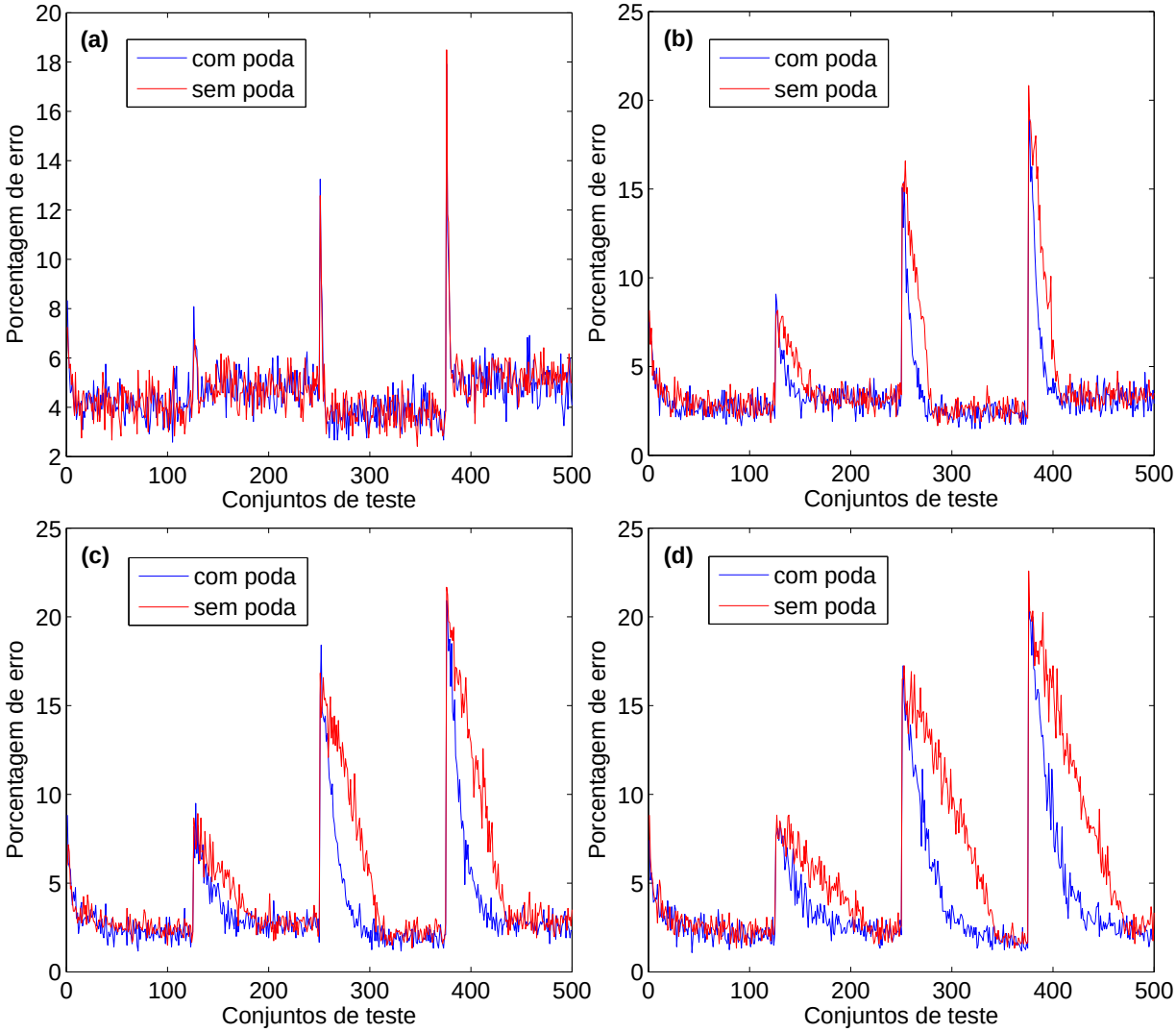

Figura 4.7: Comparação das taxas de erro considerando diferentes modelos do algoritmo KAOGINC quanto à permanência ou retirada dos componentes associados a erros de classificação. Na Figura os modelos foram gerados com, (a) $\tau = 5$, (b) $\tau = 30$, (c) $\tau = 60$ e (d) $\tau = 95$.

A poda tem o objetivo de eliminar do grafo os componentes que cometeram erros, tais componentes podem ter sido formados por ruído nos conjuntos de treinamento ou representavam conceitos passados e ainda permanecem no grafo. Dessa forma, é possível melhorar a performance em conceito estático variando τ e obter resposta rápida à mudança de conceito devido à poda. Na Figura 4.7 o que se percebe de imediato é que quanto maior o valor de τ maior o efeito aparente da poda. Na Figura 4.7(a), praticamente não há vantagem no uso da poda, pois como τ tem valor pequeno, a substituição de componentes no grafo ocorre rapidamente e o uso da poda é irrelevante. Como já mencionado, acontece que esta troca constante de componentes no grafo resulta em perda de desempenho.

Na Figura 4.7(b) nota-se que a poda tem um efeito positivo em relação à retomada de desempenho na mudança de conceito. Nas três mudanças de conceito, o algoritmo KAOGINC com poda recuperou-se com maior rapidez e, ao mesmo tempo, manteve desempenho comparável ao algoritmo KAOGINC sem poda durante as fases estáticas dos conceitos. Nas Figuras 4.7(c) e (d) o algoritmo KAOGINC com poda continuou sendo melhor que a versão sem poda, no entanto nesses casos, τ assume valor muito grande para este domínio. Evidências disso são que, o aumento de τ comparado com a Figura 4.7(b), não resultou em melhora no desempenho em conceito estático e, mesmo com a implementação da poda, a retomada de desempenho é mais demorada se comparada à Figura 4.7(b). Como já explicado, a melhora em conceito estático ocorre até certo valor de τ. Já a diminuição na eficácia da poda tem relação com o tamanho do grafo, com o tamanho do conjunto usado para realizar a poda e com a porcentagem de erros do classificador, uma vez que a poda depende da classificação de dados ainda desconhecidos do classificador.

Como visto, enquanto τ aumenta também aumenta o tempo de ajuste em relação ao número de componentes no grafo, e que isto tem relação direta com o tamanho do grafo. Quanto maior o grafo, maior o tempo que será gasto para que o mesmo volte a equilibrar-se em relação ao número de componentes, após uma mudança de conceito, como mostra a Figura 4.8. Isso se deve ao fato de que, mesmo aumentando o valor de τ, se algum desses componentes envolverer-se em erro, o mesmo é retirado do grafo. Ao passo que, sem o uso da poda, quando um componente é usado, mesmo se em uma situação que resulte em erro, o mesmo tem seu parâmetro de tempo zerado. O que significa que este componente deverá esperar mais τ iterações para ser retirado do grafo. O que faz com que o erro aumente conforme τ aumenta, quando a poda não é considerada. Como esperado, o aumento no valor de τ implica no aumento no número médio de componentes no grafo. A Figura 4.8 ilustra o número de componentes no grafo ao longo da apresentação do domínio SEA.

Note na Figura 4.8(a) que o uso da poda não resulta em grande vantagem com relação ao tamanho do grafo, de fato, a poda mantém o grafo um pouco menor em relação à versão que não usa poda, durante os conceitos estáticos. No entanto, essa pequena diferença não resulta em ganho no desempenho do classificador. Note que não existe diferença entre as versões durante as mudanças de conceito, devido ao pequeno valor de τ. Já na Figura 4.8(b), a diferença no número de componentes é maior em áreas de conceito estático e tende a aumentar quando ocorre mudanças de conceitos. Quando ocorre a mudança de conceito, o algoritmo que implementa a poda, livra-se rapidamente de componentes de conceito antigo, o que possibilita ao classificador retomar rapidamente o desempenho. De maneira análoga ao que ocorre com a taxa de erro, a variação no número de componentes também é afetada pelo tamanho do grafo, pois quanto maior o grafo mais tempo levará para os componentes de conceitos passados serem substituídos por componentes do conceito atual. Como mostra as Figuras 4.8(c) e 4.8(d), o aumento de τ causa o aumento do grafo.

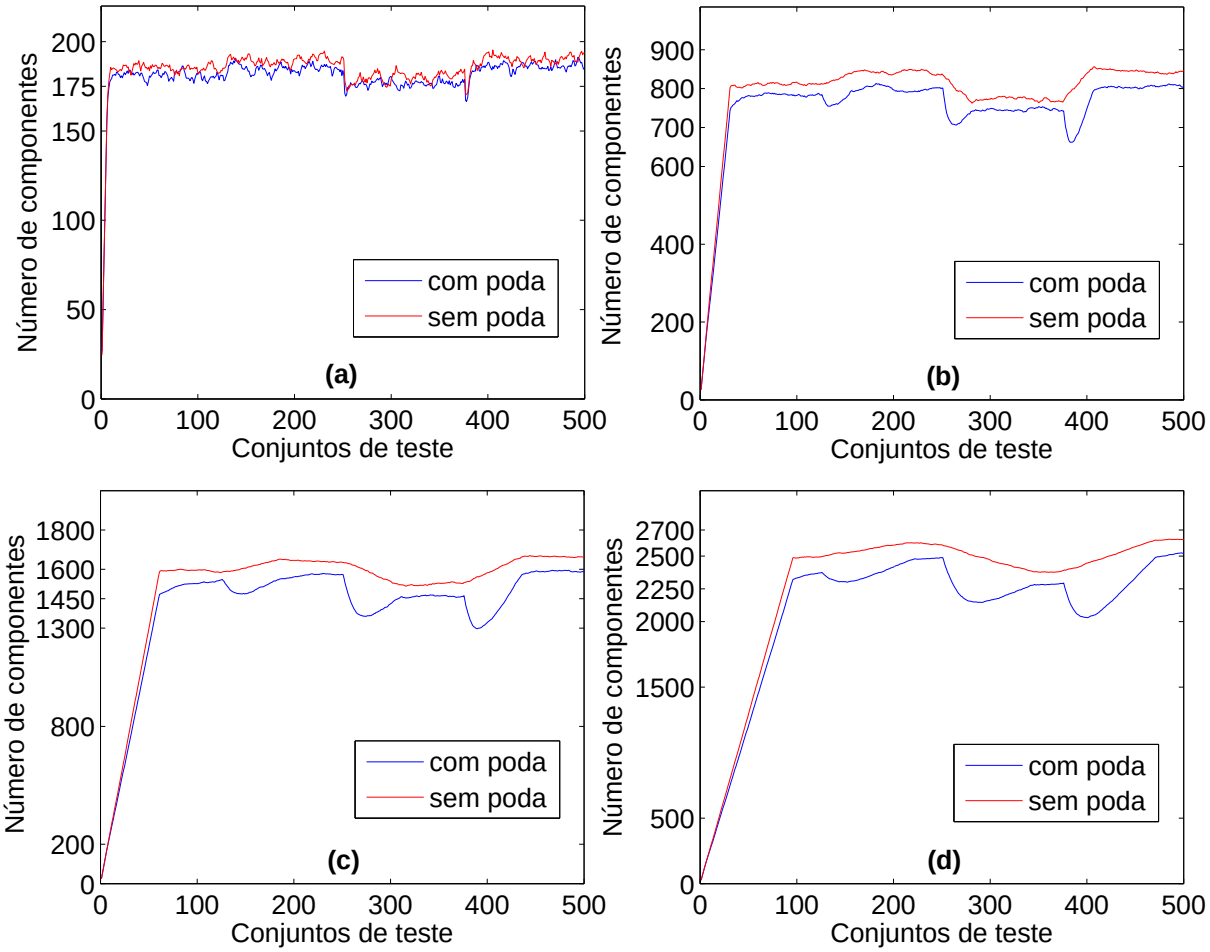

Figura 4.8: Comparação entre modelos em relação ao número de componentes considerando as diferentes versões do algoritmo quanto à permanência ou retirada dos componentes associados a erros de classificação. A Figura (a) corresponde a modelos gerado com $\tau = 5$, a (b) $\tau = 30$, (c) $\tau = 60$ e (d) $\tau = 95$.

Independente do uso da poda note que o tamanho do grafo varia de acordo com o conceito corrente, isso mostra a capacidade do algoritmo em ajustar-se à distribuição dos dados. De fato o número de vértices tende a permanecer constante ao longo do tempo, o que varia é a estruturação dos vértices em componentes. Isto significa que os componentes são formados de modo a adequarem-se ao conceito corrente. Cada conceito representa uma distribuição diferente, e resulta em componentes com padrões de conexão, tamanho e purezas diferentes. Para ilustrar esse fato considere a Figura 4.9, onde a Figura 4.9(a) mostra como os componentes variam em tamanho de acordo com o conceito em questão; ao passo que na Figura 4.9(b) é mostrada a variação no número de vértices para as mesmas execuções.

Para evidenciar a diferença no tamanho dos componentes em resposta ao conceito corrente, os experimentos utilizaram mudanças de conceitos mais abruptas que nos experimentos anteriores, gerando o domínio SEA com os seguintes limiares $\theta = \{7; 9; 6; 9, 5\}$. Nos experimentos foi considerado $\tau = 25$, e os resultados são a média de 20 execuções.

Retirar todo componente que se associou a pelo menos um erro durante a fase de teste pode parecer uma condição muito severa. Imagine a situação onde um *outlier* ou

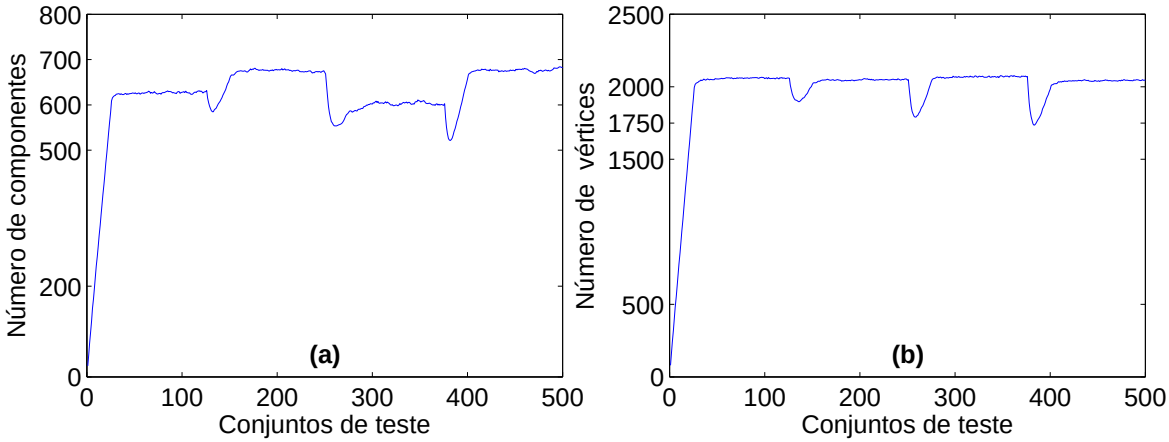

Figura 4.9: Relação entre número de componentes e número de vértices no grafo principal ao longo do processamento do domínio SEA. Para este experimento foi considerado os seguintes limiares $\theta = \{7; 9; 6; 9, 5\}$.

ruído pertencente a um conjunto utilizado para estimar o desempenho do classificador bem como realizar a poda. Este *outlier* ou ruído pode, eventualmente, localizar-se em alguma área bem definida do espaço de dados, representada por componentes de uma determinada classe, diferente à do dado ruidoso. Como resultado, no momento de classificação desta instância, esta situação ocasionará erros nos componentes usados na sua classificação e consequentemente em sua eliminação. No entanto, por mais que algumas vezes bons componentes sejam retirados do grafo injustamente, o desempenho do classificador não é afetado de maneira considerável. Isto porque a ocorrência de *outliers* ou ruídos é relativamente incomum e ocorrem em intervalos esparsos no espaço de dados. O que faz com que o desempenho do classificador não seja alterado.

Note que nos experimentos realizados neste trabalho, a poda foi realizada considerando os conjuntos de teste. Isto pressupõe que as classes das instâncias no conjunto de teste sejam conhecidas. Se o objetivo é avaliar o classificador ou mesmo compará-lo com outro classificador, esta suposição pode ser feita, e os conjuntos de teste são usados como validação. No entanto, em uma situação real, as classes dos dados que devem ser classificados não são disponíveis. Dessa forma, a poda para o classificador KAOGINC deve ser realizada utilizando o próximo conjunto de treino antes que os componentes provenientes deste conjunto sejam adicionados no grafo. Esta abordagem é frequentemente usada em algoritmos incrementais na literatura (Tsymbal *et al.*, 2006), (Scholz e Klinkenberg, 2007).

4.5 Resultados experimentais

Nesta seção são apresentados alguns resultados comparativos considerando o algoritmo incremental proposto, KAOGINC, contra os algoritmos apresentados na Seção 2.4, a saber, os algoritmos OnlineTree2, SEA, WCEA e DWM-NB, em diversos domínios artificiais e reais.

Nas seções subsequentes são apresentados os 10 domínios utilizados na comparação bem como uma breve análise dos resultados para cada um deles. Os resultados para os algoritmos usados como comparativos, são apresentados sobrepostos ao resultado do KAOGINC para efeito de comparação. Dentre os domínios considerados, seis deles são artificiais e bastante usados na literatura, ver, por exemplo, (Street e Kim, 2001),(Gama *et al.*, 2004),(Hulten *et al.*, 2001); os demais são domínios reais disponibilizados para uso público, a saber, Elec2, Poker Hand, Spambase e KDD'99, e podem ser encontrados nos repositórios (Asuncion e Newman, 2007) e (Hettich e Bay, 1999). Os domínios reais são apresentados em conjuntos intercalados rotulados e não rotulados, e o resultado para essas bases consiste na média de 20 execuções de cada algoritmo. Já os fluxos de dados para os domínios reais considera somente conjuntos rotulados e é da forma $S = \{X_1, X_2, \ldots, X_T\}$. De modo que, para estimar os erros dos algoritmos nesses domínios o conjunto X_t é usado como teste antes de ser apresentado como treino. Todas bases reais foram apresentadas uma única vez para os algoritmos, como em uma situação real.

A seleção de parâmetros para os algoritmos paramétricos (todos exceto o OnlineTree2) considerou a porcentagem média de acertos em dados de teste ao longo do fluxo de dados. Os modelos foram selecionados simplesmente escolhendo-se o de melhor resultado em cada domínio. Para cada algoritmo foi considerado os seguintes valores de parâmetros, KAOGINC, parâmetro de esquecimento $\tau \in \{1, 2, 3, \ldots, 70\}$; SEA, tamanho do comitê $\varepsilon \in \{3, 5, 7, \ldots, 71\}$; WCEA, tamanho de comitê $\varepsilon \in \{2, 3, 4, \ldots, 70\}$; DWM-NB, número de iterações para verificação do comitê $\rho \in \{1, 5, 10, \ldots, 70\}$. A última seção apresenta o teste estatístico realizado considerando as taxas de acertos ao longo de todo o fluxo de dados para todos os domínios testados, com o objetivo de definir a contribuição do algoritmo proposto.

4.5.1 Conceito SEA

Considere o domínio SEA proposto por Street e Kim (2001) e apresentado na Seção 4.3, para os limiares $\theta = \{8; 9; 7; 9, 5\}$, e que domínio é apresentado aos classificadores em conjunto de treino e teste, nesta ordem, com 80 e 60 instâncias cada conjunto, respectivamente. A Figura 4.10, ilustra a comparação entre os algoritmos vistos neste trabalho e o algoritmo proposto KAOGINC.

Para estes resultados a seleção de modelos para os algoritmos paramétricos resultou nos seguintes valores de parâmetro: KAOGINC com $\tau = 30$, SEA com tamanho de comitê de 25, WCEA também com 25 classificadores no comitê e DWM-NB com $\rho = 50$. Este domínio é, talvez, o mais considerado na validação de algoritmos de classificação incremental e de tempo real. Por essa razão, muitos resultados têm sido publicados, inclusive em alguns dos trabalhos cujos algoritmos são utilizados neste trabalho para efeito de comparação. Como é o caso do trabalho de Street e Kim (2001), que propôs o domínio, e também nos trabalhos que apresentaram os algoritmos DWM-NB (Kolter e Maloof, 2007)

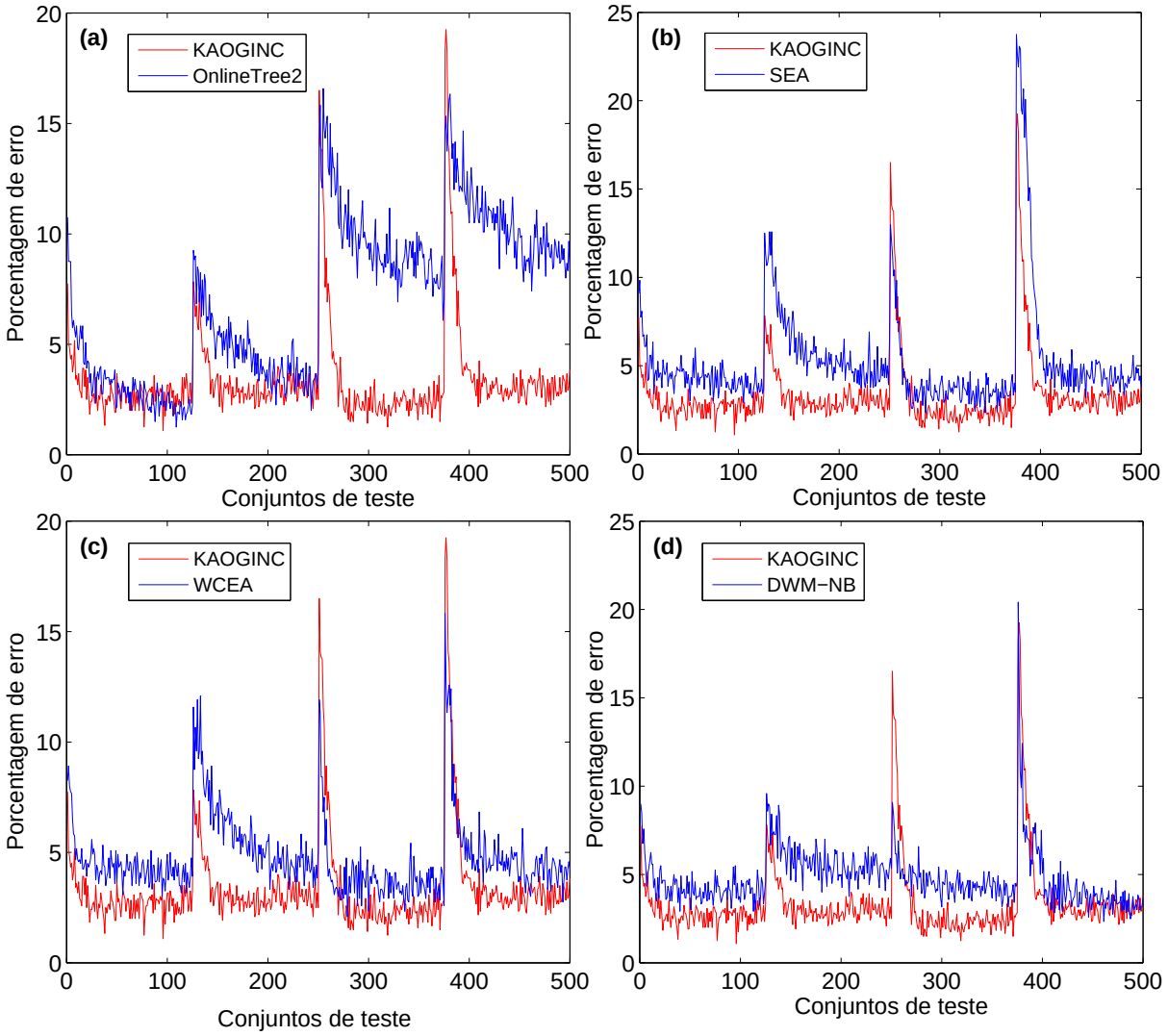

Figura 4.10: Comparações de desempenho entre os algoritmos KAOGINC e (a) OnlineTree2; (b) SEA; (c) WCEA e (d) DWM-NB para o domínio SEA. A seleção de modelos resultou nos seguintes parâmetros, KAOGINC $\tau = 30$; SEA $\varepsilon = 25$; WCEA $\varepsilon = 25$ e DWM-NB $\rho = 50$.

e OnlineTree2 (Núnez *et al.*, 2007). É importante ressaltar que cada autor considerou a apresentação do domínio de acordo com o problema que este se propôs a tratar e, por essa razão, pode ser que alguns resultados sejam diferentes dos apresentados neste trabalho. Um desses casos ocorre com o algoritmo OnlineTree2, que foi proposto originalmente para domínios de dados apresentados em tempo real. Na Figura 4.10(a), no entanto, nota-se que o algoritmo OnlineTree2 apresenta bom desempenho até a segunda mudança de conceito, mas depois disso, demora em retomar o desempenho após cada mudança de conceito. Isto se deve ao fato de que o algoritmo OnlineTree2 não reage bem a mudança de conceito abrupta, pois leva várias iterações para adaptar-se totalmente ao novo conceito. Já os algoritmos baseados em comitê têm grande capacidade de adaptação, uma vez que após cada mudança de conceito, o comitê pode restaurar-se com novos classificadores em um número de iterações equivalente à metade da capacidade do comitê, para o caso do algoritmo SEA, por exemplo. Os comitês com peso têm a capacidade de recuperarem-se mais

rapidamente, uma vez que podem ponderar a favor do classificador treinado no conceito mais novo, como é o caso dos algoritmos WCEA e DWM-NB. Mesmo assim, o melhor desempenho foi obtido pelo classificador proposto KAOGINC. Apesar de não se tratar de um comitê, o algoritmo KAOGINC tem grande capacidade de adaptação, pois além de incorporar conhecimento toda vez que um novo conjunto é apresentado, este desfaz-se de conhecimento antigo por meio do parâmetro de esquecimento, bem como livra-se de ruídos por meio da poda.

4.5.2 Hiperplano móvel

Este domínio simula um hiperplano que define duas classes e move-se ao longo do tempo. Neste problema, definido em (Hulten *et al.*, 2001) e (Fan, 2004), tipicamente, cada instância possui dimensão $p = 10$ e cada atributo é gerado aleatoriamente no intervalo $[0, 1]$. Considere o hiperplano definido na Equação (4.2), onde a_i são pesos inicializados aleatoriamente no intervalo $[-1, 1]$. O hiperplano separa o espaço de atributo em dois, atribuindo as classes, ω_1 caso a Equação (4.2) seja satisfeita e ω_2, caso contrário.

$$\sum_{i=1}^{p} a_i x_i \geq a_0 \tag{4.2}$$

O domínio possui dois tipos de mudança de conceito, gradual e abrupto. Para simular a mudança gradual, considere que q atributos ($q \leq p$) sejam constantemente alterados. Ou seja, Os pesos a_i, para $i = 1, \ldots, q$, são atualizados toda vez que um novo dado é gerado, de acordo com a Equação (4.3).

$$a_i = a_i + \alpha s_i \tag{4.3}$$

Isso faz com que o hiperplano se mova com velocidade dada por α ($\alpha = 0,0001$, nos experimentos) e $\mathbf{s} \in \{-1, 1\}$ indica a direção para a qual o hiperplano se move. Toda vez que os pesos são atualizados o limiar a_0 também é atualizado, conforme a Equação (4.4).

$$a_0 = \frac{1}{2} \sum_{i=1}^{p} a_i \tag{4.4}$$

Para simular o desvio de conceito abrupto, s é atualizado com probabilidade de 10% a cada 5.000 exemplos, o que, na prática faz com que o hiperplano mude de direção. Nas simulações foram usados um total de 40.000 exemplos. Os dados são apresentados aos classificadores, sequencialemente, em conjuntos de treino e teste, cada qual com 50 instâncias. A Figura 4.11, mostra as comparações entre o algoritmo proposto e os demais algoritmos.

Como mencionado, este domínio apresenta dois tipos de mudança de conceito, gradual e abrupta. O que exige que o classificador se adapte a diferentes situações ao longo do processamento. O algoritmo proposto obteve boa performance, ficando com o segundo melhor desempenho com relação à média nos conjuntos de teste. O algoritmo SEA obteve

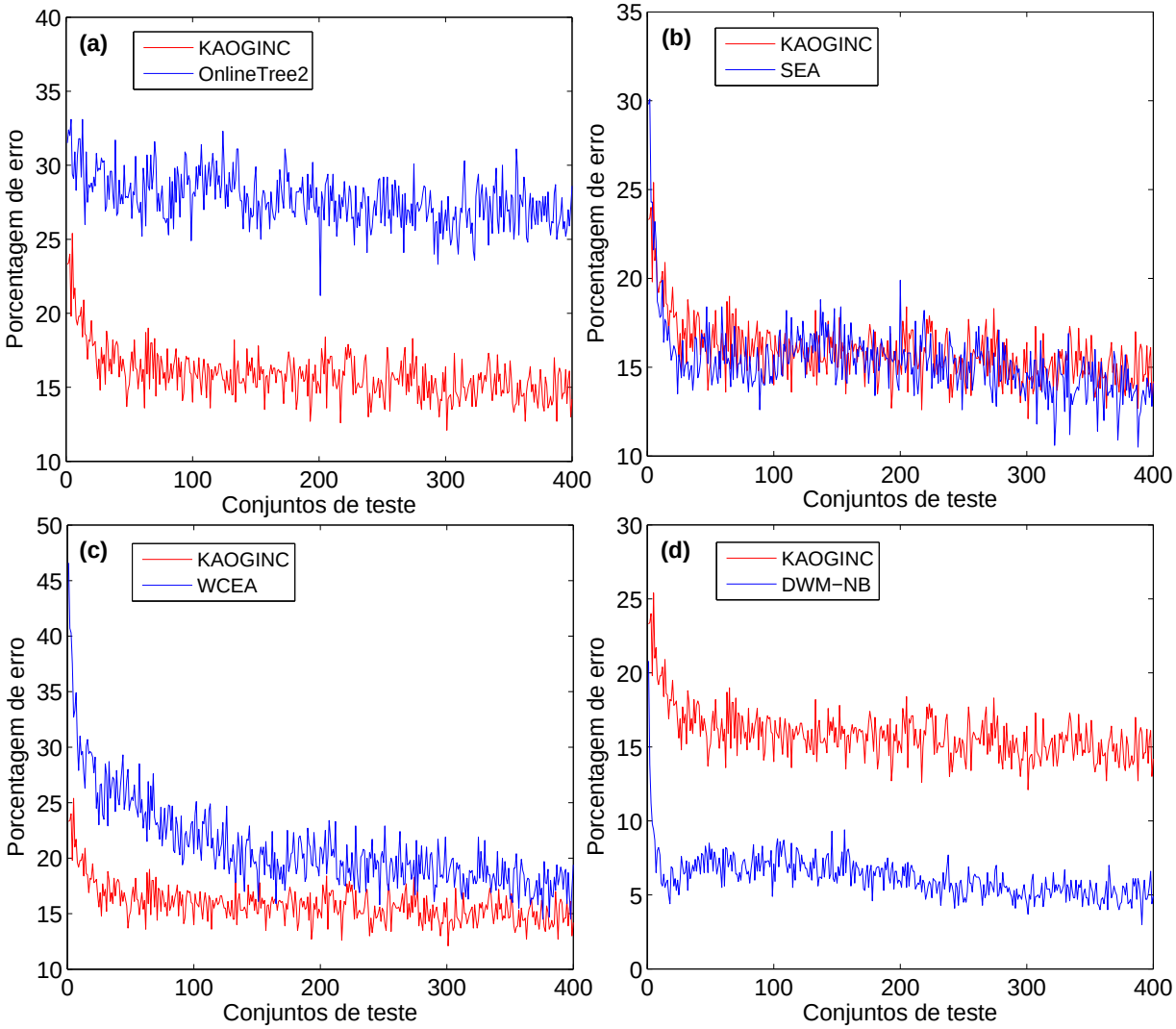

Figura 4.11: Comparações de desempenho entre o algoritmo KAOGINC e (a) OnlineTree2; (b) SEA; (c) WCEA e (d) DWM-NB para o domínio Hiperplano. A seleção de modelos resultou nos seguintes parâmetros, KAOGINC $\tau = 30$; SEA $\varepsilon = 25$; WCEA $\varepsilon = 25$ e DWM-NB $\rho = 50$.

performance bastante similar ao algoritmo proposto, o que mostra que ambos possuem excelente capacidade de adaptação. O melhor desempenho foi obtido pelo algoritmo DWM-NB, que apresentou uma diferença significativa com relação aos demais. Os fatores que permitiram esse ótimo desempenho são o fato de não haver restrição quanto ao número de classificadores que compõem o comitê, e o alto valor para o parâmetro ρ que esta associado à frequência em que o comitê é revisado. Esses fatores permitem que o comitê, formado pelo algoritmo DWM-NB, armazene classificadores treinados em diversos conceitos, mas dando mais peso aos classificadores que acertam, além do fato de diferenciar-se dos outros comitês quanto ao tipo do algoritmo base. Os algoritmos WCEA e OnlineTree2, apresentaram desempenhos discretos comparados aos demais. Principalmente, o algoritmo OnlineTree2, que apesar de apresentar bons resultados em domínios com mudança de conceito gradual, não obteve bom desempenho, devido principalmente às constantes mudanças abruptas.

4.5.3 Círculos

Esta base de dados é gerado com dados em duas dimensões, x_1 e x_2, pertencentes ao intervalo $[0, 1]$, onde as classes são determindas de acordo com a Equação (4.5). Se a equação for satisfeita, atribui-se a classe ω_1, caso contrário a classe ω_2 é atribuida.

$$(x_1 - a)^2 + (x_2 - b)^2 > r^2 \tag{4.5}$$

Na equação, a e b correspondem às coordenadas do centro do círculo e r ao raio. Para o experimento, o conjunto gerado é formado de 10.000 instâncias, divididos em conjuntos de treino e teste com 25 instâncias cada, e apresentados intercaladamente, treino e teste, para os classificadores. Para simular o desvio de conceito, a cada 2.500 iterações, a é incrementado de $0,2$ e r de $0,05$, iniciando com os valores, $a = 0,2, b = 0,5$ e $r = 0,15$. Trata-se portanto, de um domínio com mudanças de conceito gradual, uma vez que o círculo sofre alterações suaves no centro e no raio. A Figura 4.12 mostra as comparações entre o algoritmo proposto e os algoritmos considerados para comparação.

Note que, por tratar-se de um domínio com mudança de conceito gradual, o algoritmo OnlineTree2 obteve bom desempenho, ficando atrás somente do algoritmo proposto KAOGINC, com relação a média de acertos. Este resultado mostra que o algoritmo proposto consiste em uma boa opção para qualquer tipo de mudança de conceito. Os algoritmos baseados em comitê tiveram performance pior, principalmente os que utilizam pesos, como o DWM-NB e o WCEA. Note que o desempenho desses algoritmos piorou entre a primeira e terceira mudança de conceito, este fato provalvelmente está relacionado à alteração no número de exemplos de dados por classe. No primeiro conceito, a classe ω_1 é menor que a classe ω_2 pois consiste nos exemplos que encontram-se dentro do círculo. Ao passo que o raio do círculo aumenta, a classe ω_1 tende a aumentar, no entanto, durante esse processo também começa a ocorrer sobreposição entre classes. Por essa razão os conceitos intermediários apresentam maior dificuldade para os classificadores. No último conceito a diferença entre as classe é menor, bem como a sobreposição, o que explica a pequena melhora no desempenho.

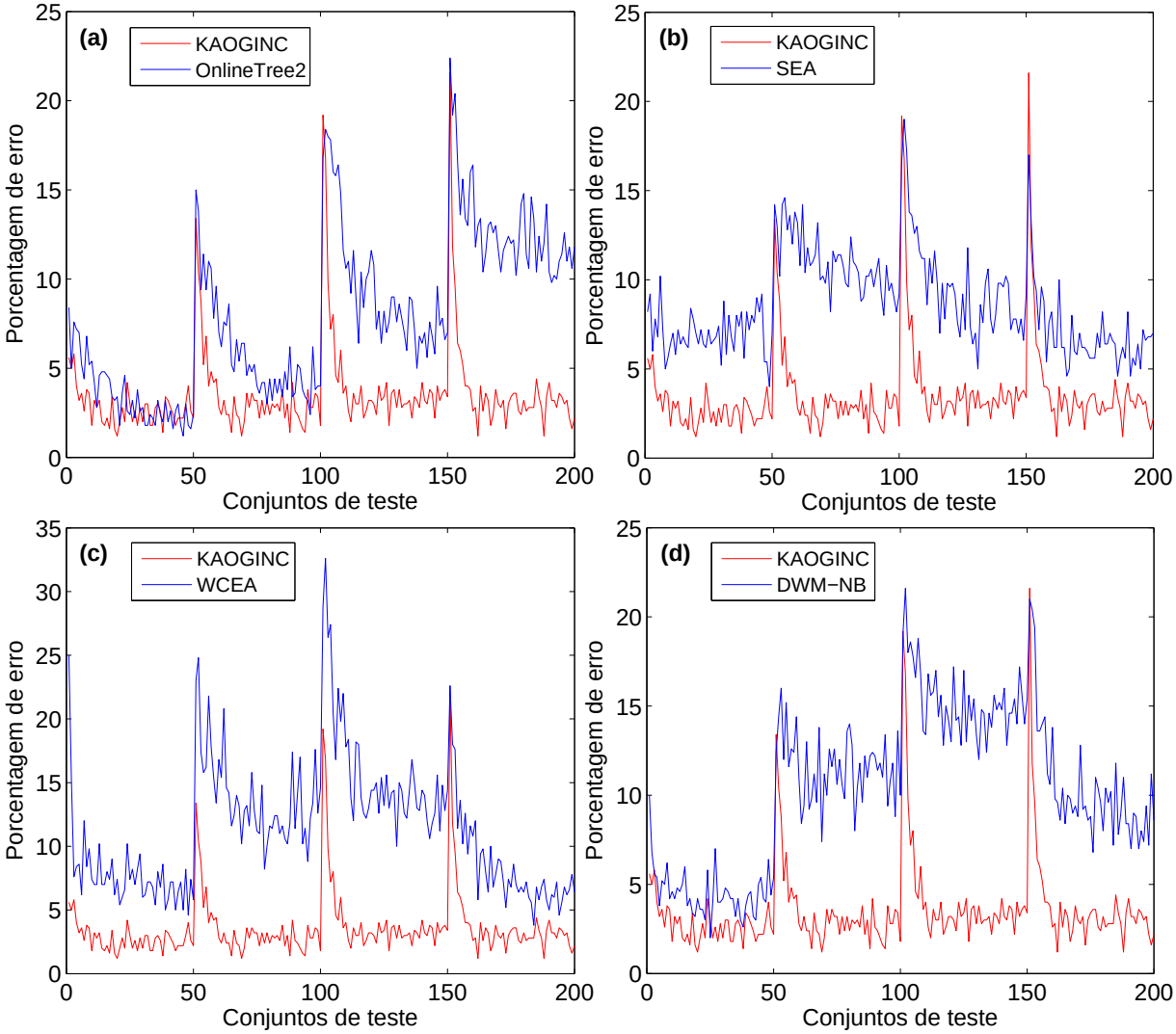

Figura 4.12: Comparações de desempenho entre os algoritmos KAOGINC e (a) OnlineTree2; (b) SEA; (c) WCEA e (d) DWM-NB para o domínio Círculos. A seleção de modelos resultou nos seguintes parâmetros, KAOGINC $\tau = 10$; SEA $\varepsilon = 5$; WCEA $\varepsilon = 35$ e DWM-NB $\rho = 25$.

4.5.4 Seno

O conjunto Seno também é constituído de dois atributos, x_1 e x_2, gerados aleatoriamente no intervalo $[0, 1]$, e a classe é atribuída de acordo com a Equação (4.6).

$$0.5 + 0.3sen(3\pi x_1) > x_2 \tag{4.6}$$

Como nos casos anteriores, atribui-se a classe ω_1 se a equação é satisfeita e ω_2 caso constrário. Para os experimentos, foram gerados 10.000 instâncias e o mudança de conceito abruta é simulado com a repentina troca na atribuição das classes, realizada três vezes neste caso. A Figura 4.13, mostra os resultados para os algoritmos considerados no domínio Seno.

Neste domínio, os algoritmos utilizaram os parâmetros KAOGINC com $\tau = 8$, o SEA com 3, WCEA com 25 e DWM-NB com $\rho = 10$. Nota-se que o domínio exige que o

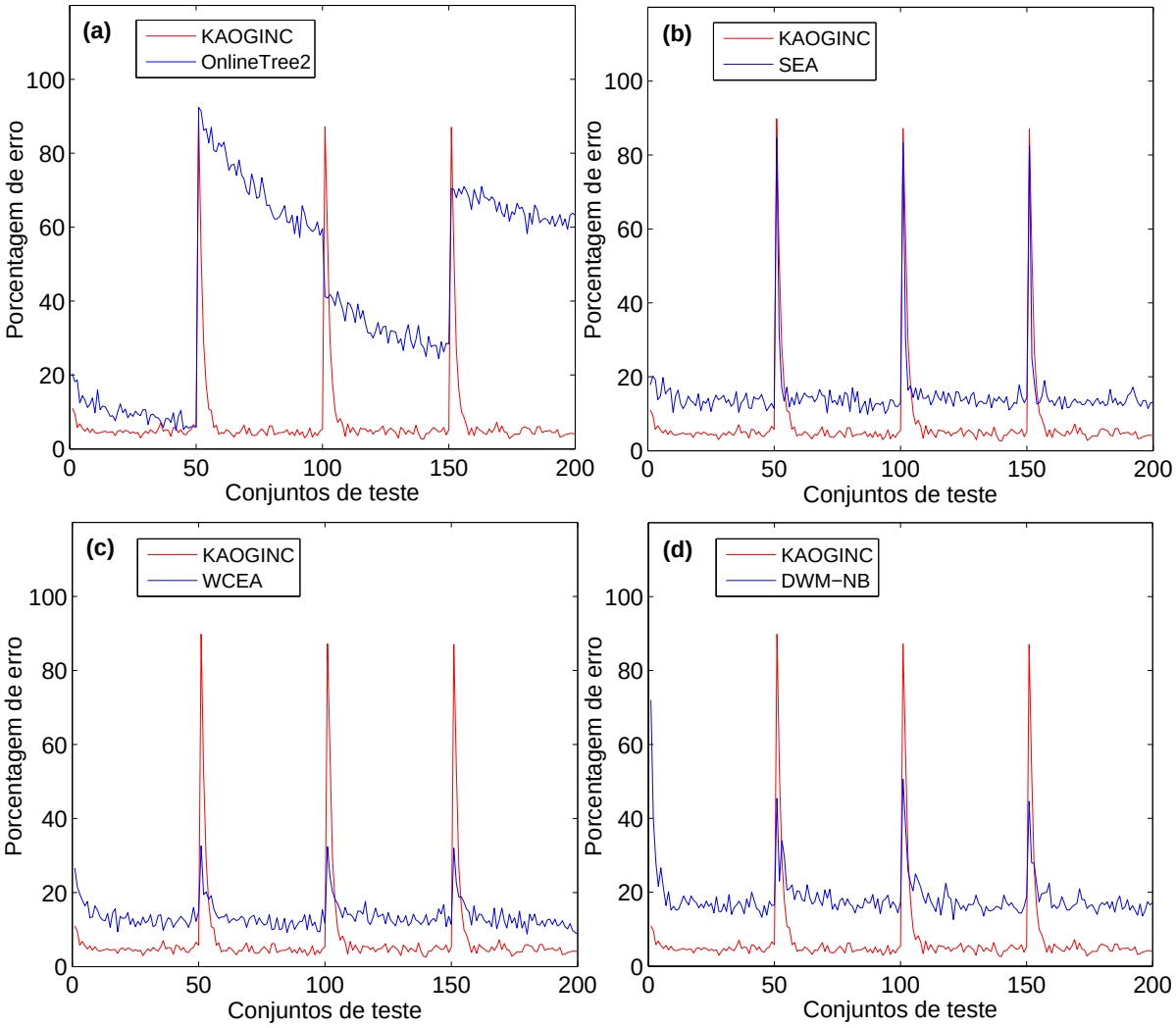

Figura 4.13: Comparações de desempenho entre os algoritmos KAOGINC e (a) OnlineTree2; (b) SEA; (c) WCEA e (d) DWM-NB para o domínio Seno. A seleção de modelos resultou nos seguintes parâmetros, KAOGINC $\tau = 8$; SEA $\varepsilon = 3$; WCEA $\varepsilon = 25$ e DWM-NB $\rho = 10$.

classificador recupere-se rapidamente, pois a distribuição varia rapidamente, o que explica o tamanho pequeno para os comitês - exceto o WCEA, que obteve seu melhor desempenho com comitê com 25 árvores de decisão. Como se trata de um domínio composto por 3 mudanças de conceito abruptas, o classificador OnlineTree2 obteve o pior desempenho, uma vez que o mesmo não se atualiza rapidamente quanto aos algoritmos baseados em comitê, ou mesmo o algoritmo KAOGINC. Como mostra a Figura 4.13, o algoritmo proposto obteve o melhor resultado, principalmente durante as fases de conceito estáticos. Apesar de apresentar alta taxa de erro nas mudanças de conceito (por volta de 85%), o algoritmo KAOGINC apresentou ótima recuperação, ainda melhor do que a recuperação apresentada pelo algoritmo WCEA, que foi o algoritmo que apresentou maior estabilidade neste domínio.

4.5.5 Gaussianas

O domínio Gaussianas consiste na geração de duas distribuições Gaussianas, uma com centro $(0, 0)$ e desvio padrão 1 e outra com centro $(2, 0)$ e desvio padrão 4. Pada cada distribuição é atribuída uma classe, que se inverte a cada 2.500 instâncias, o conjunto total compreende 10.000 instâncias. As classe não são exatamente balanceadas, para a geração de uma nova instância, sorteia-se qual distribuição será usada, e consequentemente sua classe, de modo que as Gaussianas têm tamanho similar mas não igual. Este domínio simula mudança de conceito abrupta em um cenário com dados com ruído - Gaussianas sobrepostas. As comparações entre o algoritmo proposto KAOGINC e os demais algoritmos utilizados para comparação são apresentados na Figura 4.14.

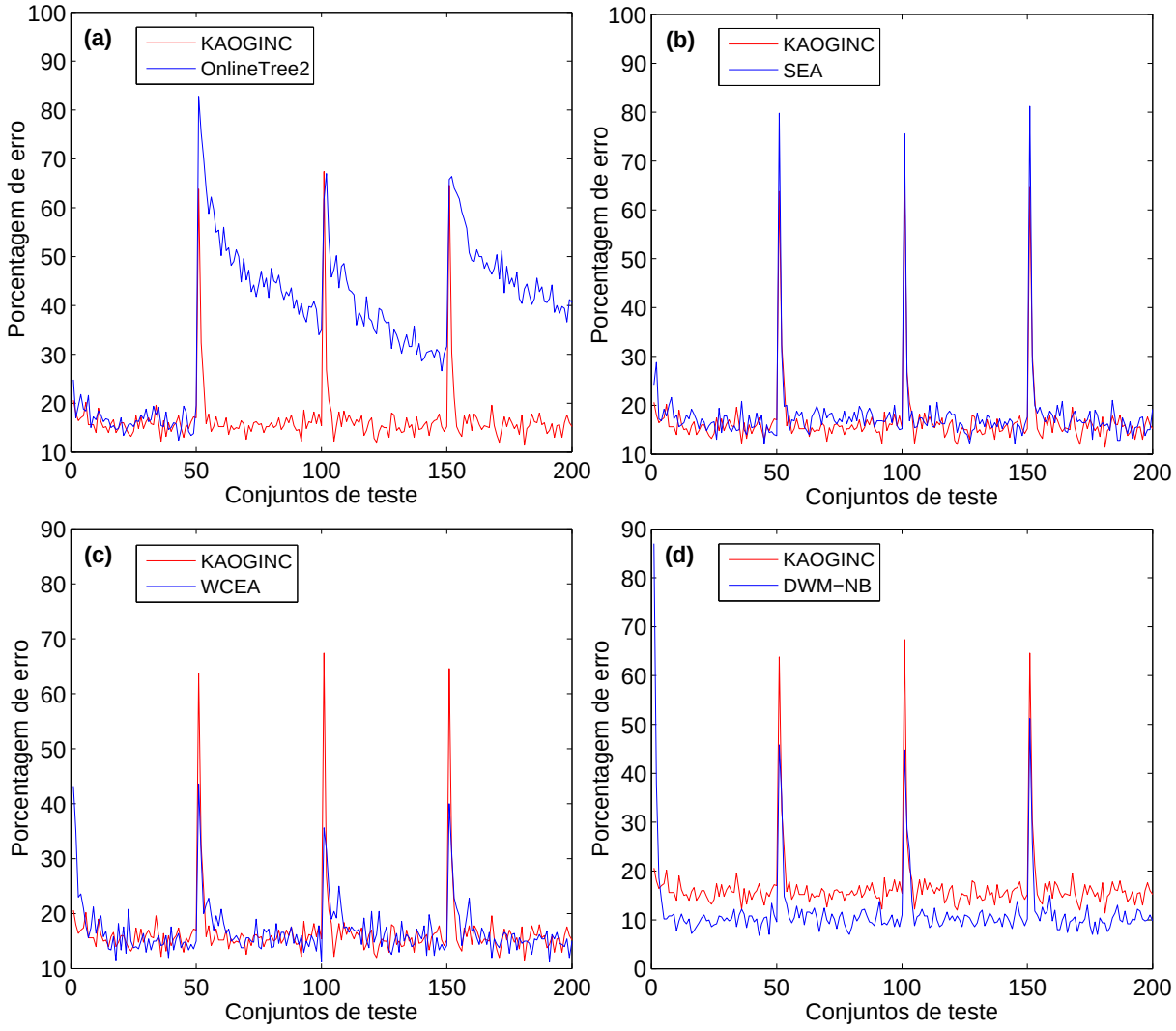

Figura 4.14: Comparações de desempenho entre os algoritmos KAOGINC e (a) OnlineTree2; (b) SEA; (c) WCEA e (d) DWM-NB para o domínio Gaussianas. A seleção de modelos resultou nos seguintes parâmetros, KAOGINC $\tau = 5$; SEA $\varepsilon = 3$; WCEA $\varepsilon = 20$ e DWM-NB $\rho = 3$.

O domínio também apresenta mudanças de conceito abruptas, por essa razão o algoritmo OnlineTree2 comportou-se como no domínio Seno, ou seja, apresentou dificuldades na recuperação de desempenho após cada mudança de conceito. Os algoritmos paramétri-

cos obtiveram melhores resultados, com os seguintes parâmetros, KAOGINC com $\tau = 5$, o SEA com 3 , WCEA com 20 e DWM-NB com $\rho = 3$. Novamente, como no domínio Seno, nota-se que este exige que o classificador recupere-se rapidamente, por essa razão o tamanho pequeno para os comitês (exceto o WCEA). Pode-se notar que o desempenho entre os algoritmos SEA e WCEA e o algoritmo KAOGINC foram similares, algumas poucas diferenças entre eles são: o algoritmo SEA obteve média de erro na mudança de conceito em torno de 80%, portanto maior que a do KAOGINC (65%) e do WCEA (40%). Já o algoritmo WCEA, mais uma vez, obteve desempenho bastante estável, no entanto, mostrou-se lento na recuperação de desempenho em comparação com os algoritmos KAOGINC e SEA. O melhor desempenho foi apresentado pelo algoritmo DWM-NB, tanto em porcentagem de acertos, em conceito estático, como em recuperação de desempenho após mudança de conceito. Nota-se que, apesar de desempenhos parecidos entre os algoritmos SEA, WCEA e KAOGINC, o algoritmo proposto obteve ligeira vantagem, ficando atrás somente do algoritmo DWM-NB.

4.5.6 Misto

Este domínio assemelha-se ao Seno, entretanto, este apresenta quatro atributos dos quais, dois deles, são bipolares (x_1 e $x_2 \in \{-1, 1\}$) e os demais x_3 e x_4 pertencem ao intervalo $[0, 1]$. A classe ω_1 é atribuída se a Equação 4.7 for satisfeita para algum dos atributos, x_1, x_2 ou x_3.

$$0.5 + 0.3sen(3\pi x_4) > x_1, x_2, x_3 \tag{4.7}$$

Para os experimentos, o conjunto completo consiste de 10.000 e as classes são ordem deatribuição das classes é invertida a cada 2.500 instâncias, simulando desvio de conceito abrupto em um domínio misto. Novamente, os conjuntos possuem 25 instâncias de treinamento e são apresentados intercaladamente.

Note que este domínio apresenta duas classes bastante misturadas e também simula mudança de conceito abrupta. Essas características colaboram para o mau desempenho do algoritmo OnlineTree2, que foi o pior dos algoritmos testados neste domínio. Ainda com relação ao algoritmo OnlineTree2, pode-se dizer que ocorreu superadaptação deste no processamento do primeiro conceito. O que, apesar de ter proporcionado bom desempenho no conceito em questão, dificultou a atualização da árvore após a mudança de conceito - o que justifica o péssimo desempenho no segundo conceito. De fato, a superadaptação inicial atrapalhou todo o restante do processamento, no terceiro conceito, igual ao primeiro, a árvore foi atualizada com exemplos do segundo conceito - que agora atrapalha a classificação de novos dados.

Os demais algoritmos obtiveram bom desempenho, principalmente os algoritmos baseados em comitê que usam peso, em especial o WCEA que apresentou o melhor desempenho. Note que o algoritmo SEA, demorou por volta de 9 iterações para retomar o desempenho,

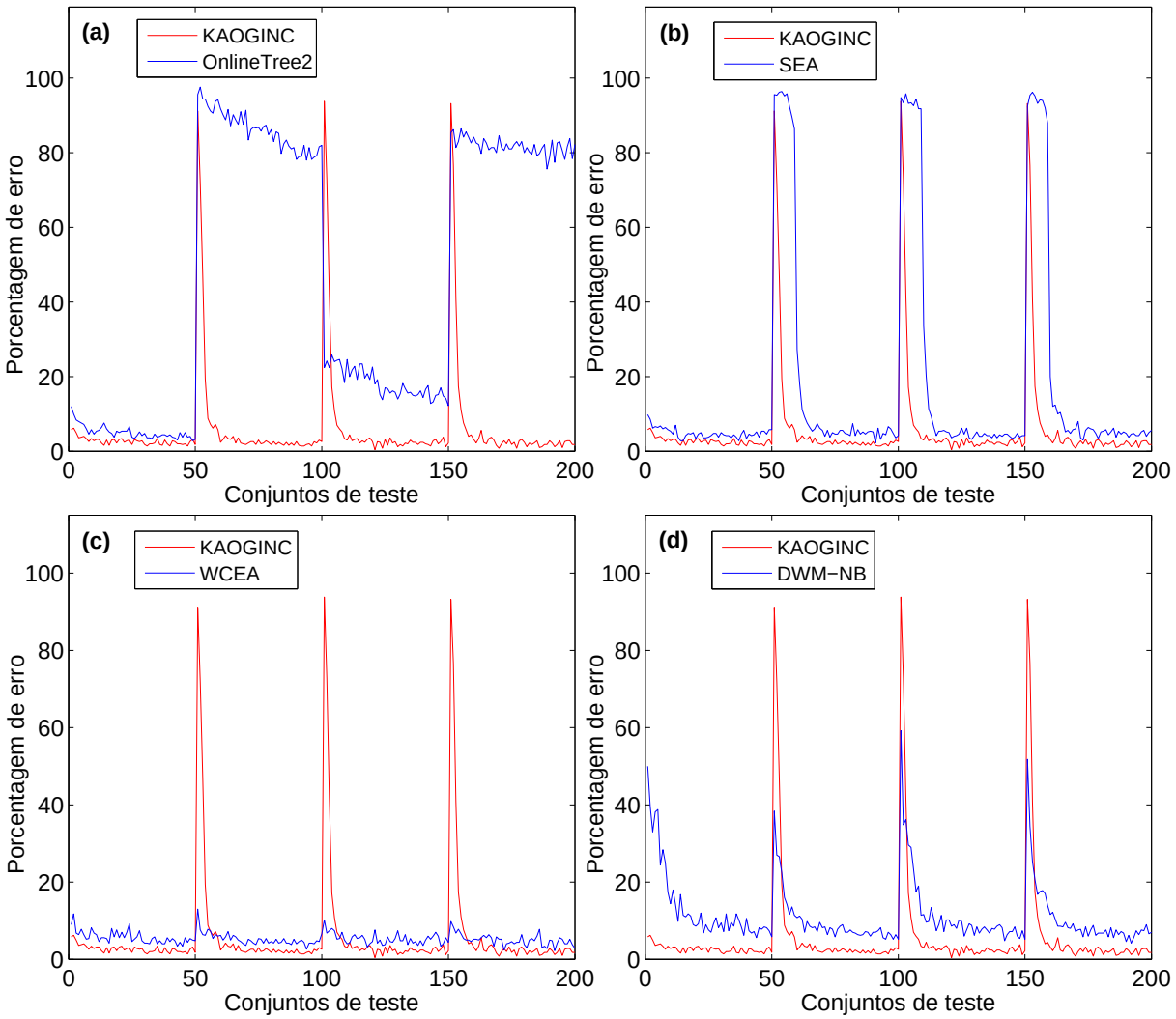

Figura 4.15: Comparações de desempenho entre os algoritmos KAOGINC e (a) OnlineTree2; (b) SEA; (c) WCEA e (d) DWM-NB para o domínio Misto. A seleção de modelos resultou nos seguintes parâmetros, KAOGINC $\tau = 18$; SEA $\varepsilon = 19$; WCEA $\varepsilon = 20$ e DWM-NB $\rho = 30$.

o que corresponde aproximadamente à renovação de metade do comitê com 19 classificadores. O KAOGINC, por sua vez, apresentou o melhor desempenho em conceito estático, no entanto, apresentou alta taxa de erro nas mudanças de conceito, o que degradou sua performance.

4.5.7 Preço de energia elétrica

Este domínio, conhecido como Elec2, é formado por dados reais que correspondem a dados sobre o mercado de eletricidade coletados no estado Australiano de New South Wales e descrito por Harries (1999) e Gama *et al.* (2004). O domínio consiste de 45.312 instâncias de dados, sendo que cada uma delas foi coletada em um intervalo de 30 minutos, entre 7 de Maio de 1996 e 5 de Dezembro de 1998. Cada instância possui 5 atributos e uma classe, que indica se o preço aumentou ou diminuiu em relação ao preço anterior. Dentre os atributos, dois deles são de tempo, dia da semana, variando no intervalo $[1, 7]$;

e período do dia, no intervalo $[1, 48]$. Os atributos x_3 e x_4 correspondem à demanda em New South Wales, em Victoria e o quinto atributo corresponde ao montante de energia reservada para ser transferida de um estado para outro. A tarefa é prever se o preço aumentará ou diminuirá ao longo do tempo.

Como trata-se de dados reais, os experimentos foram organizados de forma diferente, como sugerido por (Núnez *et al.*, 2007), os dados são processados em semanas, ou seja, cada conjunto de dados possui 336 instâncias (com exceção do primeiro com 288). Como não há muitos dados e, considerando que em uma situação real não se pode perder dados rotulados; todos os conjuntos no fluxo de dados são rotulados. A estimativa do erro é feita considerando o novo conjunto como conjunto de teste - classificando suas instâncias. Sendo que este é depois usado como treino. A Figura 4.16 mostra a porcentagem de erro das previsões de gasto por semana dada pelos algoritmos incrementais considerados para comparação neste trabalho.

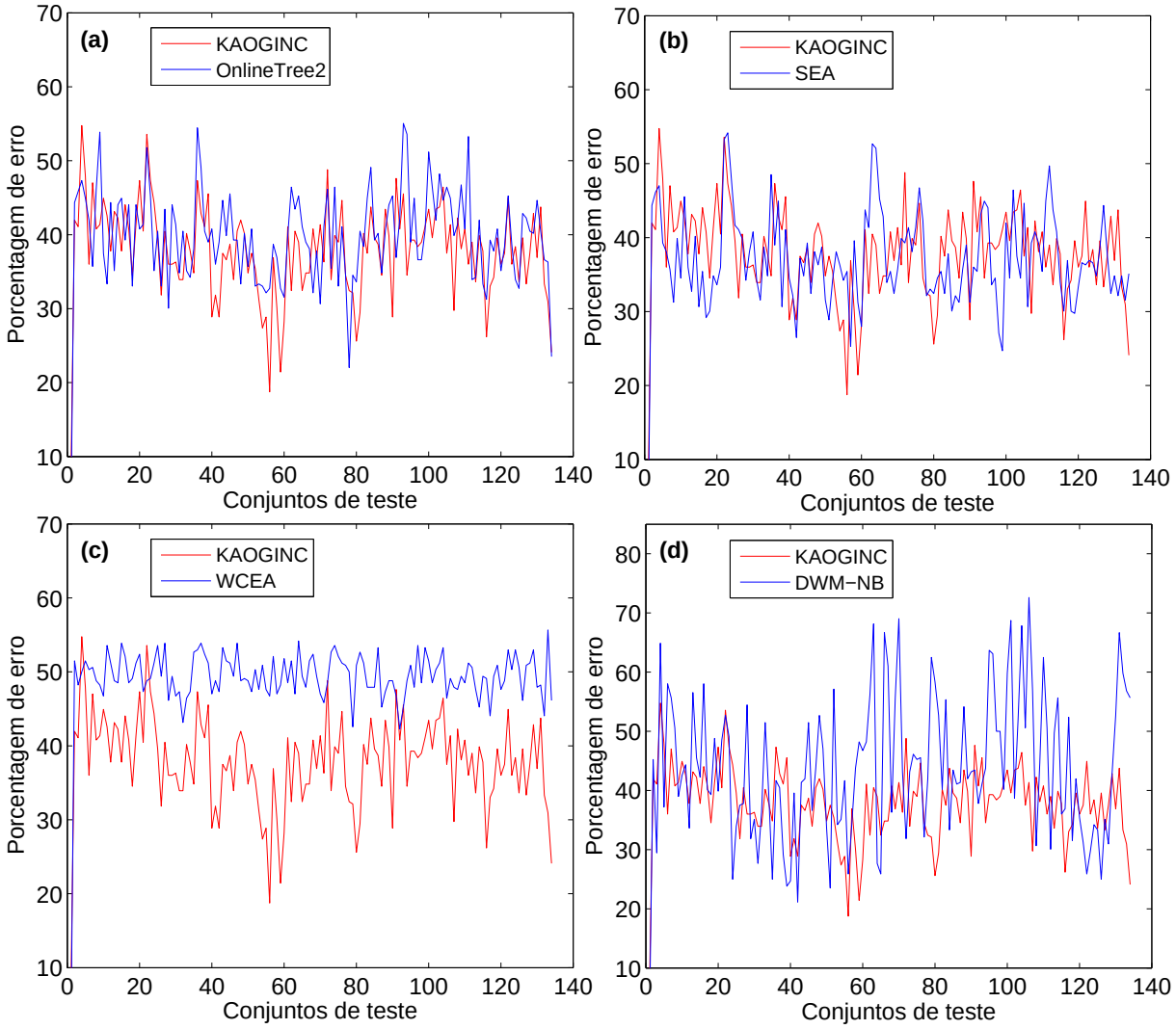

Figura 4.16: Comparações de desempenho entre os algoritmos KAOGINC e (a) OnlineTree2; (b) SEA; (c) WCEA e (d) DWM-NB para a base de dados reais Elec2. A seleção de modelos resultou nos seguintes parâmetros, KAOGINC $\tau = 65$; SEA $\varepsilon = 25$; WCEA $\varepsilon = 25$ e DWM-NB $\rho = 1$.

Pode-se observar que os três algoritmos que apresentaram melhor performance foram o SEA, o KAOGINC e o OnlineTree2, sendo que os dois primeiros tiveram performance similar entre si e pouco superior à do OnlineTree2. O algoritmo WCEA novamente foi o mais estável, no entanto, apresentou desempenho pior que os demais algoritmos. O algoritmo DWM-NB também apresentou performance ruim, principalmente porque a porcentagem de acerto variou muito.

Como na maioria dos domínios reais, as mudanças de conceito não são obvias como nos domínios artificiais. E, além disso, esse tipo de domínio é bem mais suscetível à ruídos, que muitas vezes, em situações reais exige a aplicação de algum tipo de filtro (Allen e Mills, 2004), para que o fluxo de dados possa ser processado. Os experimentos realizados neste trabalho, não consideram o uso de filtro, devido ao caráter comparativo dos testes bem como do escopo do trabalho.

4.5.8 Classificação de mão no jogo de pôquer

Este domínio consiste em 25.010 instancias de dados. Cada instância refere-se a um exemplo de mão no jogo de pôquer, onde cada mão consiste de cinco cartas dentre as 52 possíveis. Cada carta é representada por dois atributos, naipe e número, resultando em um total de 10 atributos. A classe descreve a pontuação da combinação das cinco cartas e representa as 10 possíveis mãos no jogo de pôquer, a saber, $\Omega = \{$*trash*, *one pair*, *two pairs*, *three of a kind*, *straight*, *flush*, *full house*, *four of a kind*, *straight flush*, *royal flush*$\}$. O objetivo é, dado uma sequência de 5 cartas, classificar a mão de pôquer em uma das 10 categorias quanto à sua pontuação. Os experimentos foram conduzidos sem repetição, o conjunto inicial é composto pelos primeiros 1.010 instâncias, e o fluxo de dados é composto por 240 conjuntos de 100 combinações de 5 cartas cada.

O interessante neste domínio é o fato de ser multiclasse, o que da oportunidade de verificar o desempenho dos algoritmos testados em bases com mais de duas classes. O que se observa nos resultados é que os algoritmos apresentaram performances não muito diferentes entre si. No entanto, pode-se afirmar que os algoritmos mais estáveis foram o algoritmo DWM-NB e o KAOGINC, sendo que estes foram os dois melhores neste domínio, com uma pequena vantagem do algoritmo proposto KAOGINC. O fato do domínio possuir mais que duas classes favorece o algoritmo KAOGINC que é naturalmente multiclasse. Já o algoritmo DWM-NB é favorecido por não possuir limitação no número de classificadores no comitê, dado que, dessa forma mais algoritmos base podem ser usados para representar as várias classe. Os demais algoritmos baseados em comitê têm a desvantagem de precisar alterar muitas vezes os algoritmos base quanto à classe que representa; o que também explica a maior variação nos resultados desses algoritmos. Por último o algoritmo OnlineTree2 obteve resultado equivalente aos algoritmos WCEA e SEA, provavelmente devido a motivo semelhante - dificuldade na atualização devido ao número de classes.

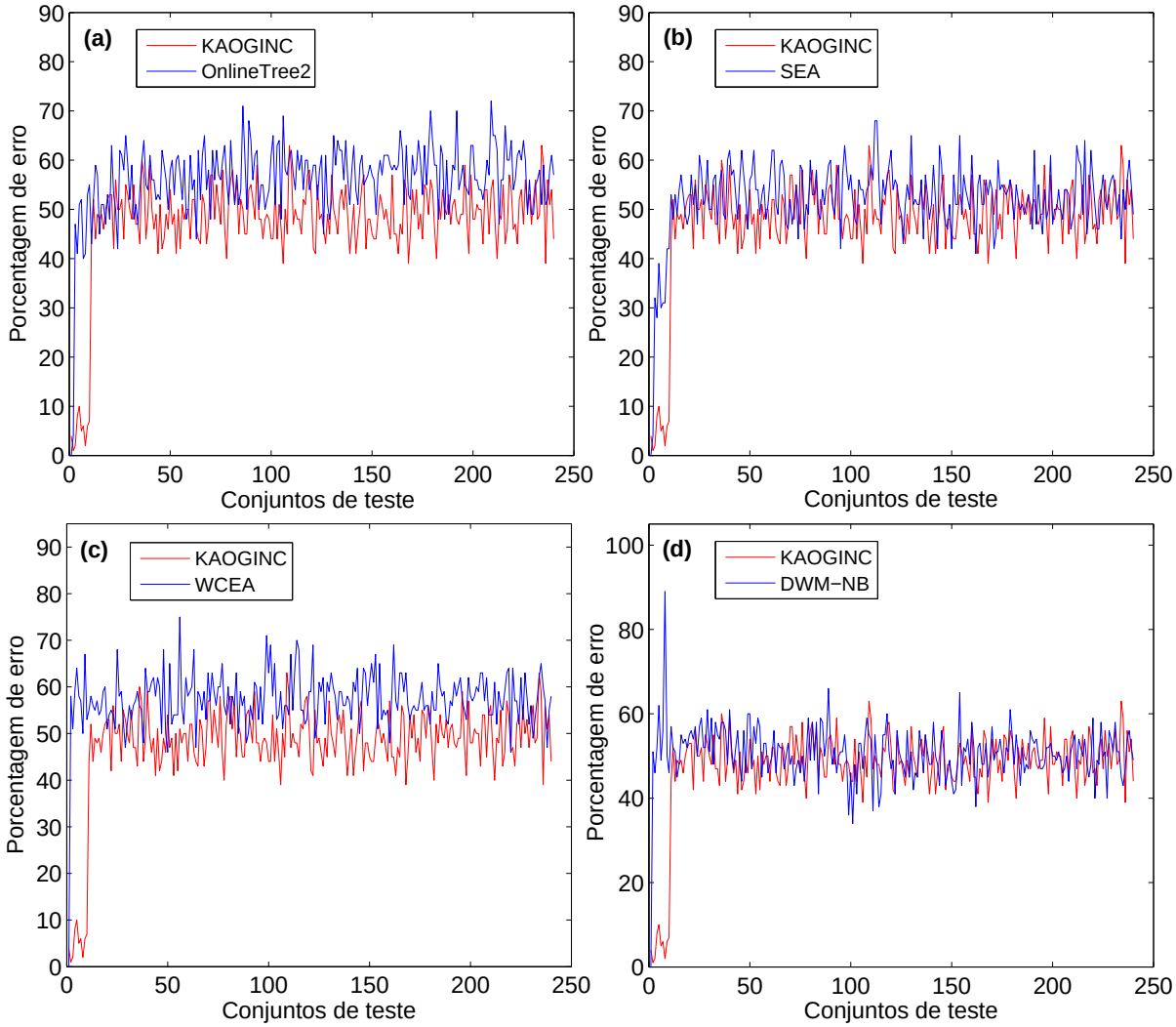

Figura 4.17: Comparações de desempenho entre os algoritmos KAOGINC e (a) OnlineTree2; (b) SEA; (c) WCEA e (d) DWM-NB para a base de dados reais Poker Hand. A seleção de modelos resultou nos seguintes parâmetros, KAOGINC $\tau = 33$; SEA $\varepsilon = 35$; WCEA $\varepsilon = 35$ e DWM-NB $\rho = 10$.

4.5.9 *Filtro de Spam*

Problema clássico de detecção de spam consiste em determinar se uma mensagem eletrônica é spam ou não, duas classes, portanto. O domínio contém 4601 instâncias com 54 atributos cada. Todos os atributos são contínuos e correspondem à frequência de repetição de palavras, caracteres ou maiúsculas e minúsculas. Para os testes com este domínio o conjunto inicial possui 1.001 instâncias, e os demais conjuntos, todos rotulados, possuem 45 instâncias. Como nos demais domínios reais todos os conjuntos são rotulados e a estimativa do erro de classificação é feita utilizando o próximo conjunto para testar o classificador, antes de usá-lo como treino. A Figura 4.18 mostra os resultados dos algoritmos da comparação entre o algoritmo KAOGINC e os demais.

Na tarefa de filtro de spam, segundo o domínio utilizado, os algoritmos baseados em comitê obtiveram vantagem em relação aos demais. O algoritmo SEA apresentou o melhor resultado, em porcentagem de acerto. O algoritmo mais estável foi o DWM-

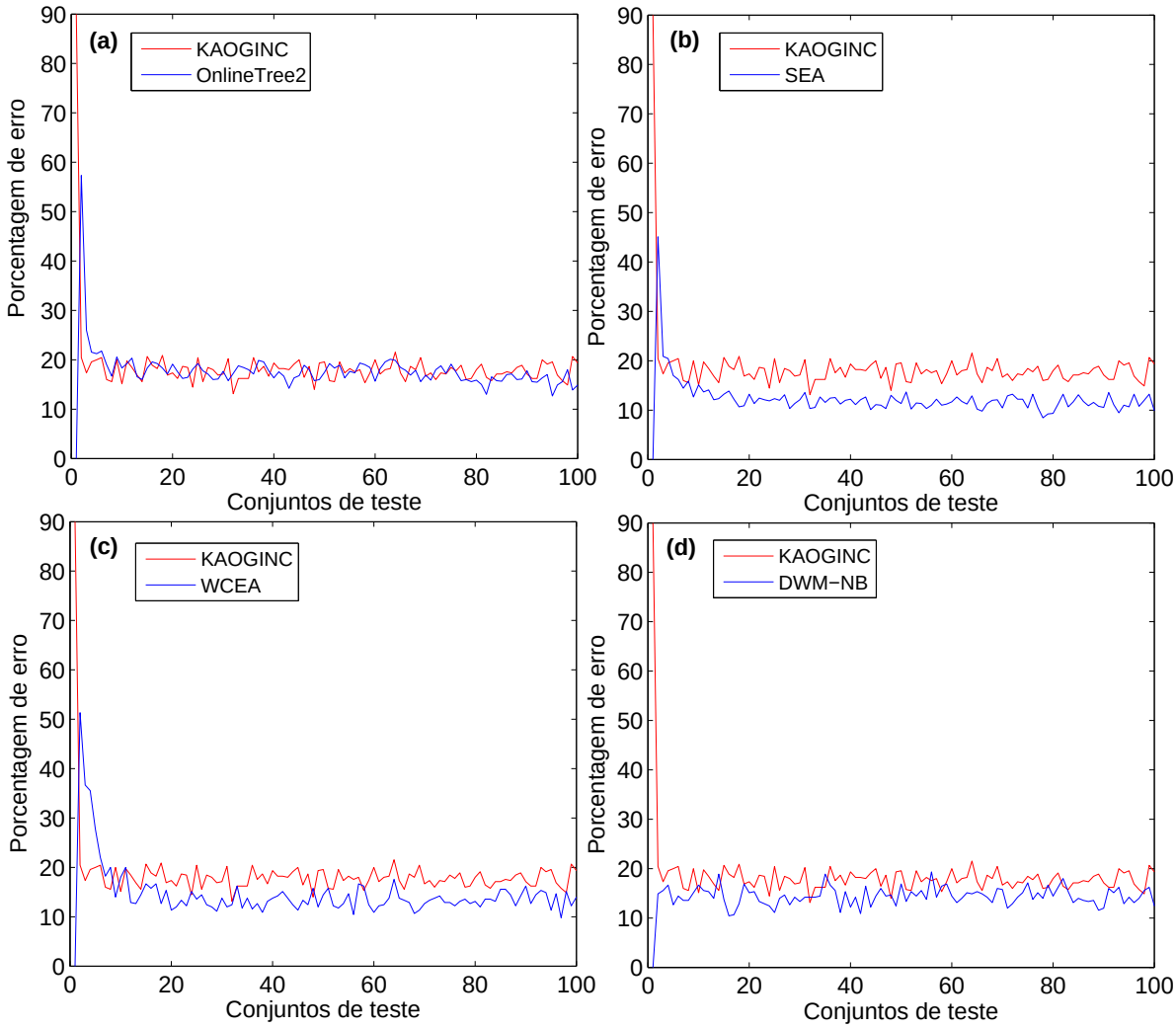

Figura 4.18: Comparações de desempenho entre os algoritmos KAOGINC e (a) OnlineTree2; (b) SEA; (c) WCEA e (d) DWM-NB para a base de dados reais Spam. A seleção de modelos resultou nos seguintes parâmetros, KAOGINC $\tau = 8$; SEA $\varepsilon = 15$; WCEA $\varepsilon = 28$ e DWM-NB $\rho = 13$.

NB uma vez que não apresentou variações na mudança de conceito, principalmente a variação apresentada pelos demais algoritmos no início do processamento. Os algoritmos KAOGINC e OnlineTree2 apresentaram desempenho semelhante, no entanto, aquém do desempenho apresentado pelos algoritmos baseados em comitê. Neste domínio o uso de vários classificadores provou-se ser mais eficiente que apenas um algoritmo adaptativo.

4.5.10 Detecção de intrusos em redes de computadores

Este domínio corresponde ao desafio proposto para a conferência KDD'99. Consiste em identificar intrusos em uma rede de computadores tendo como base um conjunto de dados com 311.029 instâncias que armazenam informações sobre uma determinada conexão. Uma conexão corresponde a uma sequencia de pacotes TCP com tempos definidos para início e fim. Neste tempo, os dados passam de um emissor a um receptor sob um determinado tipo de protocolo. Cada conexão (instância) é rotulada como normal ou como um

ataque, sendo que existem 22 tipos diferentes de ataques. Dessa forma, é desejável que um sistema classificador seja capaz de, não só determinar se ocorreu ataque, mas também que tipo de ataque aconteceu. Para as simulações neste trabalho, somente a primeira tarefa é atribuída aos classificadores, i.e. determinar a ocorrência de ataque; trata-se portanto de uma tarefa com duas classes. Para o experimento, o conjunto inicial foi diferenciado dos demais, e possui 1.029 instâncias, os demais conjuntos possuem 100 instâncias e são apresentados sequêncialmente de acordo com a ordem apresentada na base real. Este domínio é o maior utilizado neste trabalho, corresponde a uma situação real e é um bom teste para os algoritmos, principalmente no que diz respeito à estabilidade e tempo de execução. A Figura 4.19 ilustra os resultados da classificação dos algoritmos para o domínio KDD'99.

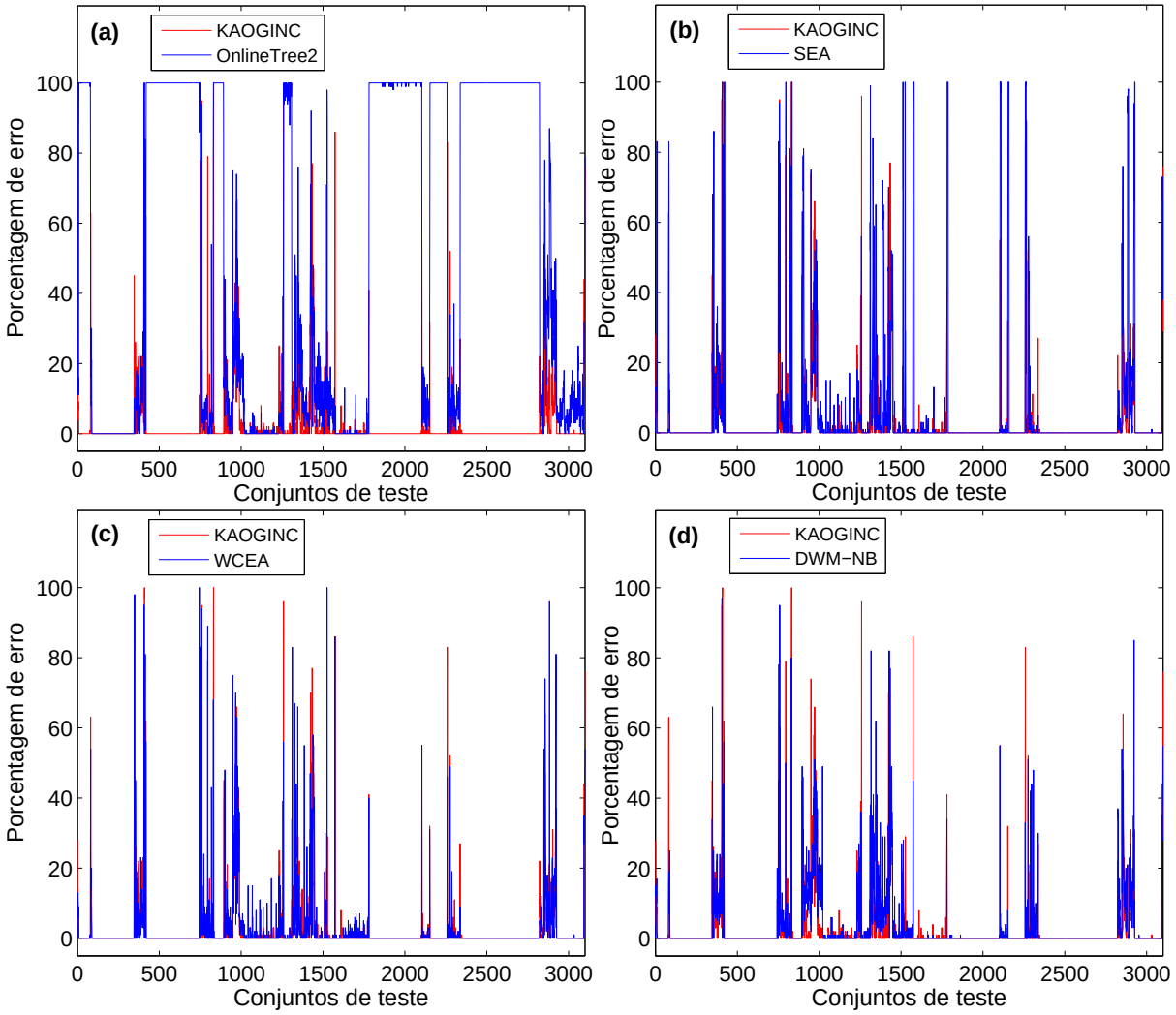

Figura 4.19: Comparações de desempenho entre os algoritmos KAOGINC e (a) OnlineTree2; (b) SEA; (c) WCEA e (d) DWM-NB para a base de dados reais KDD'99. A seleção de modelos resultou nos seguintes parâmetros, KAOGINC $\tau = 18$; SEA $\varepsilon = 15$; WCEA $\varepsilon = 28$ e DWM-NB $\rho = 30$.

Note que este apresenta diversas mudanças de conceito abruptas que, apesar de detectadas por todos os algoritmos testados, cada um reagiu de maneira diferente. Note na Figura 4.19(a) que o algoritmo OnlineTree2 apresentou o pior resultado, observe que o algoritmo apresentou dificuldade após várias mudanças de conceito e errou todas as

instâncias do conjunto repetidas vezes durante várias iterações, demorando a retomar o desempenho. O algoritmo OnlineTree2 não só apresentou o pior desempenho com relação à porcentagem de acertos, mas também foi o algoritmo mais demorado nas simulações. O algoritmo SEA também apresentou resultado insatisfatório, o maior problema ocorreu durante as mudanças de conceito, sendo que na maioria das vezes o algoritmo errou a classe de todas as instâncias do conjunto. Uma possível explicação é que tenha ocorrido superadaptação durante as fases de conceito estável, uma vez que o comitê é renovado retirando o algoritmo que mais comete erro. Os algoritmos baseados em comitê com peso obtiveram melhor resultado, possivelmente pelo fato de evitarem superadaptação devido ao uso de peso também no processo de retirada de classificadores do comitê. O algoritmo KAOGINC apresentou o melhor resultado dentre os algoritmos testados, mostrando ser uma opção a considerar no processamento de fluxo de dados.

4.5.11 Análise estatística

Esta seção apresenta a análise dos resultados dos algoritmos incrementais nos 10 domínios apresentados nas Seções 4.5.1 - 4.5.10. Para a comparação dos resultados foi usada a média de acertos ao longo do processamento do fluxo de dados, i.e. média da porcentagem de erro nos conjuntos de teste. Assim como na Seção 3.4.3, o método de análise estatística escolhido foi o teste de Friedman, descrito em detalhes por Demšar (2006). O método não depende de parâmetros e é equivalente ao método ANOVA com medidas repetidas. O passo inicial consiste em ordenar os melhores desempenho para cada domínio de dados. Dessa forma, considere as médias de acerto dos algoritmos para os domínios considerados nesta comparação. Bem como a cada resultado é atribuído a posição do resultado em relação aos demais para cada domínio mostrado na Tabela 4.1. A média das posições para cada algoritmo será usada para a comparação entre os algoritmos e apresentada ao final da tabela.

O primeiro passo, no teste de Friedman, é determinar se os resultados diferem da hipótese nula. Rejeitar a hipótese nula é importante pois garante que os algoritmos diferem entre si, em relação ao desempenho. Originalmente, Friedman (Friedman, 1937), (Friedman, 1940), propôs utilizar a distribuição χ^2 com um grau de liberdade $(N_a - 1)$, para negar a hipótese nula, como mostra a Equação (4.8).

$$\chi_F^2 = \frac{12N_d}{N_a(N_a-1)}\left[\sum_j {p_j}^2 - \frac{N_a(N_a-1)^2}{4}\right] \tag{4.8}$$

Na qual N_a corresponde ao número de algoritmos comparados, N_d ao número de domínios usados na comparação e p_j a média da posição do j-ésimo algoritmo. Entretanto, Iman e Davenport (1980) mostraram que a distribuição usada por Friedman é conservadora (Demšar, 2006), e propuseram utilizar a Equação (4.9), que é distribuída de acordo

Tabela 4.1: Resultados da comparação entre o algoritmo proposto KAOGINC e os demais algoritmos incrementais, considerando a média de acertos em conjuntos de teste ao longo do tempo. A tabela também apresenta a posição de cada algoritmo em relação aos demais para cada domínio não estacionário.

Domínio	**KAOGINC**	**OnlineTree2**	**SEA**	**WCEA**	**DWM-NB**
SEA	96,625 (1)	92,936 (5)	94,717 (4)	95,200 (2)	95,188 (3)
Hiperplano	84,429 (3)	72,285 (5)	84,793 (2)	79,048 (4)	93,816 (1)
Círculos	96,438 (1)	92,039 (2)	91,340 (3)	88,549 (5)	89,577 (4)
Seno	92,531 (1)	55,436 (5)	84,924 (3)	86,441 (2)	81,469 (4)
Gaussianas	83,337 (2)	62,687 (5)	82,035 (4)	83,124 (3)	88,301 (1)
Misto	93,951 (2)	53,287 (5)	82,421 (4)	94,674 (1)	88,524 (3)
Elec2	62,204 (2)	59,517 (3)	62,282 (1)	50,077 (5)	55,472 (4)
Poker hand	52,462 (1)	43,325 (4)	47,458 (3)	42,266 (5)	48,875 (2)
Spambase	82,371 (4)	80,248 (5)	85,468 (1)	84,437 (3)	84,677 (2)
KDD'99	97,493 (1)	49,315 (5)	94,288 (4)	97,389 (2)	96,671 (3)
Média das posições	1,8	4,4	2,9	3,2	2,7

com a distribuição F e apresenta dois graus de liberdades, $(N_a - 1)$ e $(N_a - 1) \times (N_d - 1)$.

$$F_F = \frac{(N_d - 1)\chi_F^2}{N_d(N_a - 1) - \chi_F^2} \tag{4.9}$$

O teste de Friedman utiliza a distribuição F para rejeitar a hipótese nula, calculada para os dois graus de liberdade $(N_a - 1)$ e $(N_a - 1) \times (N_d - 1)$, que no experimento corresponde ao valor da função $F(4, 36) = 2,63$, para grau de confiança $\alpha = 0,05$. Dessa forma, qualquer valor para F_F maior que $F(4, 36) = 2,63$, rejeita a hipótese nula. Para o estudo de caso em questão, $\chi_F^2 = 14,16$, e este será usado no cálculo da estimativa F_F de acordo com a Equação 4.9; como $N_a = 5$ e $N_d = 10$, $F_F \approx 4,93$, o que rejeita a hipótese nula. Portanto, até agora pode-se dizer que os algoritmos não são equivalentes.

Uma vez rejeitada a hipótese nula, prossegue-se com um teste *post-hoc*. Novamente como na Seção 3.4.3 o teste escolhido é o teste de Bonferroni-Dunn, que é indicado para a validação de um algoritmo controle, no caso o KAOGINC. Segundo o teste de Bonferroni-Dunn, se a diferença entre as médias das posições entre o algoritmo controle e algum outro algoritmo, for maior que a diferença crítica; o algoritmo que tiver menor media é considerado superior. A diferença crítica pode ser calculada como mostra a Equação (4.10).

$$DC = q_\alpha \sqrt{\frac{N_a(N_a + 1)}{6N_d}} \tag{4.10}$$

Na qual q_α corresponde ao um valor crítico para o teste de Bonferroni-Dunn, deter-

minado pelo número de classificadores e pelo grau de confiança, para o caso em questão $q_{0,05} = 2,498$. Para o estudo de caso em questão, com $\alpha = 0,05$, a diferença crítica é $DC \approx 1,44$. As diferenças entre as medias das posições entre o algoritmo KAOGINC e os demais correspondem a, OnlineTree2: $4,4 - 1.8 = 2,6 > 1,44$; SEA: $2,9 - 1,8 = 1,1 < 1,44$; WCEA: $3,2 - 1,8 = 1,4 < 1,44$; DWM-NB: $2,7 - 1,8 = 0,9 < 1,44$.

De acordo com esses resultados, pode-se dizer que o algoritmo proposto KAOGINC foi significativamente melhor que o algoritmo OnlineTree2. Já com relação aos demais, a conclusão estatística é que os dados não foram suficientes para diferenciar o algoritmo KAOGINC dos algortimos testados quanto ao desempenho. Note que, a diferença entre o algoritmo KAOGINC e o WCEA é bastante próxima à diferença crítica, para $\alpha = 0,05$. Já se o grau de confiança for considerado $\alpha = 0,1$; o valor para a diferença crítica é $DC \approx 1,29$. Aumentando o grau de confiança, diminui a certeza no resultado (de 95 para 90%), o que relaxa a diferença entre os algoritmos. Portanto para $\alpha = 0,1$, tem-se que o algoritmo KAOGINC é estatisticamente superior que os algoritmos OnlineTree2 e WCEA.

Como mostrado na análise estatística, o algoritmo proposto foi melhor que dois dos quatro algoritmos usados como na comparação, o algoritmo OnlineTree2 e o WCEA. Com relação aos demais, SEA e DWM-NB, não foi possível diferencia-los estatisticamente, no entanto, o algoritmo KAOGINC obteve a melhor média de posições entre os algoritmos. Em vista desses resultados, pode-se dizer que, quando considerados os domínios em questão sob as condições testadas, i.e. dados apresentados em conjuntos, o algoritmo KAOGINC é considerado a melhor opção entre os algoritmos testados.

Em relação aos desempenhos dos demais algoritmos nos testes realizados, concluiu-se que, o algoritmo OnlineTree2 foi o pior dos algoritmos testados, devido ao fato da demora na atualização da sua única árvore de decisão. O algoritmo mostrou-se bom para domínios que apresentam mudança de conceito gradual, no entanto, sob mudança de conceito abrupta, o mesmo leva muitas iterações para atualizar toda a árvore, o que impacta negativamente no desempenho do algoritmo. Outra desvantagem está relacionada ao tamanho do fluxo de dados, uma vez que este foi o algoritmo mais lento entre os testados. O algoritmo WCEA, apesar de não ter apresentado bons resultados em todos os domínios, mostrou-se bastante estável quanto ao desempenho em diversos domínios. O principal problema está relacionado ao tempo de execução, pois a atualização do algoritmo envolve validação cruzada. Fato que também impede que o algoritmo seja usado para processar dados em tempo real.

O algoritmo DWM-NB apresentou ótimos resultados e ficou em segundo na comparação. O ponto forte do algoritmo é a falta de restrição quanto ao tamanho do comitê, uma vez que, dessa forma, é possível armazenar mais conceitos. A maneira em que o algoritmo pondera os classificadores base no comitê impede que classificadores treinados em conceitos antigos interfiram negativamente na classificação. No entanto o ponto forte do algoritmo também pode ser um problema para seu desempenho, pois como o algo-

ritmo treina um novo classificador cada vez que o comitê comete um erro, pode acontecer que muitos classificadores sejam criados - o que pode ocasionar perda de desempenho e problemas de utilização de memória, principalmente em dados ruidosos. Por último, o algoritmo SEA é o mais simples dos algoritmos testados, consiste simplesmente em um comitê de classificares resolvido por simples votação. A simplicidade, no entanto, resulta em bom desempenho de tempo, apresentou o menor tempo de execução dentre os algoritmos testados. No geral, o algoritmo é bastante robusto e é considerado padrão como comparativo nos testes de algoritmos incrementais. Um dos pontos fracos é a demora na retomada de desempenho quando ocorre mudança de conceito brusca. Em casos que o comitê precisa ser totalmente atualizado, o número de iterações necessárias se equivale à metade do número de classificadores no comitê, uma vez que este é resolvido por votação simples.

Como conclusão geral, todos os algoritmos testados podem ser considerados no processamento de fluxo de dados, dado a restrição de cada um. Apesar de relativamente poucos domínios terem sido considerado para a realização dos testes, algoritmo proposto provou-se uma alternativa a ser considerado na classificação incremental de fluxo de dados.

Capítulo 5

Conclusões

Este trabalho abordou a classificação de dados utilizando grafos em dois contextos diferentes, a saber, dados com distribuição estacionária, i.e. onde a distribuição dos dados não se altera ao longo do tempo; e classificação de dados com distribuição não estacionária, na qual a distribuição dos dados sofre alterações ao longo do tempo. Em vista destes problemas, dois algoritmos foram desenvolvidos, o algoritmo estático, KAOG, para situações onde a distribuição não se altera, e a versão deste algoritmo que implementa aprendizado incremental, nomeado KAOGINC, para as tarefas que apresentam mudança de conceito. Ambos os algoritmos baseiam-se no grafo K-associado ótimo (ou principal) e na medida de pureza para estabelecerem a probabilidade de pertinência de um novo dado aos vários componentes do grafo e, dessa forma, atribuir a classe mais provável. Apesar disso, os algoritmos são independentes no que se refere à aplicações, considere inicialmente a análise do algoritmo estático KAOG.

Como mostrado no Capítulo 3, o algoritmo estático, KAOG, baseia-se em um grafo especial, chamado grafo K-associado ótimo. Este, por sua vez, é formado selecionando-se os melhores componentes, segundo a medida de pureza, dentre uma sequência de grafos K-associados; como somente são escolhidos os melhores componentes, o grafo resultante é ótimo em relação à pureza. A medida de pureza, definida para componentes, é parte central do algoritmo, pois esta quantifica o nível de mistura local de cada componente. De forma que, dado um exemplo a ser classificado, a pureza pode atuar como probabilidade de conexão para cada componente. Sendo que, esta depois de normalizada, é usada pelo classificador como probabilidade *a priori* para estimar a classe de novos dados. Como o grafo K-associado ótimo é construído com os componentes de maior pureza ao longo de vários grafos K-associados, os componentes no grafo ótimo, muito provavelmente, não foram formados considerando o mesmo valor de K. Intuitivamente cada componente no grafo ótimo é o componente que melhor representa a região do espaço de dados formada

pelos dados que o compõe. Portanto, é lógico pensar que, assim como na classificação feita pelo KNN, onde diferentes domínios de dados são melhores representados por diferentes valores de K, de forma análoga, em um único domínio é plausível que diferentes regiões do espaço de dados sejam melhores representadas por valores diferentes de K. Esta propriedade do grafo K-associado ótimo, não só permite uma melhor representação do espaço de dados, mas resulta em um processo de treinamento que não utiliza parâmetros. A independência de parâmetros significa que o algoritmo pode ser usado sem a necessidade de seleção de modelos, o que em alguns casos pode ser um fator indesejado, uma vez que a falta de variabilidade do algoritmo pode atrapalhar seu desempenho em alguns domínios. No entanto, considerando o algoritmo proposto, os resultados obtidos experimentalmente mostraram que este se equivale a algoritmos paramétricos do estado-da-arte na classificação de dados. Este resultado, em particular, favorece o algoritmo proposto, novamente, devido ao fato deste não possuir parâmetros e não exigir seleção de modelos, o que o torna uma escolha atrativa na prática considerando problemas de classificação de dados multiclasse.

O bom desempenho do classificador estático, KAOG, em domínios de dados com distribuição estacionária, demonstrou o potencial da abordagem em lidar com problemas de classificação, o que motivou o desenvolvimento da versão incremental. A versão incremental do algoritmo, basicamente, baseia-se na reestruturação do grafo utilizado pelo classificador, chamado grafo principal, neste caso. Para que este possa ser utilizado em domínios de dados com distribuição não estacionária, as atualizações devem compreender a inserção de novos conhecimentos e a retirada de conhecimento antigo que não é mais usado. No caso do algoritmo incremental proposto, a classificação é realizada por meio do cálculo da pertinência de um novo padrão aos componentes do grafo, de forma que a inserção e a remoção consideram as operações adição e remoção de componentes no grafo principal. À medida que os conjuntos de dados rotulados são apresentados, o algoritmo KAOGINC, utiliza o algoritmo KAOG para a construção do grafo ótimo a partir do conjunto de dados atual. Os componentes do grafo ótimo recém-construído são então adicionados no grafo principal. A remoção considera duas abordagens, a poda e a regra de não utilização. A poda retira do grafo principal todo componente associado a erro e, como mostrado na Seção 4.4, quando aplicada em conjunto com a regra de não utilização, o classificador apresenta melhores resultados. A regra de não utilização, por sua vez, utiliza o parâmetro τ para indicar o número de novos exemplos classificados para os quais qualquer componente pode permanecer sem que seja utilizado. Como visto na Seção 4.4, este parâmetro apresenta uma faixa de valores onde o desempenho melhora, sendo esta dependente do domínio em questão. A Seção 4.5 apresentou os resultados experimentais entre o algoritmo proposto e quatro algoritmos atuais. Os resultados foram favoráveis ao algoritmo proposto, uma vez que este foi melhor, em relação à média do erro, entre os demais algoritmos. De fato, o teste estatístico mostrou que o algoritmo KAOGINC é superior a dois dos quatro algoritmos testados e, com relação aos demais, o número de

domínios não foi suficiente para diferenciá-los.

Como conclusão geral pode-se dizer que este trabalho mostrou que o uso de grafos merece ser mais explorado no aprendizado supervisionado. Ambos os algoritmos propostos apresentaram ótimo desempenho nas tarefas a que foram aplicados. Em especial, o algoritmo estático KAOG, apresentou resultados comparados a algoritmos considerados clássicos na classificação multiclasse, o que não é tarefa trivial para um classificador. Ressaltando o fato que o algoritmo não possui parâmetros, e dessa forma, evita a fase de seleção de modelo em aplicações práticas. Com relação ao algoritmo incremental, KAOGINC, os resultados obtidos foram mais promissores, apesar de testado em poucos conjuntos de dados, ainda assim o algoritmo apresentou o melhor resultado dentre os algoritmos testados. Os resultados experimentais, juntamente com as análises estatísticas, justificam a pesquisa futura e a aplicação dos algoritmos em outros domínios de dados.

5.1 Principais conclusões e contribuições do trabalho

Com base nos resultados de classificação obtidos utilizando os algoritmos propostos neste trabalho, podem-se estabelecer as seguintes conclusões:

- A abstração de conjuntos de dados como grafos consiste em uma poderosa forma de representação de dados. Este tipo de estrutura carrega informações não só de vizinhos mais próximos, mas também informações de vértices em outros níveis de vizinhança. Além disso, permite modelar os dados de modo a evidenciar as estruturas presentes na distribuição dos dados, como agrupamentos ou áreas esparsas de dados.

- Métodos baseados em grafos são extensamente considerados em aplicações de aprendizado de máquina, como no agrupamento de dados e na classificação semissupervisionada - principalmente na transdução. Métodos supervisionados têm sido muito pouco pesquisados sob a abordagem de grafos na literatura. Este trabalho propôs dois métodos para solucionar problemas de classificação, em especial para dados vetoriais, com distribuição estacionária e não estacionária. O uso deste tipo de representação mostrou-se apropriado para ser usado em aplicações de classificação de dados. Por meio das análises ao longo do texto, nota-se que algoritmos de classificação baseado em grafos têm varias vantagens e, portanto é promissora.

- A medida de pureza tem como principal objetivo quantificar a mistura nos dados de um componente. No entanto, esta também pode ser usada para caracterizar classes de um conjunto de treinamento. A caracterização da estrutura de uma determinada classe, como distribuição, nível de mistura com outras classes, formato, entre outras, é importante não somente no domínio de classificação, mas também em outras aplicações como na visualização de dados (Köhler *et al.*, 2006). De forma que, a medida de pureza descrita neste trabalho é um passo nesta direção.

- Como visto no Capítulo 4, o classificador incremental baseado no grafo ótimo obteve excelentes resultados na classificação de dados não estacionários. Este fato mostra que o uso de grafos não só possibilita bom desempenho na classificação de dados com distribuição estacionária, mas também consiste em uma forma de representação que permite atualização rápida atualização do grafo e dos conceitos e consequentemente, na retomada de desempenho.

- Redes complexas constitui uma área de pesquisa extensamente estudada nos últimos anos. Porém, a maioria dos resultados focalizam a análise estrutural ou aspectos dinâmicos de redes já existente ou artificialmente construídas. De forma que, pouca atenção tem sido dada à transformação de outro tipo de dados para redes. Este trabalho é uma tentativa nesta direção, contribuindo, desta forma, com a pesquisa em redes complexas.

Em vista do desenvolvimento das duas abordagens para classificação de dados com diferentes distribuições, bem como, dos resultados obtidos, as principais contribuições do trabalho foram:

- Contribuir com a área de pesquisa de métodos de classificação basedos em grafos.

- Proposta de uma nova técnica de formação do grafo, a partir de um conjunto de dados vetoriais, facilitando a análise de dados.

- Aprofundamento de análise da medida de pureza, usada para quantificar o nível de sobreposição entre diferentes classes.

- Desenvolvimento de um método não paramétrico para classificação de dados multi-classe.

- Desenvolvimento de um algoritmo de aprendizado incremental;

5.2 Sugestões para pesquisas futuras

Como este trabalho apresentou dois tipos de classificadores, pode-se elencar algumas sugestões de pesquisa referente a cada um deles.

Classificador estático: Algumas sugestões de pesquisa referente ao algoritmo de aprendizado, KAOG, são:

- Utilizar diferentes medidas de similaridade, bem como, medidas que permitam ao classificador KAOG lidar com atributos nominais.

- Modificar o algoritmo para que possa se aplicado na classificação de dados relacionais.

- Derivar variações para a medida de pureza, por exemplo considerar duas camadas de vizinhos para cada vértice, ou considerar pesos no cálculo, etc.
- Utilizar outras maneiras de construir o grafo, por exemplo, usar a laplaciana da matriz de adjacência na construção do grafo otimo.
- Desenvolver um algoritmo que utilize a pureza, ou alguma variação desta, para aprender também com dados não rotulados.

Classificador incremental: As sugestões de pesquisa futura para o algoritmo incremental devem incluir:

- Aprendizado em tempo real, no qual o classificador precisa processar (incorporar conhecimento ou classificar) um único dado assim que este é apresentado. O classificador atual, quando de posse de único dado rotulado, o insere como um componente isolado. Pesquisas futuras deverão considerar adicioná-lo em componentes já existentes, recalculado a pureza e atualizando os parâmetros.
- Desenvolvimento de solução para diminuir o tempo exigido pela poda em situações reais. Detectar a mudança de conceito e ativar o mecanismo de poda, desativando-o quando o classificador retomar níveis aceitáveis de desempenho.
- Aplicação de algoritmos avançados que possam diminuir o tempo de processamento do algortimo KAOGINC, como algoritmos para determinar os vizinhos mais próximos com ordem de complexidade menor.
- Desenvolvimento de um algoritmo capaz de incluir conhecimentos provindos de dados rotulados e não rotulados no contexto de classificação em tempo real. Apenas recentemente métodos para tratar fluxo de dados sob a perspectiva do aprendizado semissupervisionado, têm sido propostos. Entretanto, esse métodos, em sua maioria, consistem em aplicações para algum problema específico, como classificação de trafego (Erman *et al.*, 2007) e detecção de intrusos (Yu *et al.*, 2008) em redes de computadores.

Referências Bibliográficas

Aha *et al.***(1991)** D. Aha, D. Kibler, e M. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66. Citado na pág. 45

Albert e Barabási(2002) R. Albert e A.-L. Barabási. Statistical mechanics of complex networks. *Review of Modern Physics*, 74:47–97. Citado na pág. 2, 15

Allen e Mills(2004) R. Allen e D. Mills. *Signal Analysis*: *Time, Frequency, Scale, and Structure*. IEEE Press, 1 ºedição. Citado na pág. 117

Allwein *et al.***(2000)** E. Allwein, R. Schapire, e Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1: 113–141. Citado na pág. 26

Angiulli e Pizzuti(2005) F. Angiulli e C. Pizzuti. Outlier mining in large high-dimensional data sets. *IEEE Transactions on Knowledge and Data Engineering*, 17: 203–215. Citado na pág. 23

Asuncion e Newman(2007) A. Asuncion e D. Newman. UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences, 2007. http://www.ics.uci.edu/∼mlearn/MLRepository.html. Citado na pág. 56, 73, 78, 106

Belkin e Niyogi(2003) M. Belkin e P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396. Citado na pág. 1, 18

Belkin e Niyogi(2004) M. Belkin e P. Niyogi. Semi-supervised learning on manifolds. *Machine Learning*, 56:209–239. Citado na pág. 3, 18

Belkin *et al.***(2004)** M. Belkin, I. Matveeva, e P. Niyogi. Regularization and semi-supervised learning on large graphs. Em *Proceedings of the Conference on Computational Learning Theory*, volume 8, páginas 624–638. Citado na pág. 19

Belkin *et al.***(2005)** M. Belkin, P. Niyogi, e V. Sindhwani. On manifold regularization. Em *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, páginas 17–24. Citado na pág. 19

Belkin *et al.***(2006)** M. Belkin, P. Niyogi, e V. Sindhwani. Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 1:1–48. Citado na pág. 21

Bertini Jr. *et al.***(2011a)** J. Bertini Jr., L. Zhao, e A. Lopes. A graph-based algorithm for non-stationary multi-class classification. *em submissão à Pattern Recognition Letters*. Citado na pág. 85

Bertini Jr. *et al.***(2011b)** J. Bertini Jr., L. Zhao, R. Motta, e A. Lopes. A graph-based algorithm for multi-class classification. *versão revisada em submissão à Information Sciences*. Citado na pág. 55

Bishop(2006) C. Bishop. *Pattern recognition and machine learning*. Springer. Citado na pág. 22

Blatt *et al.***(1973)** M. Blatt, S. Wiseman, e E. Donamy. Information theory and an extension of the maximum likelihood principle. Em *Proceedings of the International Symposium on Information Theory*, páginas 267–281. Citado na pág. 27

Blatt *et al.***(1996)** M. Blatt, S. Wiseman, e E. Donamy. Clustering data through an analogy to the potts model. Em *Advances in Neural Information Processing System*, volume 8, páginas 416–422. Citado na pág. 17

Boccaletti *et al.***(2002)** S. Boccaletti, J. Kurths, G. Osipov, D. Valladares, e C. Zhou. The synchronization of chaotic systems. *Physics Reports*, 366:1–101. Citado na pág. 16

Boccaletti *et al.***(2007)** S. Boccaletti, M. Ivanchenko, V. Latora, A. Pluchino, e A. Rapisarda. Detecting complex network modularity by dynamical clustering. *Physical Review E*, 75:1–4. Citado na pág. 16

Boley *et al.***(1999)** D. Boley, M. Gini, R. Gross, E.-H. Han, G. Karypis, V. Kumar, B. Mobasher, J. Moore, e K. Hastings. Partitioning-based clustering for web document categorization. *Decision Support Systems*, 27:329–341. Citado na pág. 14

Bollobás(1998) B. Bollobás. *Modern graph theory*. Springer, 1ºedição. Citado na pág. 9

Braga-Neto e Dougherty(2004) U. Braga-Neto e E. Dougherty. Is cross-validation valid for small-sample microarray classification? *Bioinformatics*, 20:374–380. Citado na pág. 30

Breiman(1996) L. Breiman. Stacked regressions. *Machine Learning*, 24:49–64. Citado na pág. 25

Breiman *et al.***(1984)** L. Breiman, J. Friedman, R. Olshen, e C. Stone. *Classification and Regression Trees*. Chapman and Hall, 1ºedição. Citado na pág. 28, 35

Brodley e Friedl(1996) C. Brodley e M. Friedl. Identifying and eliminating mislabeled training instances. Em *Proceedings of the National American Conference on Artificial Intelligence*, páginas 799–805. Citado na pág. 23

Callut *et al.***(2008)** J. Callut, K. Françoisse, M. Saerens, e P. Dupont. Semi-supervised classification from discriminative random walks. Em *Lecture Notes on Artificial Inteligence*, volume 5211, páginas 162–177. Springer-Verlag. Citado na pág. 18

Capoccia *et al.***(2005)** A. Capoccia, V. Servedioa, G. Caldarellia, e F. Colaiorib. Detecting communities in large networks. *Physica A*, 352:669–676. Citado na pág. 2

Chakrabarti *et al.***(1998)** S. Chakrabarti, B. Dom, e P. Indyk. Enhanced hypertext categorization using hyperlinks. Em *Proceedings of the International Conference on Management of Data*, páginas 307–319. ACM Press. Citado na pág. 21

Chapelle *et al.*(**2006**) O. Chapelle, A. Zien, B. Schölkopf, e (Eds.). *Semi-supervised Learning*. MIT Press, 1ºedição. Citado na pág. 1, 3, 18, 20

Chen *et al.*(**2009a**) H. Chen, L. Li, e J. Peng. Error bounds of multi-graph regularized semi-supervised classification. *Information Sciences*, 179:1960–1969. Citado na pág. 3, 21

Chen *et al.*(**2009b**) J. Chen, H. Fang, e Y. Saad. Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection. *Journal of Machine Learning Research*, 10:1989–2012. Citado na pág. 73

Clauset(**2005**) A. Clauset. Finding local community structure in networks. *Physical Review E*, 72:1–6. Citado na pág. 15

Cogger(**2010**) K. Cogger. Nonlinear multiple regression methods: a survey and extensions. *Intelligent Systems in Accounting, Finance & Management*, 17:19–39. Citado na pág. 1

Cohen *et al.*(**2008**) L. Cohen, G. Avrahami-Bakish, M. Last, A. Kandel, e O. Kipersztok. Real-time data mining of non-stationary data streams from sensor networks. *Information Fusion*, 9:344–353. Citado na pág. 4

Cormen *et al.*(**2009**) T. Cormen, C. Leiserson, R. Rivest, e C. Stein. *Introduction to Algorithms*. MIT Press, 3ºedição. Citado na pág. 71

Culp e Michailidis(**2008**) M. Culp e G. Michailidis. Graph-based semisupervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:174–179. Citado na pág. 3, 18

Danon *et al.*(**2005**) L. Danon, J. Duch, A. Arenas, e A. Díaz-Guilera. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, P09008:1–10. Citado na pág. 16

Delalleau *et al.*(**2005**) O. Delalleau, Y. Bengio, e N. Roux. Efficient non-parametric function induction in semi-supervised learning. Em *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, páginas 96–103. Citado na pág. 3, 20

Demšar(**2006**) J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30. Citado na pág. 81, 121

Diestel(**2006**) R. Diestel. *Graph theory*. Springer, 3ºedição. Citado na pág. 9

Domingos e Hulten(**2000**) P. Domingos e G. Hulten. Mining high-speed data stream. Em *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, páginas 71–80. ACM N.Y. Citado na pág. 45

Dorogovtsev e Mendes(**2003**) S. Dorogovtsev e J. Mendes. *Evolution of Networks: From Biological Nets to the Internet and WWW*. Oxford University Press, 1ºedição. Citado na pág. 2

Duda *et al.*(**2001**) R. Duda, P. Hart, e D. Stork. *Pattern Classification*. John Wiley & Sons, Inc, 2ºedição. Citado na pág. 1, 22, 24, 32

Efron(**1983**) B. Efron. Estimating the error rate of a prediction rule: improvement on cross-validation. *Journal of the American Statistical Association*, 78:316–331. Citado na pág. 28

Efron e Tibshirani(1997) B. Efron e R. Tibshirani. Improvements on cross-validation: the .632+ bootstrap method. *Journal of the American Statistical Association*, 92:548–560. Citado na pág. 28, 30

Elio e Watanabe(1991) R. Elio e L. Watanabe. An incremental deductive strategy for controlling constructive induction in learning from examples. *Machine Learning*, 7: 7–44. Citado na pág. 45

Erman *et al.***(2007)** J. Erman, A. Mahanti, M. Arlitt, I. Cohen, e C. Williamson. Offline/realtime traffic classification using semi-supervised learning. *Performance Evaluation*, 64:1194–1213. Citado na pág. 129

Esposito *et al.***(1997)** F. Esposito, D. Malerba, e G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:476–491. Citado na pág. 51

Fan(2004) W. Fan. Systematic data selection to mine concept-drifting data streams. Em *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, páginas 128–137. ACM Press. Citado na pág. 108

Fern e Givan(2003) A. Fern e R. Givan. Online ensemble learning: an empirical study. *Machine Learning*, 53:71–109. Citado na pág. 25

Figueira *et al.***(2006)** L. Figueira, L. Palma Neto, J. Bertini Jr., e M. Nicoletti. Using constructive neural networks for detecting central vestibular system lesion. *Applied Artificial Intelligence*, 20:609–638. Citado na pág. 37

Filippone *et al.***(2008)** M. Filippone, F. Camastra, F. Masulli, e S. Rovetta. A survey of kernel and spectral methods for clustering. *Pattern Recognition*, 41:176–190. Citado na pág. 18

Filzmoser *et al.***(2009)** P. Filzmoser, B. Liebmann, e K. Varmuza. Repeated double cross validation. *Journal of Chemometrics*, 23:160–171. Citado na pág. 30, 78

Franco *et al.***(2009)** L. Franco, D. Elizondo, e J. Jerez (Eds.). *Constructive Neural Networks*. Springer, 1ºedição. Citado na pág. 37

Friedman(1937) M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32: 675–701. Citado na pág. 121

Friedman(1940) M. Friedman. A comparison of alternative test of significance for the problem of m rankings. *Annals of Mathematical Statistics*, 11:86–92. Citado na pág. 121

Fu e Anderson(1986) Y. Fu e P. Anderson. Application of statistical mechanics to np-complete problems in combinatorial optimization. *Journal of Physics A: Mathematical and General*, 19:1605–1620. Citado na pág. 17

Gama *et al.***(2003)** J. Gama, R. Rocha, e P. Medas. Accurate decision trees for mining high-speed data streams. Em *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, páginas 523–528. ACM Press. Citado na pág. 45

Gama *et al.***(2004)** J. Gama, P. Medas, G. Castillo, e P. Rodrigues. Learning with drift detection. Em *Proceedings of the Brazilian Symposium on Artificial Intelligence*, volume 3171, páginas 286–295. Springer. Citado na pág. 106, 115

Gehrke *et al.***(1999)** J. Gehrke, V. Ganti, R. Ramakrishnan, e W.-Y. Loh. BOAT: optimistic decision tree construction. Em *Proceedings of the International Conference on Management of Data*, páginas 169–180. Citado na pág. 35

Gehrke *et al.***(2000)** J. Gehrke, R. Ramakrishnan, e V. Ganti. Rainforest - a framework for fast decision tree construction of large datasets. *Data Mining and Knowledge Discovery*, 4:127–162. Citado na pág. 35

Giordano *et al.***(2008)** R.C. Giordano, J. Bertini Jr., M. Nicoletti, e R.L. Giordano. Online filtering of co_2 signals from a bioreactor gas outflow using a committee of constructive neural networks. *Bioprocess and Biosystems Engineering*, 31:101–109. Citado na pág. 37

Giraud-Carrier(2000) C. Giraud-Carrier. A note on the utility of incremental learning. *AI Communications*, 13:215–223. Citado na pág. 4, 44, 86

Gross(2005) J. Gross. *Graph theory and its applications*. Chapman and Hall/CRC, 2ºedição. Citado na pág. 9

Guha *et al.***(1998)** S. Guha, R. Rastogi, e K. Shim. CURE: an efficient clustering algorithm for large databases. Em *Proceedings of the International Conference on Management of Data*, páginas 73–84. Citado na pág. 15

Guyon e Elisseeff(2003) I. Guyon e A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182. Citado na pág. 23

Han e Kamber(2006) J. Han e M. Kamber. *Data Mining*: *Concepts and Techniques*. Morgan Kaufmann, 2ºedição. Citado na pág. 1

Harries(1999) M. Harries. Splice-2 comparative evaluation: electriciy pricing. Relatório Técnico UNSW-CSE-TR-9905, Artificial Intelligence Group, School of Computer Science and Engineering, The University of South Wales, Australia. Citado na pág. 115

Hartuv e Shamir(2000) E. Hartuv e R. Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76:175–181. Citado na pág. 2

Hastie *et al.***(2009)** T. Hastie, R. Tibshirani, e J. Friedman. *The Elements of Statistical Learning*: *Data Mining, Inference and Prediction*. Springer-Verlag, 2ºedição. Citado na pág. 1, 3, 20, 27, 29, 33, 34, 64, 78

Haykin(2008) S. Haykin. *Neural Networks and Learning Machines*. Prentice Hall, 3ºedição. Citado na pág. 37

Hein *et al.***(2007)** M. Hein, J-Y. Audibert, e U. von Luxburg. Graph laplacians and their convergence on random neighborhood graphs. *Journal of Machine Learning Research*, 8:1325–1368. Citado na pág. 2, 18

Helmbold e Long(1994) D. Helmbold e P. Long. Tracking drifting concepts by minimizing disagreements. *Machine Learning*, 14:27–45. Citado na pág. 4

Hernandez-Rodriguez *et al.***(2010)** S. Hernandez-Rodriguez, J. Martinez-Trinidad, e J. Carrasco-Ochoa. Fast k most similar neighbor classifier for mixed data (tree k-msn). *Pattern Recognition*, 43:873–886. Citado na pág. 73

Hettich e Bay(1999) S. Hettich e S. Bay. The UCI KDD archive. University of California, Irvine, School of Information and Computer Sciences, 1999. http://kdd.ics.uci.edu/. Citado na pág. 106

Hodge e Austin(2004) V. Hodge e J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22:85–126. Citado na pág. 60

Hruschka *et al.***(2009)** E. Hruschka, A. Garcia, E. Hruschka Jr, e N. Ebecken. On the influence of imputation in classification: practical issues. *Journal of Experimental & Theoretical Artificial Intelligence*, 21:43–58. Citado na pág. 23

Hsu e Lin(2002) C-W. Hsu e C-J. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13:415–425. Citado na pág. 26, 78, 79

Hu *et al.***(2005)** H. Hu, X. Yan, Y. Huang, J. Han, e X. Zhou. Mining coherent dense subgraphs across massive biological networks od functional discovery. *Bioinformatics*, 1:213–221. Citado na pág. 2

Hulten *et al.***(2001)** G. Hulten, L. Spencer, e P. Domingos. Mining time-changing data streams. Em *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, páginas 97–106. ACM Press. Citado na pág. 106, 108

Iman e Davenport(1980) R. Iman e J. Davenport. Approximations of the critical region of the Friedman statistic. *Communications in Statistics*, 9:571–595. Citado na pág. 121

Jain e Dubes(1998) A. Jain e R. Dubes. *Algorithms for clustering data*. Prentice Hall, 1ºedição. Citado na pág. 14

Jain *et al.***(1999)** A. Jain, M. Murty, e P. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31:264–323. Citado na pág. 1, 13

Jank e Shmueli(2005) W. Jank e G. Shmueli. Profiling price dynamics in online auctions using curve clustering. Relatório técnico, Smith School of Business, University of Maryland. Citado na pág. 14

Jiang *et al.***(2004)** D. Jiang, C. Tang, e A. Zhang. Cluster analysis for gene expression data: a survey. *IEEE Transactions on Knowledge and Data Engineering*, 16:1370–1386. Citado na pág. 14

Joachims(2003) T. Joachims. Transductive learning via spectral graph partitioning. Em *Proceedings of the International Conference on Machine Learning)*, páginas 290–297. Citado na pág. 19

Kannan *et al.***(2004)** R. Kannan, S. Vempala, e A. Vetta. On clusterings - good, bad and spectral. *Journal of the ACM*, 51:497–515. Citado na pág. 2

Karypis *et al.***(1999)** G. Karypis, E.-H. Han, e V. Kumar. Chameleon: hierarchical clustering using dynamic modeling. *IEEE Computer*, 32:68–75. Citado na pág. 2, 14, 15

Kelly *et al.***(1999)** M. Kelly, D. Hand, e N. Adams. The impact of changing populations on classifier performance. Em *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, páginas 367–371. ACM N.Y. Citado na pág. 42

Kim(2009) J-H. Kim. Estimating classification error rate: repeated cross-validation, repeated hold-out and bootstrap. *Computational Statistics and Data Analysis*, 53: 3735–3745. Citado na pág. 29, 30

Klinkenberg e Joachims(2000) R. Klinkenberg e T. Joachims. Detecting concept drift with suport vector machines. Em *Proceedings of the International Conference on Machine Learning*, páginas 487–494. Morgan Kaufmann. Citado na pág. 43

Köhler *et al.***(2006)** J. Köhler, J. Baumbach, J. Taubert, M. Specht, A. Skusa, A. Rüegg, C. Rawlings, P. Verrier, e S. Philippi. Graph-based analysis and visualization of experimental results with ONDEX. *Bioinformatics*, 22:1383–1390. Citado na pág. 127

Kohonen(1989) T. Kohonen. *Self-Organization and Associative Memory*. Springer, 3ºedição. Citado na pág. 34

Kolter e Maloof(2007) J. Kolter e M. Maloof. Dynamic weighted majority: an ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8:2755–2790. Citado na pág. 48, 106

Krishnapuram *et al.***(2005)** B. Krishnapuram, D. Williams, Y. Xue, A. Hartemink, L. Carin, e M. Figueiredo. On semi-supervised classification. Em *Advances in Neural Information Processing System*, volume 17, páginas 721–728. Citado na pág. 20

Kubat e Krizakova(1992) M. Kubat e I. Krizakova. Forgetting and aging of knowledge in concept formation. *Applied Artificial Intelligence*, 6:195–206. Citado na pág. 45

Kulis *et al.***(2009)** B. Kulis, S. Basu, I. Dhillon, e R. Mooney. Semi-supervised graph clustering: a kernel approach. *Machine Learning*, 74:1–22. Citado na pág. 2

Lahnajärvi *et al.***(2002)** J. Lahnajärvi, M. Lehtokangas, e J. Saarinen. Evaluation of constructive neural networks with cascaded architectures. *Neurocomputing*, 48:573–607. Citado na pág. 37

Lane e Brodley(1998) T. Lane e C. Brodley. Approaches to online learning and concept drift for user identification in computer security. Em *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, páginas 259–263. Citado na pág. 44

Last(2002) M. Last. Online classification of nonstationary data streams. *Intelligent Data Analysis*, 6:129 – 147. Citado na pág. 4, 44

Li e Tian(2007) X. Li e Z. Tian. Optimum cut-based clustering. *Signal Processing*, 87: 2491–2502. Citado na pág. 14

Liu e Yu(2005) H. Liu e L. Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17:491–502. Citado na pág. 23

Liu *et al.***(2002)** X. Liu, G. Cheng, e J. Wu. Analysing outliers cautiously. *IEEE Transactions on Knowledge and Data Engineering*, 14:432–437. Citado na pág. 23

Lloyd(1982) S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:128–137. Citado na pág. 34

Lopes *et al.***(2009)** A. Lopes, J. Bertini Jr., R. Motta, e L. Zhao. Classification based on the optimal k-associated network. Em *Proceedings of the International Conference on Complex Sciences*: *Theory and Applications (COMPLEX'09)*, volume 4 of *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, páginas 1167–1177. Springer. Citado na pág. 55

Lu e Getoor(2003) Q. Lu e L. Getoor. Link-based classification. Em *Proceedings of the International Conference on Machine Learning*, páginas 496–503. Citado na pág. 21

MacQueen(1967) J. MacQueen. Some methods for classifcation and analysis of multivariate observations. Em *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, páginas 281–297. University of California Press. Citado na pág. 14, 34, 53

Macskassy e Provost(2003) S. Macskassy e F. Provost. A simple relational classifier. Em *Proceedings of the International Conference on Knowledge Discovery and Data Mining, Workshop on Multi-Relational Data Mining*, páginas 64–76. Citado na pág. 21

Macskassy e Provost(2007) S. Macskassy e F. Provost. Classification in networked data: a toolkit and a univariate case study. *Journal of Machine Learning Research*, 8: 935–983. Citado na pág. 21

Madigan e Raftery(1994) D. Madigan e A. Raftery. Model selection and accounting for model uncertainty using occam's window. *Journal of the American Statistical Association*, 89:35–46. Citado na pág. 27

Maloof e Michalski(2000) M. Maloof e R. Michalski. Selecting examples for partial memory learning. *Machine Learning*, 41:27–52. Citado na pág. 45

Maloof e Michalski(2004) M. Maloof e R. Michalski. Incremental learning with partial instance memory. *Artificial Intelligence*, 154:95–126. Citado na pág. 45, 46

Markovitch e Rosenstein(2002) S. Markovitch e D. Rosenstein. Feature generation using general constructor functions. *Machine Learning*, 49:59–98. Citado na pág. 23

Martínes e Kak(2001) A. Martínes e A. Kak. Pca versus lda. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:228–233. Citado na pág. 23

Michie *et al.***(1994)** D. Michie, D. Spiegelhalter, e C. Taylor (Eds.). *Machine Learning, Neural and Statistical Classifcation*. Ellis Horwood Series in Artificial Intelligence. Ellis Horwood, 1ºedição. Citado na pág. 33

Minku *et al.***(2010)** L. Minku, A. White, e X. Yao. The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 22:730–742. Citado na pág. 43, 44

Mitchell(1997) T. Mitchell. *Machine Learning*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1ºedição. Citado na pág. 1, 22, 31, 35, 38

Müller *et al.***(2001)** K-R. Müller, S. Mika, G. Ratsch, K. Tsuda, e B. Scholkopf. An introduction to kernel-based learning algorithms. *IEEE Transaction on Neural Networks*, 12:181–202. Citado na pág. 40

Murata *et al.***(2002)** N. Murata, M. Kawanabe, A. Ziehe, K.-R. Müller, e S.-I. Amari. On-line learning in changing environments with applications in supervised and unsupervised learning. *Neural Networks*, 15:743–760. Citado na pág. 44

Narasimhamurthy e Kuncheva(2007) A. Narasimhamurthy e L. Kuncheva. A framework for generating data to simulate changing environments. Em *Proceedings of the International Artificial Intelligence and Applications*, páginas 384–389. Citado na pág. 42

Neville *et al.***(2003)** J. Neville, D. Jensen, L. Friedland, e M. Hay. Learning relational probability trees. Em *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, páginas 625–630. Citado na pág. 21

Newman(2003) M. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256. Citado na pág. 2, 15

Newman(2004) M. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69:066133. Citado na pág. 16

Newman e Girvan(2004) M. Newman e M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:1–15. Citado na pág. 2, 15, 16

Ng e Han(2002) R. Ng e J. Han. CLARANS: a method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14:1003–1016. Citado na pág. 14

Nicoletti e Bertini Jr.(2007) M. Nicoletti e J. Bertini Jr. An empirical evaluation of constructive neural network algorithms in classification tasks. *International Journal of Innovative Computing and Applications*, 1:2–13. Citado na pág. 37

Núnez *et al.***(2007)** M. Núnez, R. Fidalgo, e R. Morales. Learning in environments with unknown dynamics: towards more robust concept learners. *Journal of Machine Learning Research*, 8:2595–2628. Citado na pág. 51, 107, 116

Opitz e Maclin(2000) D. Opitz e R. Maclin. Popular ensemble methods: an empirical study. *Journal of Machine Learning Research*, 11:169–198. Citado na pág. 25

Palma Neto e Nicoletti(2005) L. Palma Neto e M. Nicoletti. *Introdução às Redes Neurais Construtivas*. EdUFSCar, 1ºedição. Citado na pág. 37

Parekh *et al.***(2000)** R. Parekh, J. Yang, e V. Honavar. Constructive neural network learning for pattern classification. *IEEE Transactions on Neural Networks*, 11:436–451. Citado na pág. 37

Polikar *et al.***(2004)** R. Polikar, L. Udpa, S. Udpa, e V. Honavar. An incremental learning algorithm with confidence estimation for automated identification of nde signals. *IEEE transactions on ultrasonics, ferroelectrics, and frequency control*, 51:990–1001. Citado na pág. 44

Quinlan(1986) J. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106. Citado na pág. 35

Quinlan(1996) J. Quinlan. Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence*, 4:77–90. Citado na pág. 35

Quinlan(1993) J. Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kaufmann, 1ºedição. Citado na pág. 35, 78

Reed(1993) R. Reed. Pruning algorithm - a survey. *IEEE Transactions on Neural Networks*, 4:740–747. Citado na pág. 37

Reed e Marks II(1999) R. Reed e R. Marks II. *Neural Smithing*: *Supervised Learning in Feedforward Artificial Neural Networks*. The MIT Press, 1ºedição. Citado na pág. 39

Reichardt e Bornholdt(2004) J. Reichardt e S. Bornholdt. Detecting fuzzy community structures in complex networks with a potts model. *Physical Review Letters*, 93:1–4. Citado na pág. 17

Reichardt e Bornholdt(2006) J. Reichardt e S. Bornholdt. Statistical mechanics of community detection. *Physical Review E*, 74:1–14. Citado na pág. 15

Reinke e Michalski(1988) R. Reinke e R. Michalski. Incremental learning of concept descriptions: a method and experimental results. Em J. Hayes, D. Michie, e J. Richards, editors, *Machine Intelligence 11*, páginas 263–288. Oxford Clarendon Press, 1ºedição. Citado na pág. 45

Rissanen(1983) J. Rissanen. A universal prior for integers and estimation by minimum description length. *Annals of Statistics*, 11:416–431. Citado na pág. 27

Rosenblatt(1958) F. Rosenblatt. The percepton: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408. Citado na pág. 38

Saad(1999) D. Saad. *On-Line Learning in Neural Networks*. Cambridge University Press, 1ºedição. Citado na pág. 45

Sarle(1995) W. Sarle. Stopped training and other remedies for overfitting. Em *Proceedings of the Symposium on the Interface of Computing Science and Statistics*, páginas 352–360. Citado na pág. 24

Schaeffer(2007) S. Schaeffer. Graph clustering. *Computer Science Review*, 1:27–34. Citado na pág. 2, 13, 15

Schaffer *et al.***(1992)** J. Schaffer, D. Whitely, e L. Eshelman. Combinations of genetic algorithms and neural networks: a survey of the state of the art. Em *Proceedings of the International Workshop of Genetic Algorithms and Neural Networks*, páginas 1–37. Citado na pág. 37

Schapire(1990) R. Schapire. The strength of weak learnability. *Machine Learning*, 5: 197–227. Citado na pág. 25

Schlimmer e Granger(1986) J. Schlimmer e R. Granger. Beyond incremental processing: tracking concept drift. Em *Proceedings of the National Conference on Artificial Intelligence*, páginas 502–507. AAAI Press. Citado na pág. 42, 45

Scholz e Klinkenberg(2007) M. Scholz e R. Klinkenberg. Boosting classifiers for drifiting concepts. *Intelligent Data Analysis*, 11:3–28. Citado na pág. 105

Schwarz(1978) G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464. Citado na pág. 27

Segal *et al.***(2003)** E. Segal, H. Wang, e D. Koller. Discovering molecular pathways from protein interaction and gene expression data. *Bioinformatics*, 19:264–276. Citado na pág. 21

Shawne-Taylor e Cristianini(2004) J. Shawne-Taylor e N. Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, 1ºedição. Citado na pág. 39, 40

Simard *et al.***(1993)** P. Simard, Y. Cun, e J. Denker. Eficient pattern recognition using a new transformation distance. Em *Advances in Neural Information Processing Systems*, páginas 50–58. Morgan Kaufmann. Citado na pág. 33

Sindhwani *et al.***(2005)** V. Sindhwani, P. Niyogi, e M. Belkin. Beyond the point cloud: from transductive to semi-supervised learning. Em *Proceedings of the International Conference on Machine learning*, páginas 824–831. Citado na pág. 20, 21

Street e Kim(2001) N. Street e Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. Em *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, páginas 377–382. ACM N.Y. Citado na pág. 46, 95, 99, 106

Strogatz(2001) S. Strogatz. Exploring complex networks. *Nature*, 410:268–276. Citado na pág. 2, 15

Syed *et al.***(1999)** N. Syed, H. Liu, e K. Sung. Handling concept drift in incremental learning with suport vector machines. Em *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, páginas 272–276. ACM N.Y. Citado na pág. 42

Szummer e Jaakkola(2002) M. Szummer e T. Jaakkola. Partially labeled classification with markov random walks. Em *Advances in Neural Information Processing Systems*, volume 14, páginas 945–952. MIT Press. Citado na pág. 18

Taxt e Lundervold(1994) T. Taxt e A. Lundervold. Multispectral analysis of the brain using magnetic resonance imaging. *IEEE Transactions on Medical Imaging*, 13:470–481. Citado na pág. 14

Theodoridis e Koutroumbas(2008) S. Theodoridis e K. Koutroumbas. *Pattern recognition*. Elsevier, 4ºedição. Citado na pág. 4

Toyama *et al.***(2010)** J. Toyama, M. Kudo, e H. Imai. Probably correct k-nearest neighbor search in high dimensions. *Pattern Recognition*, 43:1361–1372. Citado na pág. 73

Tsang e Kwok(2007) I. Tsang e J. Kwok. Large-scale sparsified manifold regularization. Em *Advances in Neural Information Processing Systems*, volume 19, páginas 1401–1408. MIT Press. Citado na pág. 19

Tsymbal *et al.***(2006)** A. Tsymbal, M. Pechenizkiy, P. Cunningham, e S. Puuronen. Handling local concept drift with dynamic integration of classifiers: domain of antibiotic resistance in nosocomial infections. Em *Proceedings of the International Symposium on Computer-Based Medical System*, páginas 56–68. IEEE Press. Citado na pág. 105

Utgoff(1988) P. Utgoff. Id5: an incremental id3. Em *Proceedings of the 5th International Conference on Machine Learning*, páginas 107–120. Morgan Kaufman. Citado na pág. 45

Utgoff *et al.***(1997)** P. Utgoff, N. Berkman, e J. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29:5–44. Citado na pág. 45, 48

Vapnik(1999) V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 2ºedição. Citado na pág. 40, 78

Vishwanathan *et al.***(2010)** S. Vishwanathan, N. Schraudolph, R. Kondor, e K. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242. Citado na pág. 18, 22

von Luxburg(2007) U. von Luxburg. A tutorial on spectral clustering. *Statistical Computation*, 17:395–416. Citado na pág. 12

Wang e Zhang(2008) F. Wang e C. Zhang. Label propagation through linear neighborhoods. *IEEE Transactions On Knowledge and Data Engineering*, 20:55–67. Citado na pág. 20

Wang *et al.*(2003) H. Wang, W. Fan, P. Yu, e J. Han. Mining concept drifting data streams using ensemble classifiers. Em *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, páginas 226–235. Citado na pág. 44, 50

Watts e Strogatz(1998) D. Watts e S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442. Citado na pág. 77

West(2000) D. West. *Introduction to Graph Theory*. Prentice-Hall, 2ºedição. Citado na pág. 9

Widmer e Kubat(1996) G. Widmer e M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–101. Citado na pág. 4, 42, 45

Wilson e Martinez(2000) R. Wilson e T. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38:257–286. Citado na pág. 73

Witten e Frank(2005) I. Witten e E. Frank. *Data Mining*: *Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2ºedição. Citado na pág. 49

Wong e Yao(1992) S. Wong e Y. Yao. An information-theoretic measure of term specificity. *Journal of the American Society for Information Science*, 43:54–61. Citado na pág. 2

Yang e Zhou(2008) C. Yang e J. Zhou. Non-stationary data sequence classification using online class priors estimation. *Pattern Recognition*, 41:2656–2664. Citado na pág. 44

Yao(1999) X. Yao. Evolving neural networks. Em *Proceedings of the IEEE*, páginas 1423–1447. Citado na pág. 37

Yu *et al.*(2008) Y. Yu, S. Guo, S. Lan, e T. Ban. Anomaly intrusion detection for evolving data stream based on semi-supervised learning. Em *Proceedings of the International Conference on Advances in neuro-information processing*, páginas 571–578. Citado na pág. 129

Zhao *et al.*(2008) L. Zhao, T. Cupertino, e J. Bertini Jr. Chaotic synchronization in general network topology for scene segmentation. *Neurocomputing*, 71:3360–3366. Citado na pág. 17

Zhou e Schölkopf(2004) D. Zhou e B. Schölkopf. Learning from labeled and unlabeled data using random walks. Em *Proceedings of the DAGM Symposium on Pattern Recognition*, páginas 237–244. Citado na pág. 3, 18

Zhou *et al.*(2004) D. Zhou, T O. Bousquet, Lal, J. Weston, e B. Schölkopf. Learning with local and global consistency. Em *Advances in Neural Information Processing Systems*, páginas 321–328. Citado na pág. 18

Zhou(2003a) H. Zhou. Network landsape from a brownian particle's perspective. *Physical Review E*, 67:041908. Citado na pág. 16

Zhou(2003b) H. Zhou. Distance, dissimilarity index, and network community structure. *Physical Review E*, 67:061901. Citado na pág. 16

Zhu(2008) X. Zhu. Semi-supervised learning literature review. Relatório Técnico 1530, Computer-Science, University of Wisconsin-Madison. Citado na pág. 1, 3, 12, 18

Zhu(2005) X. Zhu. Semi-supervised learning with graphs. Relatório Técnico Doctoral Thesis, School of Computer Science, Carnegie Mellon University. Citado na pág. 13

Zhu e Ghahramani(2002) X. Zhu e Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. Relatório Técnico CMU-CALD-02-107, Carnegie Mellon University, Pittsburgh. Citado na pág. 18

Zhu e Lafferty(2005) X. Zhu e J. Lafferty. Harmonic mixtures: combining mixture models and graph-based methods for inductive and scalable semi-supervised learning. Em *Proceedings of the International Conference on Machine Learning*, páginas 1052–1059. ACM Press. Citado na pág. 3, 20

Zhu *et al.***(2003)** X. Zhu, Z. Ghahramani, e J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. Em *Proceedings of the International Conference on Machine Learning*, páginas 1052–1059. ACM Press. Citado na pág. 19