4
Estimação de Problemas de Desempenho

A necessidade de compreender dados instrumentados, complexos e volumosos está se tornando cada vez mais importante para a área de software em geral. Dado um conjunto de dados coletados e algum conhecimento prévio do problema, é possível construir modelos probabilísticos que representem melhor os dados disponíveis (Trivedi, 2002). Nesse contexto, Teoria da Probabilidade e Inferência Estatística proveem um arcabouço natural para o problema de aprender modelos probabilísticos a partir de dados instrumentados. De acordo com Hastie et al. (Hastie et al., 2001), muitos problemas de aprendizado podem ser resolvidos da forma a seguir. Primeiramente, uma classe de modelo, tal como rede Bayesiana, rede neural ou árvore de decisão, é escolhida para um problema particular. Em seguida, para a classe de modelo escolhida, a estrutura do modelo é selecionada. Essa estrutura define uma classe paramétrica de modelos. Finalmente, os parâmetros são estimados usando os dados disponíveis.

Este capítulo apresenta uma avaliação de diferentes classes de modelos de aprendizado estatísticos aplicados para a estimação de problemas de desempenho. O objetivo do capítulo é identificar as classes de modelo mais apropriadas para responder a questão $\mathbf{Q_1}$ colocada no Capítulo 1: *considerando um conjunto corrente de métricas observadas, quais métodos da teoria do aprendizado estatístico são mais apropriados para prever ou estimar quando um sistema (ou seja, uma aplicação baseada em middleware) não conseguirá atingir seu objetivo num futuro próximo*? Para tanto, o capítulo está organizado da seguinte forma. Inicialmente, são abordados alguns conceitos estatísticos importantes usados neste trabalho. Em seguida, são discutidas algumas classes de modelos tradicionais em aprendizado estatístico, bem como algumas métricas comuns de avaliação de desempenho dos modelos. Finalmente, é apresentado um estudo experimental envolvendo as seguintes classes de modelo: diferente classes de redes Bayesianas, árvore de decisão e *Support Vector Machine* (SVM), ou máquinas de vetores de suporte. Cada classe de modelo é implementada no SMART e avaliada no contexto do cenário de referência descrito no Capítulo 3. Diferentes estruturas e parâmetros são avali-

ados no sentido de determinar o modelo que melhor descreve o comportamento do desempenho de uma aplicação MapReduce.

4.1 Conceitos Básicos

Para as definições a seguir e no restante deste trabalho é usada a seguinte notação. Letras maiúsculas denotam variáveis aleatórias (X, Y). Letras minúsculas denotam valores específicos para essas variáveis (x, y). Letras maiúsculas em negrito denotam conjuntos de variáveis aleatórias ($\mathbf{X}$, $\mathbf{Y}$), enquanto letras minúsculas em negrito denotam atribuições de valores às variáveis desses conjuntos ($\mathbf{x}$, $\mathbf{y}$). As definições apresentadas na Seção 4.1.1 são baseadas principalmente no livro (NIST/SEMATECH, 2011).

4.1.1 Fundamentos de Probabilidade

Em Teoria da Probabilidade, um problema do mundo real é representado através de um conjuntos de variáveis aleatórias $X_1, X_2, ...X_n$, onde cada variável possui um domínio de valores possíveis. Um dos conceitos fundamentais dessa teoria é a *distribuição de probabilidade conjunta*, $P(X_1, X_2, ...X_n)$, a qual especifica uma probabilidade para cada combinação possível de valores para todas as variáveis.

Outro conceito relevante é a noção de *exemplo*. Seja um conjunto $\mathbf{X} = \{X_1, X_2, ..., X_n\}$ de variáveis aleatórias onde cada variável X_i está associada a um domínio Ω_{X_i}. Suponha um experimento envolvendo $\mathbf{X}$. No tempo t, observa-se uma saída particular $\mathbf{x}^{(t)} = \{x_1, x_2, ..., x_n\}$. Denomina-se tal saída um *exemplo*, ou seja, uma instância do vetor aleatório $\mathbf{X}^{(t)}$.

Um conjunto de exemplos forma um *dataset*. Formalmente, denomina-se *dataset* o conjunto $D = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$ resultante de um experimento aleatório no qual N exemplos são amostrados independentemente a partir de uma distribuição de probabilidade $P(X_1, X_2, ...X_n)$ sobre $\mathbf{X}$.

Dado um *dataset*, é possível construir um *modelo probabilístico* que representa os dados em questão. Define-se um modelo probabilístico M para um conjunto $\mathbf{X}$ de variáveis aleatórias como um conjunto de distribuições de probabilidade conjunta, parametrizado por um conjunto de parâmetros $\mathbf{\Theta}$, ou seja,

$$M \equiv P(\boldsymbol{X}|\mathbf{\Theta}). \tag{4-1}$$

Portanto, um modelo probabilístico é um conjunto de distribuições de probabilidade conjunta do mesmo tipo. Dito de outra forma, um modelo pro-

babilístico é uma família de modelos governados por um ou mais parâmetros.

Uma distribuição de probabilidade de M com uma configuração de parâmetros particular é chamada de *hipótese*. Aprender um modelo probabilístico a partir de um *dataset* é encontrar a hipótese que melhor explica os dados presentes no *dataset*.

Neste trabalho, os termos *exemplo* e *instância* são usados como sinônimos quando se referem a uma amostra de um *dataset*. Os termos *hipótese* e *modelo* também são usados de forma indistinta para denotar um modelo probabilístico com uma configuração particular.

4.1.2 Classificação e Aprendizado Supervisionado

Classificação é uma tarefa que ocorre em diversas atividades humanas. De maneira geral, o termo *classificação* refere-se a qualquer contexto no qual alguma decisão ou estimação é feita com base em informações disponíveis correntemente. De acordo com essa definição, um *procedimento de classificação* consiste em um método formal que repetidamente toma decisões diante de novas situações.

Segundo Michie et al. (Michie et al., 1994), existem duas formas distintas de se realizar uma tarefa de classificação. Na primeira abordagem, dado um conjunto de observações, o objetivo é estabelecer a existência de classes nesse conjunto. Essa abordagem é conhecida como *aprendizado não supervisionado*. Na segunda abordagem, as classes são conhecidas e o objetivo é estabelecer a regra que mapeia uma observação a uma das classes existentes. Essa abordagem é denominada *aprendizado supervisionado*. Neste trabalho, o termo *classificação* refere-se apenas a aprendizado supervisionado.

Em um cenário de aprendizado supervisionado, um *classificador* é uma função que atribui um rótulo ou classe para um objeto descrito por um conjunto de atributos. Esse classificador é construído a partir de um conjunto de dados rotulados. Dito de outra forma, num cenário típico de aprendizado supervisionado, existe uma métrica categórica, ou seja, um rótulo, que se deseja estimar baseado em um conjunto de atributos. Existe também um *dataset* a partir do qual é possível observar o rótulo e os valores dos atributos para um conjunto de objetos. Usando o *dataset*, é possível construir um modelo de estimação ou classificador que estima uma classe ou rótulo para novos objetos com uma determinada probabilidade de acerto.

Formalmente, seja um vetor $\mathbf{X} = \{X_1, X_2, ..., X_n\}$ de variáveis aleatórias. Denomina-se *atributo* cada variável X_i. Cada atributo, por sua vez, possui um domínio Ω_{X_i}. Seja C uma variável aleatória não observada, cujos valores

pertencem ao domínio $\Omega_C = \{c_1, c_2, ..., c_m\}$. C é chamada *variável de classe* e os valores de C são denominados *rótulos* ou *classes*. Seja f a função que mapeia o espaço de entrada $\Omega_{\mathbf{X}}$ no espaço de saída Ω_C. Ou seja, f é a função a ser aprendida. Considere também um *dataset* $D = \{< \mathbf{x}^{(1)}, c^{(1)} >, < \mathbf{x}^{(2)}, c^{(2)} >, ..., < \mathbf{x}^{(N)}, c^{(N)} >\}$ de N exemplos rotulados. O problema tratado em aprendizado supervisionado é construir um classificador h_C a partir de D que seja o mais próximo possível de f.

4.2 Classes de Modelos em Aprendizado Supervisionado

Existem diversos algoritmos que se baseiam em aprendizado supervisionado e um estudo comparativo envolvendo muitos desses algoritmos é apresentado por Michie et al. em (Michie et al., 1994). Nesse trabalho, os autores mostram que não existe um único algoritmo de aprendizado supervisionado que se sobressai sobre todos os demais em todos os casos. Ou seja, alguns algoritmos de aprendizado supervisionado podem ser mais adequados que outros em determinados problemas. Portanto, estudos comparativos para determinar o algoritmo mais apropriado para um problema particular é essencial. Nesse contexto, este capítulo apresenta um estudo comparativo entre diferentes classes de modelos de aprendizado supervisionado para o problema da caracterização de desempenho de aplicações baseadas em *middleware*. A seguir, é apresentada uma breve explicação das classes de modelos que são utilizadas neste estudo. Essas classes correspondem aos principais algoritmos de aprendizado supervisionado disponíveis atualmente.

4.2.1 Redes Bayesianas

Dado um conjunto de variáveis aleatórias representando um problema particular, uma rede Bayesiana representa graficamente a distribuição de probabilidade conjunta dessas variáveis. Formalmente, seja um conjunto de variáveis aleatória $\mathbf{X} = \{X_1, X_2, ..., X_n\}$. Uma rede Bayesiana sobre $\mathbf{X}$ é uma tupla $BN = (S,\mathbf{\Theta}_S)$. O primeiro componente da tupla, S, é um grafo direcionado acíclico cujos nós representam as variáveis aleatórias do problema e os arcos representam dependências diretas entre essas variáveis. O segundo componente da tupla, $\mathbf{\Theta}_S = \{\mathbf{\Theta}_1,\mathbf{\Theta}_2, ...,\mathbf{\Theta}_n\}$, é um conjunto de funções de probabilidade condicional, onde cada $\mathbf{\Theta}_i = P(X_i|\mathbf{Pa}_i) \in \mathbf{\Theta}_S$ representa uma função de probabilidade condicional sobre os valores de X_i, dado os valores de seus pais $\mathbf{Pa}_i$. Além disso, a rede satisfaz a condição Markoviana: cada nó é independente de todos os seus não descendentes, dado seus pais em S. Devido

a essa condição, a distribuição de probabilidade conjunta sobre $\mathbf{X}$ pode ser representada de forma compacta pela Equação 4-2:

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}_i). \quad (4\text{-}2)$$

Assumindo-se valores discretos para as variáveis aleatórias, cada $P(X_i|\mathbf{Pa}_i)$ representa uma tabela de probabilidade condicional (*Conditional Probability Table* ou CPT). Cada CPT associada à variável X_i é composta por q_i linhas, uma para cada configuração pai $pa_j \in \Omega_{\mathbf{Pa}_i}$. As entradas em cada linha associada à configuração pa_j representam a probabilidade $P(X_i = x_k|\mathbf{Pa}_i = pa_j)$, para cada valor $x_k \in \Omega_{X_i}$.

Redes Bayesiana como Classificadores

Em um problema de classificação, as variáveis do modelo podem ser separadas em dois grupos: o conjunto dos atributos, ou seja, $\mathbf{X} = \{X_1, X_2, ..., X_n\} \in \Omega_{\mathbf{X}}$, e o conjunto formado apenas pela variável de classe $C \in \Omega_C = \{c_1, c_2, ...c_m\}$. O objetivo é estimar corretamente o valor c da variável C, dado um exemplo $\mathbf{x} = \{x_1, x_2, ..., x_n\}$. Uma rede Bayesiana aplica-se a um problema de classificação de forma natural e direta. Uma das variáveis da rede é selecionada como a variável de classe e as demais são consideradas atributos. Em seguida, dado um exemplo $\mathbf{x} = \{x_1, x_2, ..., x_n\}$, encontra-se a classe $c \in \Omega_C$ que maximiza

$$P(c_j, \mathbf{x}|S),\ j = \{1, 2, ..., m\}. \quad (4\text{-}3)$$

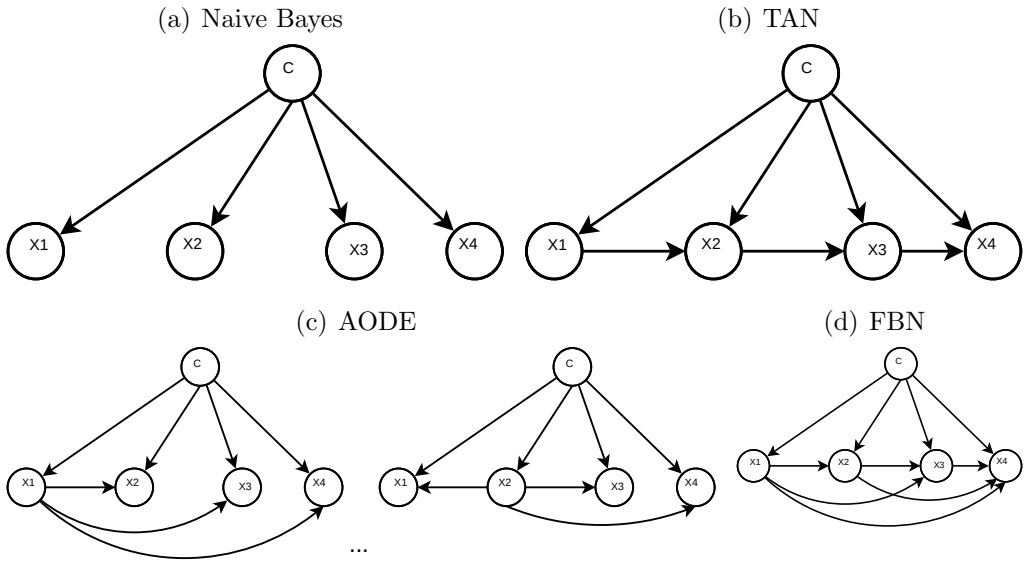
Construindo Redes Bayesianas

A construção de uma rede Bayesiana envolve três tarefas principais: *i*) determinar o modelo de rede a ser usado; *ii*) de acordo com o modelo escolhido, aprender a estrutura que melhor representa os dados e; *iii*) estimar os parâmetros da rede.

Existem diversos modelos ou classes de redes Bayesianas. Em particular, neste trabalho, quatro modelos são avaliados: *Naive Bayes* (NB), *Tree Augmented Naive Bayes* (TAN), *Aggregating One-Dependence Estimators* (AODE) e *Full Bayesian Network* (FBN). Como pode ser observado na Figura 4.1, esses modelos diferem entre si pela complexidade da estrutura da rede. Tipicamente, quanto maior a complexidade da estrutura da rede, maior é o poder de representação da mesma (Jordán, 2006). Todavia, o gargalo de qualquer algoritmo de aprendizado baseado em redes Bayesiana é a determinação da estrutura.

Dessa forma, estruturas complexas são mais dispendiosas em termos computacionais.

Figura 4.1: Redes Bayesianas com diferentes estruturas.



NB (Domingos e Pazzani, 1997) representa a classe de redes Bayesianas mais restritiva, pois esse modelo proíbe estritamente dependências entre os atributos, dado o valor da variável de classe. Ou seja, no modelo NB todos os atributos possuem apenas um nó pai, sendo este a variável de classe (Figura 4.1(a)). A suposição de independência entre os atributos torna a estrutura da rede muito simples e eficiente em termos computacionais. Todavia, como a suposição de independência não é real em muitos problemas, modelos NB podem ser tendenciosos (Langley et al., 1992).

No extremo oposto, FBN (Su e Zhang, 2006) representa a classe de redes Bayesianas menos restritiva pois são permitidas dependências entre todos os atributos. Ou seja, no modelo FBN um atributo pode depender de todos os demais. Na estrutura de um modelo FBN, a primeira variável é o nó pai de todos as outras; a segunda variável é o nó pai de todas as outras, exceto da primeira e assim sucessivamente, como mostra a Figura 4.1(d). Enquanto essa abordagem permite a construção de redes complexas, a maior quantidade de parâmetros a serem estimados aumenta a variância do modelo e o custo computacional para estimá-lo.

Entre os dois extremos estão classes como TAN (Friedman et al., 1997) e AODE (Webb et al., 2005). Essas classes de modelo buscam reter a simplicidade do modelo NB ao mesmo tempo que relaxam a suposição de independência entre os atributos. Um classificador TAN é uma rede Bayesiana que

contém a estrutura de um rede NB mas que permite que cada atributo tenha um outro atributo como pai, além da própria variável de classe (Figura 4.1(b)). Por outro lado, AODE é um agregado de pequenos modelos TAN cuja acurácia é obtida pela média das acurácias dos modelos menores. Em AODE, um modelo TAN é construído para cada atributo não classe, sendo esse atributo o pai de todos os demais no modelo (Figura 4.1(c)).

Uma vez definido o modelo da rede Bayesian, o problema de aprendizado procede determinando a estrutura da rede. Uma abordagem utilizada para este fim consiste em definir uma métrica de qualidade a qual é usada para aferir o quão bem uma estrutura representa o conjunto de dados observados. Um algoritmo de busca é então utilizado para encontrar, dentro do espaço de estruturas possíveis, aquela que otimiza a métrica de qualidade. Definida a estrutura, os parâmetros da rede podem ser estimados diretamente dos dados.

4.2.2 Árvore de Decisão

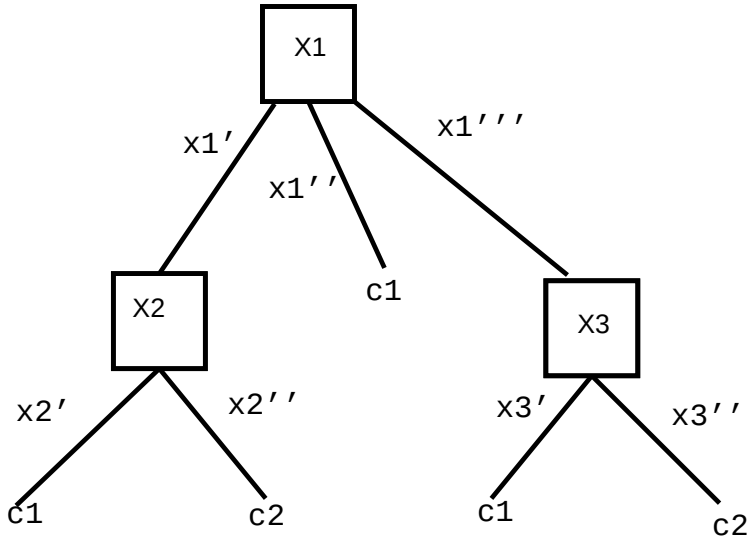
Uma abordagem "dividir e conquistar" para o problema de aprendizado supervisionado conduz a uma representação denominada *árvore de decisão* (Quinlan, 1993). Como ilustrado na Figura 4.2, em uma árvore de decisão, cada nó da árvore especifica um teste sobre algum atributo do modelo. Cada ramo partindo de um nó corresponde a um dos valores possíveis para o atributo. Uma instância é classificada partindo da raiz, testando o atributo especificado por esse nó e movendo para o ramo correspondente ao valor do atributo testado. Esse processo é repetido para as subárvores da raiz e para as subárvores desses nós até que uma folha seja encontrada. Cada folha provê um rótulo ou classe que classifica uma instância.

Construindo uma Árvore de Decisão

O processo de construção de uma árvore de decisão é conhecido como particionamento sucessivo. Inicialmente a árvore está vazia e o algoritmo começa a construí-la a partir da raiz, adicionando nós internos ou nós folhas à medida que se desce nos ramos da árvore. A questão central do algoritmo é a escolha do atributo que será testado em cada nó. Essa decisão é tomada considerando o atributo que melhor classifica os exemplos do *dataset* naquele instante. Para isso, usa-se todos os exemplos que estão sob aquele ramo particular.

Para determinar o atributo que melhor classifica um conjunto de exemplos em um ponto específico da árvore, os algoritmos de árvore de decisão

Figura 4.2: Estrutura de uma árvore de decisão.



utilizam uma propriedade estatística denominada *ganho de informação* (Mitchell, 1997). Essa métrica quantifica o quão eficaz é um atributo ao separar os exemplos de um *dataset* de acordo com os valores da variável de classe. Uma definição importante nesse processo é o conceito de *entropia* (Cover e Thomas, 2006), ou seja, o nível de pureza de um conjunto de exemplos. Dado um *dataset* D, a entropia em D é dada por:

$$Entropia(D) = \sum_{i=1}^{c} -p_i \log_2 p_i, \tag{4-4}$$

onde c é o número de classes ou rótulos e p_i é a proporção de exemplos com o rótulo i em D. O ganho de informação é a redução esperada na entropia quando um atributo A particular é usado para particionar os exemplos do *dataset*. Formalmente:

$$Ganho(D, A) = Entropia(D) - \sum_{v \in \Omega_A} \frac{|D_v|}{|D|} Entropia(D_v), \tag{4-5}$$

onde Ω_A é o conjunto de todos os valores possíveis para o atributo A e D_v é o subconjunto de D para o qual o valor do atributo A é v. O primeiro termo da Equação 4-5 é a entropia original do *dataset* D e o segundo termo é a entropia esperada para D após ser particionado por A.

Por fim, é importante ressaltar que muitos algoritmos usados para construir árvores de decisão executam uma poda após a indução da árvore. A poda é realizada removendo da árvore os nós folhas com alta taxa de erro. Esse procedimento assegura uma árvore menor e mais acurada, especialmente sobre *datasets* diferentes daquele no qual o modelo foi construído.

4.2.3 Support Vector Machine

O classificador SVM foi desenvolvido tendo como base o trabalho de Vapnik (Vapnik, 1995) sobre minimização do risco estrutural. Vapnik estabeleceu que o controle da capacidade de um classificador é necessário a fim de maximizar o potencial de generalização de uma função de classificação. A capacidade de um classificador mede o quanto este consegue capturar a complexidade dos dados. Em geral, quanto maior a capacidade, melhor o desempenho do classificador no conjunto de treinamento, embora o erro de generalização para um conjunto de teste seja maior. Vapnik estabeleceu que a teoria da minimização do risco estrutural permite especificar uma capacidade que minimiza o erro de generalização, provendo a base para o desenvolvimento do SVM.

SVM é um classificador que se baseia na teoria da maximização marginal a fim de minimizar o erro de generalização. Isto difere dos critérios usados pelos outros classificadores, os quais minimizam o erro médio quadrático. Para ilustrar, considere um *dataset* $D = \{\mathbf{x}_1, \mathbf{x}_2..., \mathbf{x}_l\}$ de l observações. Para simplificação, considere que cada instância do *dataset* D pertence a uma classe positiva ou negativa, tal que $\{\mathbf{x}_i, y_i\}, i = 1...l$, onde $y_i \in \{-1, 1\}$ e $\mathbf{x}_i \in \mathbf{R^d}$. A separação das instâncias pertencentes à classe positiva das negativas é feita por um hiperplano. Os pontos que residem no hiperplano são determinados por $\mathbf{w}.\mathbf{x} + b = 0$, onde $\mathbf{w}$ é um vetor normal, perpendicular ao hiperplano e $\frac{|b|}{||\mathbf{w}||}$ define a distância entre o hiperplano e a origem. Em cada lado do hiperplano, existe uma margem que define as fronteiras de cada classe, tal que:

$$y_i(\mathbf{w}.\mathbf{x} + b) \geq 1. \tag{4-6}$$

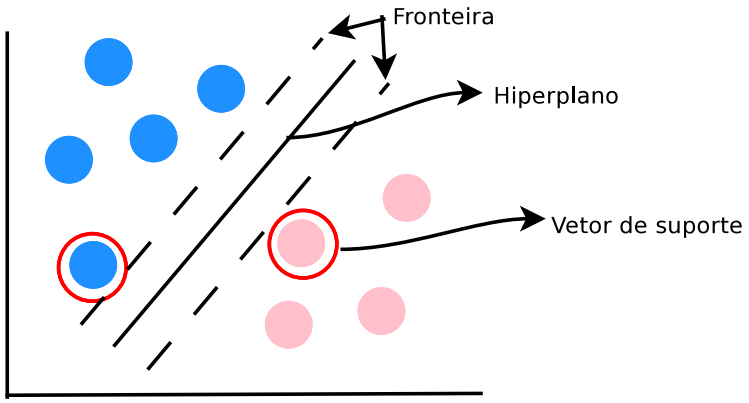
Durante a construção de um classificador SVM, o objetivo é maximizar esta margem através do posicionamento do hiperplano. Por geometria, é possível mostrar que a largura da margem é $\frac{|2|}{||\mathbf{w}||}$ e que a margem do hiperplano pode ser maximizada minimizando $||\mathbf{w}||^2$ sujeito à restrição 4-6. Os exemplos localizados sobre as fronteiras das classes, ou seja, os que satisfazem a igualdade em 4-6, são denominados *vetores de suporte* e constituem os exemplos do *dataset* de treinamento que apóiam ou definem o hiperplano. A Figura 4.3 ilustra um classificador SVM.

O algoritmo de construção de um classificador SVM é frequentemente representado em termos de multiplicadores de Lagrange, α_i. Nessas condições, a posição ótima do hiperplano pode ser obtida maximizando

$$\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i.\mathbf{x}_j, \tag{4-7}$$

sujeito às seguintes restrições:

Figura 4.3: Um classificador SVM.



$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \tag{4-8}$$

$$\sum_i \alpha_i y_i = 0. \tag{4-9}$$

Classificação baseada em Hiperplano

O hiperplano de um classificador SVM define as fronteiras das classes como observado no *dataset* de treinamento. Este hiperplano é usado diretamente para classificar novos exemplos. O lado do hiperplano onde reside um novo exemplo $\mathbf{x}$ pode ser calculado pela Equação 4-10

$$f(x) = sgn(\sum_i \alpha_i y_i \mathbf{x}.\mathbf{x}_i + b). \tag{4-10}$$

Embora o objetivo de um classificador SVM seja determinar a classe a que pertence uma instância, essa classificação nem sempre é direta. Ou seja, nem sempre os dados podem ser separados perfeitamente. Quando isso ocorre, um variável de custo C é introduzida para penalizar classificações incorretas no *dataset* de treinamento. Essa penalidade se traduz em uma restrição sobre os multiplicadores de Lagrange tal que

$$0 \leq \alpha_i \leq C. \tag{4-11}$$

Portanto, para treinar um classificador SVM em um *dataset* onde os dados não podem ser separados perfeitamente, basta aplicar a restrição 4-11 ao conjunto de restrições de 4-7.

Função Kernel

No núcleo de um classificador SVM está uma função *kernel* cujo propósito é converter o espaço de entrada (ou de características) em um espaço de dimensão maior. Por exemplo, embora não especificado explicitamente, na Equação 4-10 existe a função *kernel* $K(\mathbf{x}, \mathbf{x}_i)$ tal que

$$f(x) = sgn(\sum_i \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) + b), \tag{4-12}$$

onde $\mathbf{x}$ representa o vetor de entrada, $\mathbf{x}_i$ os exemplos do treinamento e y_i a classe associada ao exemplo. Nesse exemplo, a função *kernel* é dada por $K(\mathbf{x}, \mathbf{x}_i) = \mathbf{x}.\mathbf{x}_i$. Essa função é conhecida como função *kernel* linear, uma vez que ela simplesmente encontra o produto dos vetores de entrada em um espaço linear.

Além da separação linear, a implementação do SVM pode ser estendida para contemplar a classificação de dados não separáveis linearmente. O objetivo de um SVM, neste caso, é permitir que classificações não lineares possam ser aplicadas a um *dataset*, mapeando os vetores de entrada para um espaço de dimensão maior. Nesse novo espaço, uma classificação linear pode ser alcançada. A função *kernel* para esta tarefa pode ser definida genericamente como:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\phi}(\boldsymbol{x_i}).\boldsymbol{\phi}(\boldsymbol{x_j}) \tag{4-13}$$

onde $\boldsymbol{\phi}$ é a função usada para converter um vetor de entrada $\mathbf{x}$ para um espaço desejável. Em geral, a função de mapeamento usada depende da aplicação, mas as seguintes funções são comumente empregadas:

Tabela 4.1: Funções *kernel* não lineares comumente usadas.

Descrição	$K(\mathbf{x}_i, \mathbf{x}_j)$
Radial Basis Function (RBF)	$\exp[-\gamma\|\|\mathbf{x}_i - \mathbf{x}_j\|\|^2]$
Polinomial de grau d	$(\mathbf{x}_i + \mathbf{x}_j)^d$

4.3 Discretização

Alguns algoritmos de classificação não são projetados para processar dados em escala numérica, sendo capazes apenas de tratar dados nominais. Para usar esses algoritmos de forma genérica, atributos numéricos devem primeiramente ser discretizados em um número pequeno de intervalos distintos.

Discretização é um tipo de pré-processamento muito comum em aprendizado de máquina e pode prover, além de generalização, melhora no desempenho dos algoritmos. Por exemplo, embora muitos algoritmos de indução de árvores de decisão possam processar valores numéricos, algumas implementações tornam-se mais eficientes quando dados nominais estão presentes.

Existem diversos métodos de discretização, sendo um dos mais populares o que divide o intervalo de valores de um atributo em compartimentos de largura fixa. Como exemplo de como esse método converte valores numéricos em valores nominais, considere um atributo A cujos valores são divididos em 20 níveis. O rótulo $c = 1$ cobre os valores entre 0% e 5% do valor máximo do atributo enquanto o rótulo $c = 20$ cobre os valores entre 95% e 100% desse valor.

4.4 Métricas de Acurácia

Um fator crucial em aprendizado supervisionado é a acurácia do classificador aprendido. Por esse motivo, a métrica natural para medir o desempenho de um classificador é o seu *poder de predição*, ou seja, a acurácia do classificador ao prever um rótulo. Para estimar o desempenho de um classificador ao prever o rótulo de um dado não observado anteriormente, utilizamos um *dataset* independente daquele usado no treinamento. Chamamos esse *dataset* de teste.

Dado um *dataset* de teste, a classificação para cada exemplo desse *dataset* pode ser *correta*, caso o classificador estime o rótulo real do exemplo, ou *incorreta*, se a estimação não concorda com o valor real do rótulo. Chama-se *acurácia* a proporção de classificações corretas.

Considere um problema de classificação binária, ou seja, um problema onde os rótulos dos exemplos podem assumir apenas dois valores: 0 ou 1. Estimar o valor do rótulo de um exemplo, nessas condições, pode levar a quatro resultados diferentes, como mostra a Tabela 4.2. Tomando o valor 0 como positivo e 1 como negativo, *positivos verdadeiros* (TP) e *negativos verdadeiros* (TN) são classificações corretas. Um *falso positivo* (FP) ocorre quando o classificador estima o valor 0 para um rótulo, quando o valor real da classe é 1. De maneira análoga, um *falso negativo* (FN) ocorre quando o classificador estima o valor 1 quando o valor real do rótulo é 0. Nesses termos, a acurácia do classificador pode ser expressa pela Equação 4-14.

$$Acuracia = \frac{TP + TN}{TP + TN + FP + FN}. \tag{4-14}$$

A métrica de acurácia tal como definida na Equação 4-14 pode dar uma

Tabela 4.2: Resultados possíveis para uma classificação binária.

		Rótulos Estimados	
		0	1
Rótulos Reais	0	TP	FN
	1	FP	TN

impressão incorreta do desempenho do classificador quando um dos dois valores da variável de classe é raro. Como exemplo, considere um *dataset* onde apenas 10% dos exemplos possuem o rótulo igual a 0. Um classificador trivial que estime sempre o valor 1 obtém uma acurácia de 90% nessas condições. Uma métrica mais apropriada para esse caso é tomar a média das precisões com que o classificador estima cada rótulo. Chamamos essa métrica de *acurácia balanceada* (BA). Formalmente:

$$BA = \frac{precisao(0) + precisao(1)}{2}, \tag{4-15}$$

onde a precisão com que um rótulo r é estimado, considerando r o rótulo positivo, é dado por:

$$precisao(r) = \frac{TP}{TP + FP}. \tag{4-16}$$

Além das métricas de acurácia definidas nas Equações 4-14 e 4-15, a *taxa de alarme falso* (FA) e a *taxa de detecção* (DET) para um rótulo r também podem oferecer informações importantes sobre o comportamento do classificador. As Equações 4-17 e 4-18 definem formalmente essas métricas para um rótulo r considerado como o rótulo positivo.

$$FA(r) = \frac{FP}{TN + FP} \tag{4-17}$$

$$DET(r) = \frac{TP}{TP + FN} \tag{4-18}$$

Ainda com relação à avaliação de desempenho, uma questão importante é como dividir o *dataset* disponível em um conjunto de treinamento e outro de teste. Uma técnica denominada *k-fold cross validation* é comumente usada com esse propósito. Nessa técnica, o *dataset* é dividido em um número k de partições, cada uma com aproximadamente o mesmo tamanho. Uma das partições é usada para teste, enquanto as demais $k - 1$ partições são usadas para treinamento. Em seguida, o processo é repetido k vezes, de forma que, no final, cada exemplo seja usado exatamente uma vez para teste. Cada processo de treinamento/teste produz um valor para a métrica de acurácia. A acurácia

final do classificador é obtida pela média das k acurácias intermediárias. Um padrão que tem sido adotado para k é o valor 10 (Witten e Frank, 2005).

4.5 Estimação de Problemas de Desempenho

Neste capítulo, é avaliado um mecanismo para prever problemas de desempenho. Dado um conjunto de métricas e os valores observados para essas métricas em um tempo t qualquer, deseja-se prever se será possível ou não manter um SLO, definido pelo usuário, em um futuro próximo. O problema é modelado usando classificação.

Na modelagem adotada neste trabalho, um classificador é uma função $f : M \rightarrow S$ que associa a um vetor $\mathbf{m} \in \Omega_M$ um rótulo $s \in \Omega_S$. O vetor $\mathbf{m}$ é um conjunto de valores para n métricas observadas no instante t. O rótulo s é um dos dois estados possíveis para o conjunto $S = \{1, 0\}$, onde 1 denota um estado de conformidade com um SLO específico e 0 representa um estado de violação. Um *log* de métricas observadas durante a operação do sistema, ou seja, o *dataset* $D = \{(\mathbf{m}^{(1)}, s^{(1)}), (\mathbf{m}^{(2)}, s^{(2)}), ..., (\mathbf{m}^{(N)}, s^{(N)})\}$, é o conjunto de dados de treinamento fornecido ao procedimento de aprendizado. Note que em cada exemplo $(\mathbf{m}^{(i)}, s^{(i)})$ do treinamento, existe um intervalo fixo l, $l \geq 0$, entre os tempos em que $\mathbf{m}$ e s são coletados. Esse intervalo descreve o horizonte de tempo que se deseja antecipar.

O problema descrito acima é supervisionado porque um procedimento externo identifica os valores de s que correspondem a cada vetor $\mathbf{m}$. Assim, dado um SLO definido pelo usuário, um procedimento supervisor que atribui rótulos aos exemplos do *dataset* de treinamento e o próprio *dataset*, um modelo de relacionamento entre M e S é induzido. Esse modelo é usado, então, para decidir se um conjunto particular de métricas irá se correlacionar com um estado de conformidade ou violação de SLO. Além da própria classificação, o modelo provê também a probabilidade de um vetor de métricas pertencer a cada estado. Ou seja, dada uma instância $\mathbf{m}$, o modelo estima a probabilidade de $\mathbf{m}$ estar em conformidade ($P(s^+|\mathbf{m})$, $s^+ = 1$) ou violação ($P(s^-|\mathbf{m})$, $s^- = 0$) em relação ao SLO.

Na seção seguinte, é apresentado um estudo experimental para investigar, dentre as classes de modelos de aprendizado supervisionado descritas neste trabalho, aquelas mais apropriadas para modelar o relacionamento entre M e S. Os experimentos realizados neste trabalho utilizam o cenário de referência descrito no Capítulo 3. Com relação aos experimentos, as seguintes observações são ressaltadas:

– O *framework* MapReduce executa a aplicação *WordCount*.

– As métricas apresentadas na Tabela 3.1 constituem as métricas do vetor **m**. Essas métricas são coletadas a cada 15 segundos. Embora esse intervalo possa ser mudado, para os propósitos deste trabalho, ele é mantido constante. Essa decisão foi motivada pelo fato desse valor ter sido suficiente para capturar estados de conformidade e de violação durante a execução da aplicação MapReduce.

– Nos experimentos realizados neste trabalho, assumimos que o intervalo l entre as medições de **m** e s é igual a zero.

4.6 Experimentos

A fim de avaliar o quão aplicável são os algoritmos descritos na seção 4.2 (DT, NB, TAN, AODE, FBN e SVM) para derivar modelos que assistam o gerenciamento autonômico do desempenho de sistemas baseados em middleware, um estudo comparativo é proposto. Nesse estudo, os seis algoritmos são avaliados em relação a diversos critérios importantes para a construção de modelos de desempenho, como os descritos abaixo.

– Acurácia: esse critério avalia o quão acurados são os diferentes algoritmos ao estimar problemas de desempenho.

– Tempo requerido para treinamento: esse critério estabelece o tempo que leva cada algoritmo para induzir um modelo de desempenho.

– Tempo requerido para classificação: esse critério estabelece o tempo que leva cada algoritmo para classificar uma única instância ou exemplo.

– Sensibilidade ao número de exemplos em estado de violação: esse critério avalia o comportamento dos algoritmo quando esses são aplicados em *datasets* onde o número de instâncias em estado de violação é pequeno. Esse critério se torna importante para o problema proposto, uma vez que eventos de degradação de desempenho tendem a ser mais raros que eventos de normalidade.

– Sensibilidade ao tamanho do *dataset* de treinamento: esse critério avalia o comportamento dos algoritmos em relação ao tamanho do conjunto de dados disponível para treinamento. Esse critério é particularmente importante para mecanismos *online* de captura de comportamento de sistemas, uma vez que esses sistemas podem executar em ambientes que requerem a reconstrução frequente dos modelos, em virtude de mudanças no contexto de execução.

As implementações dos seis algoritmos foram providas pela ferramenta Weka (Witten e Frank, 2005) e as configurações utilizadas são descritas na Tabela 4.3.

Tabela 4.3: Configuração dos parâmetros usados nos algoritmos de aprendizado supervisionado.

NB	foi utilizada a classe *weka.classifiers.bayes.NaiveBayes*. Como não há seleção de modelos nesse tipo de rede Bayesiana, não é necessário configurar nenhum parâmetro para esse algoritmo.
TAN	foi utilizada a classe *weka.classifiers.bayes.BayesNet* e o algoritmo de busca *weka.classifiers.bayes.net.search.local.TAN*. A métrica de qualidade otimizada foi a entropia (Witten e Frank, 2005). Outras métricas de qualidade, como AIC e MDL também foram avaliadas. Todavia essas métricas não resultaram em modelos significativamente melhores.
AODE	foi utilizada a classe *weka.classifiers.bayes.AODE*. Como não há seleção de modelos nesse tipo de rede Bayesiana, não é necessário configurar nenhum parâmetro para esse algoritmo.
FBN	foi utilizado o algoritmo proposto por Su e Zhang em (Su e Zhang, 2006). Nesse algoritmo, as probabilidades vinculadas aos nó da rede não são capturadas pela Equação 4-2, mas sim através de uma árvore de decisão. O uso de uma árvore de decisão para capturar a CPT e a ausência de seleção de modelos em virtude da estrutura da rede (que assegura a dependência entre as variáveis), torna o algoritmo em questão muito rápido, especialmente quando comparado a outros algoritmos que constroem redes genéricas.
DT	foi utilizada a classe *weka.classifiers.trees.J48* que implementa o algoritmo C4.5 (Quinlan, 1993). A seguinte configuração foi usada por esta implementação: fator de confiança ou limiar de poda de 0.3 e número mínimo de exemplos por folha igual a 2. Esses valores foram estimados por um procedimento automático de seleção de parâmetros provido pelo Weka.
SVM	foi utilizada a biblioteca WLSVM (EL-Manzalawy e Honavar, 2005), uma implementação da biblioteca LibSVM para a ferramenta Weka. A biblioteca LibSVM é computacionalmente mais eficiente que as implementações SVM disponíveis no Weka, além de fornecer mais implementações para esse tipo de classificador. Em particular, neste trabalho, foi avaliada apenas a função *kernel* RBF. Tipicamente, essa função é capaz de tratar os casos em que a relação entre os atributos e a classe é não linear, além de requerer menos parâmetros que as demais funções. A seguinte configuração foi usada para essa implementação: $C = 32$ e $\gamma = 0.5$ Os valores desses parâmetros foram estimados por um procedimento automático provido pela biblioteca WLSVM.

4.6.1 Log da Aplicação MapReduce

Para treinar os algoritmos e induzir o modelo de comportamento de uma aplicação baseada em *middleware*, foi utilizado um *dataset* com 79319 instâncias ou exemplos. Esse *dataset* foi gerado a partir de um *log* de execuções da aplicação *WordCount* e o comportamento, isto é, o desempenho, dessa aplicação em cada execução, em diferentes cenários de carga.

Um cenário de carga exercita diferentes tipos de recursos computacionais e diferentes intensidades de utilização desses recursos. Os cenários de carga utilizados neste trabalho foram criados a partir de injetores, os quais podem gerar três padrões de cargas: CPU, disco e rede. A intensidade de cada padrão de carga varia entre 20%, 50% e 80% de utilização.

No total, foram utilizados 36 cenários de carga: 27 cenários correspondendo a todas as combinações possíveis entre três recursos (CPU, disco e rede), podendo cada recurso assumir três valores (20%, 50% e 80%) de utilização; 3 cenários abrangendo apenas carga de CPU (20%, 50% e 80% de utilização); 3 cenários abrangendo apenas carga de disco (20%, 50% e 80% de utilização) e 3 cenários abrangendo apenas carga de rede (20%, 50% e 80% de utilização). O arquivo de *log* consiste no conjunto de métricas (Tabela 3.1) coletadas durante a execução da aplicação em cada cenário. Além disso, cada cenário foi executado 30 vezes, resultando em 1080 execuções. Durante cada execução, a aplicação *WordCount* processa um arquivo de 1GB usando 3 *Workers*, cada qual instanciado em um nó de um *cluster*. Um desses nós recebe a injeção de carga. O *Master* e o serviço de evento são instanciados em outros dois nós. Cada nó possui dois processadores Intel Pentium IV de 3 GHz e 3 GB de memória. Os nós estão conectados por uma rede Ethernet de 1 Gigabit.

Após a geração do *log*, os dados passaram por um pré-processamento. O intuito da função de pré-processamento é preparar os dados para que os mesmos possam ser usados como entrada para os algoritmos de classificação. A função de pré-processamento utilizada neste trabalho segue a metodologia proposta por por Cohen et al. (Cohen et al., 2004) e consiste nos seguintes passos. Primeiramente, elimina-se possíveis ruídos, ou *outliers*, do *dataset*. Para isto, foi utilizado o conceito de *percentil*. Um percentil é o valor de uma variável abaixo do qual se encontra uma certa porcentagem de observações. Por exemplo, o 20ºpercentil é o valor abaixo do qual se encontram 20% das observações de uma variável. Para a remoção de ruídos, o 5ºe o 95ºpercentil do atributo *avg_response_time* foram calculados e todos os exemplos cujo valor da variável não estava entre os dois percentis foram eliminados do *dataset*.

O segundo passo da preparação dos dados consiste na estimação da

variável de classe, isto é, a definição do procedimento supervisor que atribui rótulos para os exemplos do *dataset*. Para isto, foi definido um valor limiar ou SLO para a variável *avg_response_time*. Novamente o conceito de percentil foi utilizado para computar o SLO. Como ilustração, considere que o SLO é definido como o 80ºpercentil de *avg_response_time*. Isto significa que o valor limiar para *avg_response_time* é aquele abaixo do qual se encontram 80% das observações. Nos exemplos em que o valor da variável *avg_response_time* excede o SLO, atribui-se o valor 0 à variável de classe, indicando violação. Nos demais exemplos, atribui-se o valor 1 à variável de classe para indicar conformidade. Esse procedimento foi aplicado no *dataset* deste trabalho. Todavia, o SLO variou do 50ºao 90ºpercentil da variável *avg_response_time*. Vale ressaltar que a escolha pela técnica de percentil para computar o SLO retira do usuário a responsabilidade de definir um valor absoluto como limiar para a variável *avg_response_time*. O valor absoluto será computado em tempo de execução, tendo como base os valores observados e o percentil definido pelo usuário.

O terceiro e quarto passos da preparação dos dados são representados, respectivamente, pela normalização e discretização do *dataset*. Na normalização, para cada atributo que compõe um exemplo, divide-se o valor do atributo naquele exemplo pelo maior valor encontrado para o atributo em todo o *dataset*, resultando em um número entre 0 e 1. Em seguida, para cada atributo que compõe um exemplo, o valor normalizado é discretizado em um valor nominal. Este valor, por sua vez, é determinado por um método de discretização de largura fixa aplicado ao domínio do atributo, como discutido na Seção 4.3.

O resultado final da fase de pré-processamento consistiu em um *dataset* composto por 79319 instâncias. No restante deste trabalho, iremos nos referir a esse conjunto de dados como o *dataset MapReduce*. Esse *dataset* foi usado para treinar os classificadores nos experimentos a seguir.

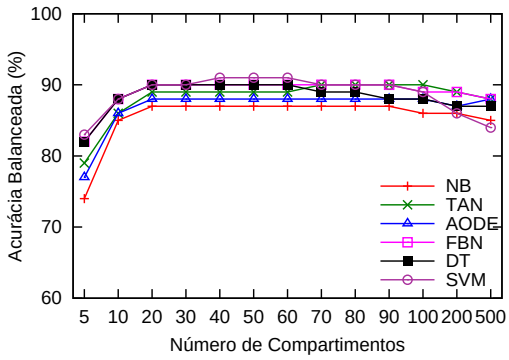
4.6.2 Avaliação do Nível de Discretização

Nesta seção, são compartilhados os resultados obtidos no estudo comparativo que avaliou o desempenho dos seis algoritmos de aprendizado de máquina no contexto de estimação de problemas de desempenho. Primeiramente, analisa-se como o número de intervalos usados no método de discretização afeta o desempenho dos algoritmos. Para esse experimento e no restante deste trabalho, a acurácia balanceada (BA) é usada como a medida do poder de estimação dos algoritmos. Nos gráficos da Figura 4.4 são mostrados, para cada algoritmo, a acurácia e o tempo computacional requerido para induzir um

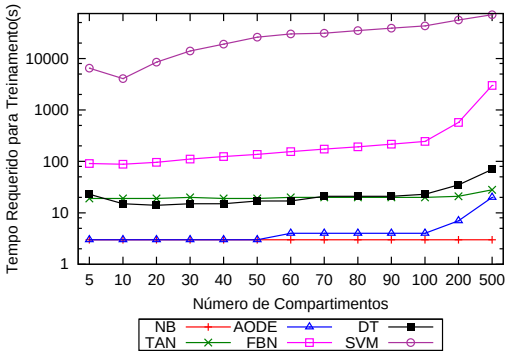
modelo de desempenho usando *10-fold cross validation*, quando o número de intervalos de discretização varia de 5 a 500 níveis. Na Figura 4.4(a), é possível notar que há uma melhora na acurácia dos algoritmos quando o número de intervalos aumenta de 5 para 20. A partir desse valor, no entanto, o aumento do número de intervalos não reflete em um aumento na acurácia. A Figura 4.4(b) mostra que há uma pequena elevação no tempo requerido para treinar o modelo quando o número de intervalos cresce. Essa elevação é mais nítida para o SVM e mais branda para o classificador NB. Com base nesse experimento, o valor de 40 intervalos foi escolhido como o nível de discretização usado nos demais experimentos deste trabalho.

Figura 4.4: Avaliação do comportamento dos algoritmos de aprendizado supervisionado em relação ao número de intervalos na discretização.

(a) Acurácia



(b) Tempo de Treinamento



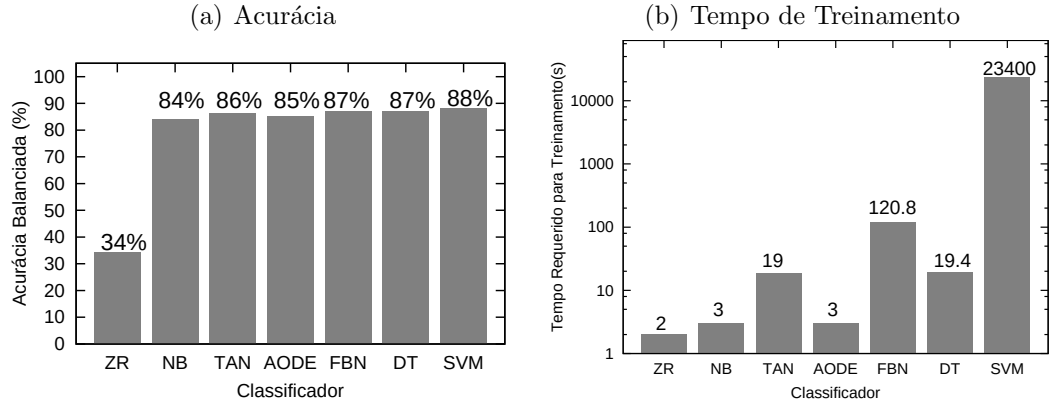
4.6.3 Avaliação do Desempenho dos Algoritmos

Os algoritmos também foram avaliados em relação à acurácia ao estimar um problema de desempenho. Nessa avaliação, foi incluído também o classificador ZeroR (ZR) ou moda. Esse classificador simplesmente reporta o rótulo maioritário e serve como uma referência de baixo poder preditivo para os outros algoritmos. Nos gráficos da Figura 4.5 são mostrados, para cada algoritmo, a acurácia e o tempo requerido para treinamento. Em cada gráfico, as barras correspondem ao desempenho, isto é, a acurácia na Figura 4.5(a) e o tempo computacional na Figura 4.5(b), de cada algoritmo, computado como a média dos desempenhos obtidos sobre o *dataset MapReduce* para diferentes definições de SLO, ou seja, 50º, 60º, 70º, 80ºe 90ºpercentis. Os classificadores foram treinados usando *10-fold cross validation* e o nível de discretização foi definido em 40 intervalos. O gráfico da Figura 4.5(b) está em escala logarítmica.

A Tabela 4.4 apresenta os resultados desse experimento de forma mais nítida e completa. As colunas dessa tabela refletem a taxa de detecção e a taxa de alarme falso para o estado violação, a acurácia balanceada e o tempo requerido para treinamento. É possível notar que, exceto por ZR, todos os classificadores atingiram um nível alto de acurácia, a qual variou entre 84% e 88%. ZR, ao contrário, apresentou um desempenho muito inferior, com acurácia de aproximadamente 34%. Essa diferença mostra que a maior parte das estimações realizadas não são relações triviais. Portanto, um classificador muito simples como o ZR não foi capaz de capturar o relacionamento subjacente corretamente.

Em todos os *datasets* avaliados, SVM apresentou a melhor acurácia. No entanto, essa precisão é acompanhada por um custo computacional alto. O tempo requerido para treinar o classificador SVM é no mínimo duas ordens de grandeza maior que a maioria dos outros algoritmos. DT e FBN também apresentaram uma acurácia elevada, mas o custo computacional de FBN foi alto, comparado com os demais algoritmos. Em geral, as redes Bayesianas mais simples, como NB, TAN e AODE, apresentaram um bom compromisso entre acurácia e custo computacional. No entanto, esses métodos geraram mais alarmes falsos.

Figura 4.5: Comparação dos algoritmos de aprendizado supervisionado em relação ao poder de estimação e tempo requerido para treinamento.



Além do esforço computacional para treinar um modelo de previsão, o tempo requerido para classificar uma instância do vetor de métricas também foi avaliado. Em todos os algoritmos, esse tempo foi negligenciável, sendo menor que 1 milissegundo. O classificador SVM, no entanto, levou, em média, 5 milissegundos para classificar uma única instância.

Sendo o SVM um algoritmo de difícil parametrização, outros testes foram realizados com esse classificador com o intuito de confirmar os resultados ilustrados na Figura 4.5, especialmente o tempo de treinamento. A primeira abordagem nesse sentido foi avaliar o desempenho do algoritmo no *dataset MapReduce* sem, no entanto, discretizar os exemplos desse *dataset*. Essa abordagem, todavia, não resultou em uma melhora significativa do tempo de treinamento, o qual continuou algumas ordens de grandeza maior que o tempo requerido pelos outros classificadores. Outra abordagem avaliada foi utilizar outra implementação, isto é, outra biblioteca, para o algoritmo SVM. A biblioteca LibLinear (Fan et al., 2008), conhecida pelo bom desempenho em grandes *datasets*, foi utilizada com esse intuito. Para esse teste, o tempo de treinamento do algoritmo SVM foi avaliado tanto no *dataset MapReduce* com discretização quanto no *dataset* sem discretização. Para treinar um classificador SVM no *dataset* com discretização, a biblioteca LibLinear levou aproximadamente 720 segundos, atingindo uma acurácia de 77%. O tempo requerido para treinamento, neste último caso, foi significativamente menor que o tempo requerido pela biblioteca LibSVM. Por outro lado, para treinar o mesmo classificador no *dataset* sem discretização, o tempo requerido para treinamento também foi próximo de 700 segundos. A acurácia obtida pelo classificador, no entanto, não ultrapassou 50%. Esse fato ilustra que, embora existam implementações rápidas para o classificador SVM, o processo de parametrização desses algoritmos não é trivial. Como nenhuma das duas abordagens resultou em um classificador SVM cujo compromisso entre custo e acurácia se aproximasse dos demais classificadores, optamos por manter, para os próximos experimentos, a implementação LibSVM e o processo de discretização do *dataset*. Essa última decisão, isto é a opção pela discretização do *dataset*, foi tomada para manter padronização com a abordagem usada nos demais classificadores.

Tabela 4.4: Comparação detalhada do desempenho dos algoritmos de aprendizado supervisionado.

	Taxa de Detecção	**Alarme Falso**	**Acurácia**	**Tempo Treinamento**
ZR	-	-	34.0%	2s
NB	88.2%	13.0%	84.0%	3s
TAN	82.4%	9.0%	86.8%	19s
AODE	87.4%	11.6%	85.4%	3s
FBN	83.2%	8.6%	87.4%	120.8s
DT	82.6%	8.8%	87.4%	19.4s
SVM	82.2%	8.0%	87.8%	23400s

4.6.4 Avaliação do Impacto da Definição do SLO

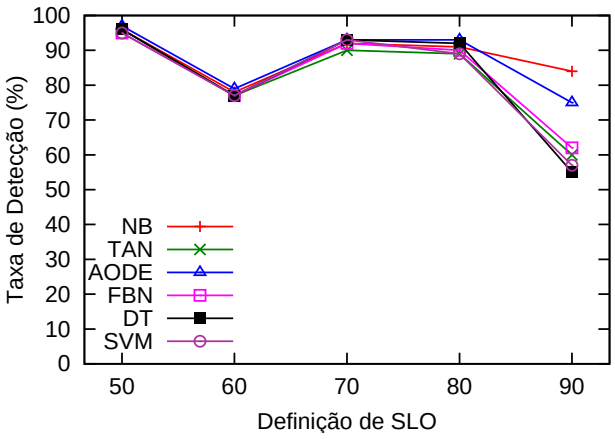
A acurácia média atingida por cada classificador no *dataset MapReduce* e em diferentes definições de SLO é alta, como mostra a Tabela 4.4. Todavia, na maioria dos algoritmos investigados, a taxa média de detecção para o estado de violação apresenta um desempenho inferior à acurácia média. Esse desempenho inferior pode ser melhor explicado analisando-se a Tabela 4.5. Nessa tabela, é listada não a média do desempenho dos algoritmos, mas o desempenho obtido separadamente em cada uma das definições possíveis para o SLO. Para simplificação, a Tabela 4.5 mostra apenas o resultado para o classificador TAN. É possível notar que, nas diferentes definições de SLO, a taxa de detecção varia mais intensamente que a acurácia balanceada. Esse fato permite concluir que a taxa de detecção é mais sensível à definição do SLO que a métrica de acurácia.

Tabela 4.5: Acurácia e tempo de processamento do classificador TAN em cada *dataset* gerado a partir de uma definição de SLO.

SLO	Taxa de Detecção	Alarme Falso	Acurácia	Tempo Treinamento
50º	96%	28%	86%	19s
60º	77%	6%	87%	19s
70º	90%	4%	93%	19s
80º	89%	4%	89%	19s
90º	60%	3%	79%	19s

Por outro lado, o número de instâncias em estado de violação diminui à medida que o valor do SLO aumenta de 50ºpara 90ºpercentil. Isso ocorre porque o SLO representa um valor limiar para o tempo médio de resposta dos serviços. À medida que o valor do percentil aumenta, o valor limiar também aumenta e o número de instâncias em estado de violação diminui. Dessa forma, para analisar a sensibilidade dos algoritmos em relação ao número de instâncias em violação, a taxa de detecção foi plotada em função do SLO. O gráfico da Figura 4.6 ilustra essa relação. Como esperado, quando o valor do percentil associado ao SLO aumenta, a taxa de detecção diminui. A única exceção ocorre no 60ºpercentil. Entretanto, os testes realizados não permitiram explicar esse fenômeno. Esse comportamento é observado para todos os algoritmos. Para avaliar qual algoritmo apresenta a maior sensibilidade à definição do SLO, cada curva da Figura 4.6 foi aproximada para uma reta e a taxa de variação da reta foi calculada. A tabela da Figura 4.6 mostra esses valores. É possível notar que os classificadores NB e AODE apresentam as menores sensibilidades.

Figura 4.6: Comparação dos algoritmos de aprendizado em relação à sensibilidade ao SLO.



	Taxa de Variação
NB	0.3
TAN	0.9
AODE	0.55
FBN	0.82
DT	1.02
SVM	0.95

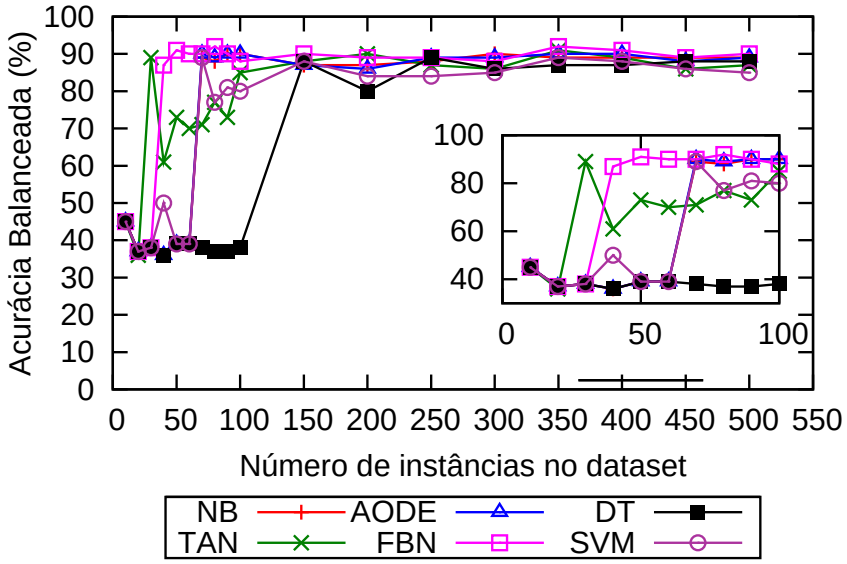
4.6.5 Avaliação do Impacto do Tamanho do Dataset

A sensibilidade dos classificadores em relação ao tamanho do *dataset* de treinamento também foi analisada. Para esse experimento, pequenos *datasets*, cujos tamanhos variaram de 10 a 500 exemplos, foram criados. Cada novo *dataset* foi gerado tomando instâncias aleatórias do *dataset MapReduce*. Em seguida, a acurácia balanceada de cada classificador, em todos os *datasets*, foi computada. A Figura 4.7 mostra o resultado desse experimento. É possível notar que a acurácia dos classificadores aumenta à medida que o tamanho do *dataset* aumenta. Todavia, a acurácia dos classificadores satura gradualmente. Em geral, a partir de 150 instâncias a acurácia se torna estável para todos os algoritmos. Alguns classificadores, no entanto, obtêm uma acurácia elevada em *datasets* menores, notadamente o classificador FBN. Por outro lado, é possível notar claramente que o classificador DT possui o pior desempenho, revelando a grande sensibilidade desse algoritmo em relação ao tamanho do *dataset*.

4.6.6 Avaliação do Desempenho do SMART

No Capítulo 3, foi discutida a importância da política de escalonamento para aplicações MapReduce. Foi estabelecido que a abordagem de escalonamento adotada para o *framework* MapReduce proposto neste trabalho é ordenar o conjunto de *Workers* pela probabilidade de um nó violar um SLO. Dessa forma, nós com maior probabilidade de violação são colocados no final da fila de *Worker*, uma vez que assume-se que tais nós têm maior probabilidade de degradação. Para verificar a eficácia dessa política de escalonamento, a máquina de inferência do SMART, a qual infere a probabilidade de violação do SLO em um nó, foi implementada usando o classificador TAN. Como verificado nos

Figura 4.7: Comparação dos algoritmos de aprendizado supervisionado em relação ao número de instâncias no *dataset* de treinamento.



experimentos descritos neste capítulo, esse classificador atingiu uma acurácia acima de 80%, além de ser bastante eficiente.

Para este experimento, foram utilizados 10 nós assim distribuídos: 1 nó para o componente *Master* (nó d_4), 1 nó para o serviço de evento do SCS (nó d_1) e 8 nós para os componentes *Workers* (nós d_2 a d_{10}, exceto o nó d_4). A aplicação é configurada com 20 componentes *Workers*, resultando em, aproximadamente, 2 *Workers* instanciados por nó. A máquina de inferência do SMART é treinada com o *dataset MapReduce* e o SLO é definido como o 80ºpercentil de *avg_response_time*. Durante a execução da aplicação, o sistema é submetido a um cenário de carga que envolve CPU, disco e rede, como mostra a Tabela 4.6.

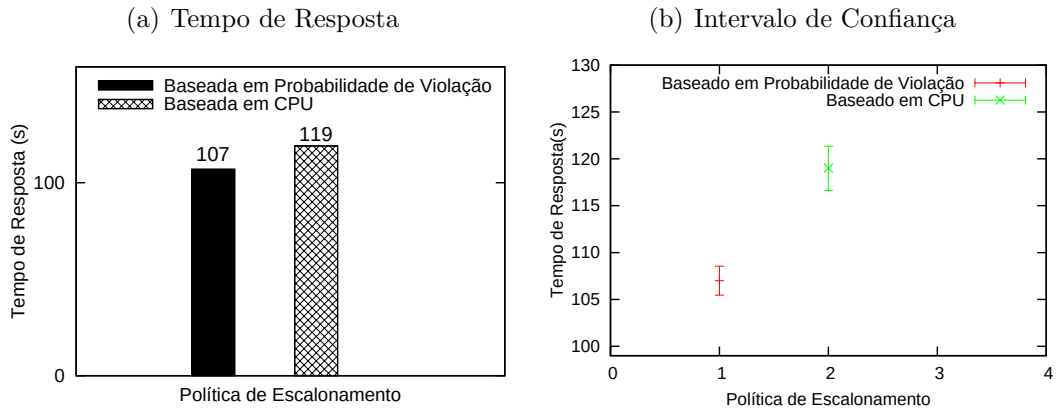
Tabela 4.6: Cenário de carga para o experimento que testa a eficácia do SMART.

Nó	**Padrão de Carga**
d_2, d_3,d_5 e d_{10}	-
d_6	50% de CPU
d_7	50% de CPU e 50% de disco
d_8	20% de rede e 80% de disco
d_9	50% de rede

Nas condições descritas acima, o desempenho da aplicação MapReduce foi comparado em duas situações: usando a política de escalonamento baseada

em CPU e a política baseada em probabilidade de violação. O desempenho da aplicação é medido considerando o tempo total de execução. Para fins de comparação, considerou-se a política que proporciona o melhor desempenho, aquela que resulta em um menor tempo de execução da aplicação. A Figura 4.8(a) mostra a média do tempo total de execução da aplicação MapReduce, em 10 execuções, para cada política de escalonamento, usando um nível de confiança de 95% no intervalo de confiança. A Figura 4.8(b) ilustra o valor médio do tempo de execução e o intervalo de confiança para cada política. Enquanto a política baseada em probabilidade de violação proporciona um tempo médio de execução de 107 segundos, a política baseada em CPU proporciona um tempo médio de execução de 119 segundos. Ou seja, a política baseada em probabilidade de violação consegue melhorar em 10% o desempenho da aplicação. Esse resultado demonstra que a técnica de aprendizado estatístico é capaz de capturar o comportamento da aplicação MapReduce com maior precisão que uma regra simples baseada apenas na utilização de CPU.

Figura 4.8: Desempenho da aplicação MapReduce usando uma política de escalonamento baseada em CPU e na probabilidade de violação.



4.7 Considerações Finais

Embora todos os seis algoritmos apresentaram um poder de estimação elevado para prever problemas de desempenho, o classificador SVM foi consistentemente melhor em relação a este critério. Todavia, o tempo requerido para treinar o classificador SVM foi absolutamente maior. Mesmo usando uma implementação do classificador própria para *datasets* grandes, o menor tempo de treinamento requerido foi aproximadamente sete vezes maior que os demais classificadores. Devido a esse fato, o classificador SVM foi considerado não

apropriado para o problema de estimação proposto neste trabalho. Os classificadores FBN e DT também se mostraram muito acurados. De fato, considerando o critério poder de estimação, os dois algoritmos se comportam de forma muito semelhante. Entretanto, enquanto FBN manteve-se acurado mesmo em *datasets* menores, o classificador DT apresentou uma grande sensibilidade ao tamanho do conjunto de dados observados. Por outro lado, o classificador FBN apresentou o segundo pior custo computacional. Essas desvantagens tornaram tais algoritmos menos aplicáveis para o problema proposto. Finalmente, os classificadores NB, AODE e TAN, apesar de atingirem níveis de acurácia menores que os demais e as maiores taxas de alarmes falsos, apresentaram o melhor compromisso entre acurácia e custo computacional. Adicionalmente, esses algoritmos também apresentaram baixa sensibilidade ao número de instâncias em estado de violação e atingiram acurácia razoável diante de *datasets* pequenos. Particularmente, esses três classificadores foram considerados as opções mais adequadas para modelar e prever problemas de desempenho para sistemas baseados em *middleware*.

Para comparar o desempenho da aplicação MapReduce usando uma política de escalonamento baseada em CPU e outra baseada na probabilidade de violação, o classificador TAN foi implementado como parte da máquina de inferência do SMART. Esse fato comprovou experimentalmente que a política de escalonamento baseada na probabilidade de violação proporciona um melhor desempenho para aplicações MapReduce que uma política de escalonamento baseada apenas em CPU. Portanto, o modelo de desempenho capturado pelo SMART revelou-se mais efetivo que o modelo baseado meramente na utilização de CPU.

Apesar dos classificadores NB, AODE e TAN terem sido considerados os classificadores mais adequados para o problema de estimação proposto neste trabalho, é importante destacar que três problemas podem comprometer a eficácia do uso dessas técnicas na construção de ferramentas autonômicas de gerenciamento. O primeiro problema se refere à taxa de alarme falso dos algoritmos e, nesse sentido, este trabalho confirma a constatação feita em outros trabalhos, em especial (Powers et al., 2005), que destacam a dificuldade de manter baixa a taxa de alarme falso dos métodos de classificação. O segundo problema refere-se à interpretabilidade dos modelos construídos. Muitos dos modelos derivados dos algoritmos analisados neste trabalho não são de fácil interpretação e este fato dificulta a utilização dos modelos como ferramentas que possam explicar as causas de um problema observado. Por fim, o terceiro problema refere-se ao próprio processo de aprendizado, o qual acontece uma única vez e de forma *offline*. No Capítulo 5 é proposto um mecanismos para

lidar com os dois primeiros problemas. O terceiro problema é abordado no Capítulo 6.