BOOSTED LAZY ASSOCIATIVE CLASSIFIER

VAUX SANDINO DINIZ GOMES

BOOSTED LAZY ASSOCIATIVE CLASSIFIER

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais – Departamento de Ciência da Computação como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Loïc Pascal Gilles Cerf

Belo Horizonte

Novembro de 2017

Gomes, Vaux Sandino Diniz

G633b Boosted lazy associative classifier / Vaux Sandino Diniz Gomes. — Belo Horizonte, 2017
xxi, 75 p.: il.; 29 cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Loïc Pascal Gilles Cerf

1. Computação – Teses. 2. Mineração de dados – Computação. 3. Aprendizado do computador. 4. Lazy associative classifier. 5. Boosting. I. Orientador. II. Título.

CDU 519.6*82 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

BOOSTED LAZY ASSOCIATIVE CLASSIFIER

VAUX SANDINO DINIZ GOMES

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. LOIC PASCAL GILLES CERF - Orientador
Departamento Ciência da Computação - UFMG

PROF. ADRIANO ALONSO VELOSO
Departamento de Ciência da Computação - UFMG

PROF. WAGNER MEIRA JÚNIOR
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 14 de Novembro de 2017.

Dedico à minha família e amigos.

"Simplicity is prerequisite for reliability."
(Edsger Dijkstra)

Resumo

Aprendizado de Máquina é uma subárea de Mineração de Dados que busca maneiras de conferir às máquinas a habilidade de aprender a partir de conjuntos de dados sem que estas sejam explicitamente programadas para tal tarefa. A máquina, então, deve ser capaz de extrair e generalizar informações de dados, e, posteriormente, usar estas informações para compreender dados nunca observados.

Algoritmos de classificação associativa extraem padrões frequentes existentes em conjuntos pre-classificados de dados transformando-os em regras capazes de predizer a classe de instâncias dos dados. As regras são humanamente inteligíveis, o que as tornam atrativas em casos onde é necessário entender ou explicar como o algoritmo chegou à cada predição de classe.

Frequentemente, algoritmos classificação associativa sofrem com problemas de processamento, dado o número padrões existentes nos dados. O Lazy Associative Classifier (LAC) supera essa adversidade decompondo o problema de procurar por todos os padrões em vários subproblemas menores. Contudo o LAC utiliza indiscriminadamente todas as regras que consegue formar. Isto pode levar à perda de acurácia e de interpretabilidade das classificações do algoritmo.

Neste trabalho, revisamos o processo do LAC e propomos o BLACk: um algoritmo de classificação associativa que utiliza *boosting* para montar um modelo aditivo com os mesmos padrões encontrados pelo LAC e que melhora a acurácia e a legibilidade do classificador. Foi possível comprovar estatisticamente que o BLACk é mais preciso que o LAC e que o número de regras do BLACk é algumas ordens de grandeza menor que o número de regras do LAC, o que o torna mais humanamente inteligível que o LAC.

Palavras-chave: Aprendizado de Máquinas, Classificação Associativa, Algoritmo *lazy*, *Boosting*.

Abstract

Lazy machine learning algorithms have to learn every time it is been given a new example, however knowing which example is being classified gives them the advantage of adjusting their knowledge search accordingly. The Lazy Associative Classifier (LAC) is a rule-based demand-driven lazy machine learning algorithm that takes advantage of the information present in the example being classified by focusing its effort on inducing only rules that cover that particular example. Each rule comes from a frequent pattern present in the data.

While, associative classifiers, in general, suffer from searching frequent patterns among the large number of existing patterns within the data, LAC breaks that problem down into many subproblems, solving one small problem at a time. Rule-based algorithms are often caught in the dilemma of not knowing the best way to combine their rules in order to form the best possible classifier. Usually, the choosing of a rule metric followed by a simple voting is used (as simple as assigning an importance – or weight – of one to each rule and averaging the accounts by each class). This approach is easily proven to be frail. Furthermore, LAC uses all rules available, which can be considered a large quantity of rules, regardless of their prediction quality.

In this work we use a boosting algorithm known as Confidence-Rated Adaboost in conjunction with LAC to form a new, more accurate and smaller (in number of rules present in each model) classifier algorithm called BLACk. We prove that our approach is superior in terms of accuracy to LAC and other associative classifier. Nevertheless, we show that the built classifiers model are less complex compared to those built by LAC.

Keywords: Data Mining, Associative Classification, Lazy Classifier, Boosting.

Lista de Figuras

4.1 Lazy Associative Classifier. 21

5.1 Visão geral do Confidence-Rated Adaboost. 26
5.2 Confidence-Rated Adaboost para regras de decisão. 30
5.3 SLIPPER: *Simple Learner with Iterative Pruning to Produce Error Reduction.* 34

6.1 Relação entre o BLACk e o LAC 36
6.2 Boosted Lazy Associative Classifier of k rules. 39
6.3 BLACk multiclasse. 44

7.1 Erros de treinamento normalizados do BLACk. 52
7.2 Complexidade dos padrões escolhidos 56
7.3 Acurácia dada a Complexidade dos Padrões disponíveis 59
7.4 Conjunto de dados Twitter: Acurácia 66

Lista de Tabelas

2.1 Conjunto de dados S (10 primeiras instâncias) seguido de uma instância a classificar. 6

4.1 Conjunto de dados projetados a partir de $X' = \{a_2, a_5, a_{10}\}$. 18
4.2 Lista de regras. 21

6.1 Exemplo de execução do BLACk. 41

7.1 Média de uso de regras vazias pelo BLACk comum e pelo BLACk-E (cuja projeção contém instâncias vazias). 60
7.2 Tamanho dos modelos. 61
7.3 Resultados: Acurácias 63
7.4 Resultados: Teste de normalidade e teste pareado de diferença 64

A.1 Informações gerais sobre os conjuntos de dados. 73

Sumário

Resumo **xi**

Abstract **xiii**

Lista de Figuras **xv**

Lista de Tabelas **xvii**

1 Introdução **1**

2 Definições e conceitos básicos sobre classificação **5**
2.1 Conjunto de dados 5
2.2 Algoritmo de classificação 7
2.2.1 Modos de operação 7
2.2.2 Qualidade do aprendizado 7
2.3 Padrões e padrões frequentes 8
2.4 Regras de decisão 8

3 Trabalhos relacionados **11**
3.1 Busca de padrões 11
3.1.1 Completa 11
3.1.2 Não-Completa 12
3.2 Classificação associativa 13
3.2.1 Modo *eager* 13
3.2.2 Modo *lazy* 14
3.3 Classificação por conjuntos de classificadores 15

4 Lazy Associative Classifier **17**
4.1 Projeção dos dados 17
4.2 Busca por padrões 18

4.3 A Geração de regras 20
4.4 Classificação 20
4.5 Algoritmo 20

5 Boosting **23**
5.1 Confidence-Rated AdaBoost 24
5.1.1 Limites do erro de treinamento 26
5.2 Uma versão do CRA para uso de regras 27
5.3 SLIPPER 31
5.3.1 Geração de regras 32
5.3.2 Algoritmo 33
5.3.3 Validação-Cruzada-Interna 34

6 Boosted Lazy Associative Classifier **35**
6.1 Primeiros conceitos 35
6.2 Algoritmo 38
6.3 Exemplo 39
6.4 Complexidade computacional 41
6.5 BLACk multiclasse 42
6.6 O uso de padrões fechados 45
6.7 Validação-Cruzada-Interna 45
6.8 Intâncias de fora da projeção 45
6.9 Comparações 46

7 Experimentos **49**
7.1 Erro de treinamento 49
7.2 Caracterização dos padrões escolhidos 52
7.3 Acurácia e complexidades 54
7.4 Instâncias de fora da projeção 56
7.5 Tamanho dos modelos 59
7.6 Acurácia dos modelos 62
7.6.1 Comparando os modelos 62
7.7 Aplicação em dados esparsos 65

8 Conclusão **67**

Referências Bibliográficas **69**

Anexo A Conjuntos de Dados **73**

B Propriedades matemáticas **75**
B.1 Propriedades dos radicais 75

Capítulo 1

Introdução

Não é possível contar a história da humanidade sem mencionar as máquinas. "Estas são os únicos frutos da criação humana que dedicam as vidas para nos atender" [Moraes & Abreu, 2006]. As máquinas têm como objetivo principal a resolução de problemas. Para isso, estas devem ser programadas, seja por meio de engrenagens que movimentam um relógio, seja por meio de instruções em linhas de código que definem os cálculos que um programa deve executar.

Existe, porém, problemas difíceis de serem programados de maneira explícita. Nestes casos, usam-se técnicas que permitem que as máquinas aprendam a solucionar tais problemas. Aprendizado de Máquina busca maneiras de conferir às máquinas esta habilidade de aprender a solucionar problemas sem que sejam explicitamente programadas.

Uma maneira de ensinar uma máquina a resolver um problema é apresentando exemplos deste problema juntamente com suas respectivas soluções. Espera-se que com um número suficiente de exemplos a máquina possa extrair e generalizar informação de modo a conseguir relacionar os exemplos com as soluções. Este processo caracteriza o Aprendizado Supervisionado.

Quando o problema apresentado à máquina possui um conjunto predefinido de soluções, então define-se uma tarefa de Classificação. Neste tipo de tarefa cada exemplo é denominado de instância e cada solução é denominada de classe. A máquina deve aprender a separar as instâncias do problema em classes. Em outras palavras, a máquina deve ser capaz de identificar a qual classe cada instância pertence. Algoritmos de classificação podem ser usados em áreas como medicina [Kononenko, 2001; Cruz & Wishart, 2006], análise de texto [Sebastiani, 2002], análise de mercado [Galindo & Tamayo, 2000], detecção de personalidade [Golbeck et al., 2011] etc. A qualidade das estratégias usadas pelos diferentes algoritmos pode ser dada pela acurácia, isto

é, a fração de classificações corretas de um conjunto de instâncias nunca vistas pela máquina.

Considere o contexto de uma empresa de cartão de crédito que recebe diariamente milhares de pedidos de crédito. Cada pedido contém várias informações sobre um cliente. Um pedido representa uma instância do problema. As classes são as possíveis soluções para o problema: "liberar" e "não liberar" crédito. Assim como um analista de crédito classifica cada solicitação de acordo com as informações presentes nos pedidos de crédito (liberando crédito ou não), uma máquina pode ser treinada para separar os pedidos por suas classes através das informações contidas no pedido em questão e nos pedidos que foram previamente classificados pelo analista (juntamente com as classificações atribuídas a estes). Desta maneira, as máquinas conseguem aprender tarefas que previamente eram realizadas por humanos especializados.

Neste trabalho abordaremos o *Lazy Associative Classifier* (LAC). Este é um algoritmo que extrai informações das instâncias em forma de padrões e utiliza esses padrões para gerar regras que podem ser usadas no momento da classificação. O LAC é categorizado como um algoritmo de classificação associativa. Tais algoritmos são competitivos com relação a outras estratégias de classificação. As regras geradas são humanamente interpretáveis – cada regra compreende um padrão que consiste de uma seleção das informações contidas nas instâncias. Em outras palavras, podemos ler e entender o porquê de uma dada instância ter sido classificada pelo algoritmo como sendo de certa classe. Esta é uma característica atrativa, por exemplo, em tarefas relacionadas à medicina (podendo explicar o motivo de um diagnóstico) e processamento de texto (explicando o sentimento de um texto a partir das palavras contidas neste).

Algoritmos de classificação associativa podem sofrer com problemas de velocidade. A maioria destes algoritmos buscam por padrões seletos dentre uma quantidade possivelmente enorme de padrões contidos nos dados de treinamento (isto é, o conjunto de todas as instâncias cujas classes são conhecidas à máquina) antes de classificar qualquer instância do problema (chamamos esse modo de operação de *eager*). Este modo de busca pode ser uma tarefa computacionalmente cara ou até inviável (explicaremos com mais detalhes na próxima seção), por isso o LAC opta por esperar até o momento da classificação de uma instância (chamamos esse modo de operação de *lazy*) focando a busca apenas naqueles padrões que contenham alguma informação em comum com a instância sendo classificado. Desta forma, o algoritmo evita ter que buscar por todos os padrões e diminui o custo da tarefa.

Após geradas as regras, muitos algoritmos diferem-se na maneira de usá-las. Enquanto alguns algoritmos selecionam um conjunto de regras para classificar cada instância, outros preferem uma única regra chamada de "a melhor regra". O LAC utiliza

todas as regras geradas, indiscriminadamente. Isto pode levar à perda de acurácia por parte do classificador – em outras palavras, pode levar à perda da qualidade preditiva do classificador. A interpretabilidade da classificação também é afetada se um grande número de regras é usado para classificar uma instância.

O objetivo deste trabalho é melhor combinar as regras do LAC por meio de uma técnica chamada *boosting*. Esta técnica é conhecida por aumentar a acurácia de algoritmos de classificação. Isto é, o *boosting* é um algoritmo que combina classificadores com taxas de error menor que 50% em um modelo classificador cuja taxa de erro é arbitrariamente menor que os classificadores que o compõe [Schapire, 2003]. Quando aplicado ao LAC, o *boosting* possibilita o aumento de acurácia do classificador e também um controle sobre o número de regras usadas. Assim, pretendemos criar um classificador mais acurado e mais legível. Sendo estas, as duas motivações deste trabalho. Chamamos a nossa abordagem de BLACk (*Boosted Lazy Associative Classifier of k rules*).

Organização do trabalho. Este trabalho está organizado como segue: a Seção 2 introduz os principais conceitos usados no restante do trabalho; a Seção 3 lista uma série de trabalhos relacionados à algoritmos de classificação e ao *boosting*; a Seção 4 descreve o processo do LAC; a Seção 5 descreve a técnica de *boosting* que será aplicada ao LAC; a Seção 6 introduz o BLACk, a nossa abordagem derivada da união do LAC com o *boosting*; finalmente, a Seção 7 descreve, exibe e discute os resultados dos nossos experimentos; e a Seção 8 sumariza o nosso trabalho.

Capítulo 2

Definições e conceitos básicos sobre classificação

Em uma tarefa de classificação supervisionada assume-se que a máquina tem acesso a um conjunto de dados de treinamento composto por uma série de instâncias de um mesmo problema e suas respectivas classes. Espera-se que a máquina seja capaz de extrair e generalizar informação de modo a conseguir relacionar as instâncias com as classes. Para isso, ela deve construir uma função de mapeamento entre as instâncias e as classes do problema. Esta função, posteriormente, pode ser usada para prever a classe de instâncias nunca vistas. Neste trabalho, a função de mapeamento será construída a partir da combinação de regras de decisão.

2.1 Conjunto de dados

Seja Y um conjunto finito de classes e I um conjunto finito de atributos Booleanos. Durante a maior parte desse trabalho assumiremos que $Y = \{-1,\ +1\}$ ou, simplesmente $\{-, +\}$. Um conjunto de dados de treinamento é um multiconjunto $S = \{(X, y) \mid X \subseteq I,\ y \in Y\}$. Cada par (X_i, y_i) é chamado de instância. A Tabela 2.1 exibe uma representação de S. Cada linha numerada representa uma instância de S e cada coluna a_j representa um atributo de I. Se $a_j \in X_i$ então existe uma marcação $\times$ no cruzamento da linha da instância (X_i, y_i) com o atributo a_j (caso $a_j \notin X_i$, então não existe a marcação). A última linha da tabela é uma instância a classificar. Esta instância não pertence a S, pois sua classe é desconhecida – a máquina é responsável pela classificação da instância.

O exemplo da empresa de cartão de crédito, comentado na seção anterior, pode ser representado pela tabela abaixo da seguinte maneira: Cada linha da tabela (isto é, cada

instância) representa um pedido de crédito; cada atributo representa uma informação dos pedidos. Assuma que as informações existentes são oriundas de perguntas de sim/não. Por exemplo, se o cliente que requisitou crédito já teve algum pedido de crédito negado. Dependendo da pergunta, a resposta mais pertinente pode variar. Isto é, podemos estar mais interessados em saber se uma dada pergunta foi respondida com um "sim", enquanto para outra pergunta o "não" pode ser uma informação mais pertinente. Assim, $a_j \in X_i$ implica em dizer que a resposta da instância da linha i para a pergunta representada pelo atributo a_j foi resposta tida como pertinente, logo deve existir uma marcação $\times$ no cruzamento da linha i como atributo a_j. Podemos ainda ter dois atributos a_j e a_k mapeando uma mesma pergunta, onde $a_j = \bar{a_k}$. A última linha da tabela representa um pedido novo que espera por classificação.

#	Classe	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
1	-	×			×			×			
2	+	×			×			×			×
3	-		×			×			×		
4	+			×			×		×		
5	-	×			×			×			
6	-			×	×			×			
7	-	×			×			×			
8	-			×		×		×			
9	-		×				×		×		×
10	+			×			×		×		×
t	?		×			×				×	×

Tabela 2.1: Conjunto de dados S (10 primeiras instâncias) seguido de uma instância a classificar.

Dizemos que os dados da tabela acima estão em um formato binário, dado que as marcações informam a presença ou a ausência de um certo atributo. Em outras palavras, os valores 1 (se atributo presente) e 0 (se atributo ausente) poderiam ser utilizados para representar as mesmas informações exibidas na tabela. Algoritmos de classificação associativa trabalham com esta representação de dados. Se um conjunto de dados é não binário, este deve passar por um processo de binarização. Comumente cada atributo não binário passa por um processo de *binning*, onde os valores de um atributo (tipicamente numérico) são agrupados em categorias (tipicamente particionando os valores possíveis em intervalos). Por exemplo, um atributo não binário usado para informar idade pode ser substituído por categorias como jovem, adulto e idoso. Então, cada categoria pode ser representada por um atributo binário. Portanto, temos agora tantos atributos binários quanto categorias e somente um dos atributos que representam esta categoria estará ativo. Logo, se a categoria "jovem" estiver ativa (1), as demais

estarão desativadas (0). O processo pode levar à perda de informação e ao aumento da dimensionalidade dos dados.

2.2 Algoritmo de classificação

2.2.1 Modos de operação

Algoritmos de classificação passam por uma fase de treinamento onde recebem um conjunto de dados de treinamento S e devem ser capazes de inferir uma função $f_S : X \rightarrow Y$. A função inferida é chamada de modelo classificador (ou simplesmente, classificador). Dependendo do modo de operação do algoritmo, esta fase de treinamento pode acontecer uma única vez (modo *eager*) ou sempre que existir uma instância a classificar (modo *lazy*).

Algoritmos de classificação *eager* constroem um único modelo classificador que é usado para classificar qualquer instância de classe desconhecida (como é o caso da instância t da Tabela 2.1). Algoritmos de classificação *lazy*, por sua vez, constroem um classificador para cada instância a ser classificada. O classificador formado é usado apenas para classificar a instância que deu início ao processo do algoritmo, depois disto pode ser removido da memória.

2.2.2 Qualidade do aprendizado

Podem-se usar diferentes medidas que informam a qualidade de um classificador, como acurácia, precisão, revocação, etc. Cada uma destas medidas evidencia determinado aspecto de qualidade do classificador. A acurácia será usada como medida de qualidade. Comumente, a acurácia (assim como as demais medidas de qualidade) é calculada através de validação-cruzada [Stone, 1974]. de k-dobras.

A validação cruzada consiste em fatiar o conjunto de treinamento em k conjuntos de instâncias de tamanhos iguais (ou aproximadamente iguais), opcionalmente mantendo-se as mesmas proporções de classe em cada fatia. Em um total de k iterações, a fatia k é usada para compor o conjunto de validação V_k e as demais fatias são usadas para compor o conjunto de treinamento S_k. Cada instância do conjunto de validação (chamadas de instâncias de teste) tem sua classe omitida à máquina. Para um algoritmo da classificação *eager*, na iteração k forma-se um classificador M_k usando o conjunto de treinamento S_k, então M_k é usado para classificar todas as instâncias em V_k. Ao fim das k iterações temos a predição de classe para cada instância de teste e

podemos comparar aquele valor com a classe verdadeira daquela instância. A acurácia, então, é a fração de instâncias de teste corretamente classificadas.

2.3 Padrões e padrões frequentes

Classificadores associativos são construídos a partir da combinação de regras originadas de padrões encontrados nos dados. Um padrão A é um subconjunto de elementos de I. Logo, existem $2^{|I|}$ padrões nos dados. O tamanho de um padrão é dado pelo número de atributos binários que o compõe.

Os padrões usados pelos algoritmos de classificação são tipicamente escolhidos por ultrapassar um limiar mínimo de qualidade definido pelo usuário. Várias medidas de qualidade podem ser usadas (vide Geng & Hamilton, 2006). A abordagem a ser apresentada usa padrões frequentes para montar suas regras. Um padrão é dito frequente se atende a um limiar mínimo de **suporte** escolhido pelo usuário da aplicação. O suporte de um padrão é a fração de instâncias dos dados em que o padrão ocorre:

$$supp(A, S) = \frac{|\{(X, y) \in S \mid A \subseteq X\}|}{|S|}. \tag{2.1}$$

Observando a Tabela 2.1, seja $A_1 = \{a_1, a_4, a_7\}$, $A_2 = \{a_2, a_5\}$ e $s = 3/10$ o suporte mínimo para que um padrão seja considerado frequente. Uma vez que $supp(A_1, S) = 4/10$ e $supp(A_2, S) = 1/10$, A_1 é frequente e A_2 não é.

2.4 Regras de decisão

Seja uma regra de decisão uma implicação na forma $A \rightarrow c$, onde $A \subseteq I$ é chamado padrão da regra e $c \in Y$ é chamado de classe da regra. Cada atributo binário $\{a_1, \ldots, a_m\} \in A$ é entendido como uma condição binária na forma $a_i \in X$, onde X é o conjunto de atributos de uma instância a classificar.

Algoritmos de classificação associativa trabalham apenas com dados binários, contudo outros algoritmos conseguem trabalhar com diferentes tipos de dados. Nestes casos, o padrão da regra é formada por condições nas seguintes formas: $a_n = v$, onde a_n é um atributo nominal e v é um valor legal para a_n; ou $a_c \leq \theta$ ou $a_c \geq \theta$, onde a_c é um atributo contínuo e θ é um valor qualquer que ocorre para a_c nos dados de treinamento.

Cobertura Se o padrão da regra ocorre em um instância a classificar, dizemos que a regra cobre a instância. Esta regra pode ser usada para classificar a instância em questão.

Confiança A confiança de uma regra indica quanto o padrão da regra está associado à classe da regra. A confiança é a fração de instâncias do conjunto de treinamento em que o padrão da regra ocorre cuja a classe da instância coincide com a da regra:

$$conf(A \to c, S) = \frac{|\{(X, c) \in S \mid A \subseteq X\}|}{|\{(X, y) \in S \mid A \subseteq X\}|}. \tag{2.2}$$

Complexidade A complexidade de uma regra será dada por $|A|$. Não existe, de fato, um consenso quanto à definição da complexidade, mas, como esperamos que os atributos das regras sejam lidos, esta é considerada uma boa medida.

Ordem A ordem da regra está também associada a $|A|$. De modo que, durante o curso deste trabalho, quanto maior for a complexidade de uma regra, isto é, quanto mais atributos existirem em A, maior será a ordem da regra.

Capítulo 3

Trabalhos relacionados

3.1 Busca de padrões

A busca de padrões pode ser feita de maneira completa ou não-completa. Por definição, os algoritmos completos mineram todos os padrões que satisfaçam uma determinada medida de qualidade escolhida pelo usuário da aplicação. Uma medida de qualidade bastante comum é a frequência, indicada pelo valor de suporte do padrão. A busca completa de padrões possui complexidade exponencial no número de atributos, o que pode tornar o processo computacionalmente caro ou até inviável. Além disso, muitos algoritmos de busca completa de padrões exigem várias passagens sobre os dados para calcular as métricas de qualidade escolhidas pelo usuário. Assim, abordagens de busca não-completa foram propostas afim de prover uma maneira rápida de encontrar padrões frequentes.

3.1.1 Completa

Um algoritmo de busca completa de padrões frequentes bastante conhecido é o Apriori [Agrawal et al., 1994]. O algoritmo busca em uma lista de padrões candidatos de tamanho k e seleciona os padrões que forem frequentes. A frequência é determinada através de *pattern matching* que funciona comparando os atributos do padrão com os de cada instância do conjunto de treinamento. Logo após, o algoritmo combina estes padrões para montar a próxima geração de candidatos, de modo que para a geração k os padrões frequentes de tamanho k são combinados afim de gerar padrões candidatos de tamanho $k + 1$. É sabido que o Apriori tem problemas de velocidade. Primeiro, é computacionalmente custoso trabalhar com um grande número de padrões candidatos. Han et al., 2000 argumentam que se existirem 10^4 padrões frequentes

de tamanho 1, o Apriori precisará gerar mais de 10^7 padrões candidatos de tamanho 2. O mesmo autor também argumenta que para descobrir um padrão frequente de tamanho 100, o algoritmo precisará gerar $2^{100} \approx 10^{30}$ padrões candidatos. Segundo, é computacionalmente custoso calcular a frequência de cada um dos padrões candidatos, especialmente para conjuntos de dados com muitas instância.

Outro algoritmo de mineração completa de padrões é o FP-Growth [Han et al., 2000]. O FP-Growth utiliza de uma estrutura de árvore (chamada *Frequent Pattern Tree* – FP-Tree) durante o processo de descoberta de padrões para guardar informações quantitativas sobre a frequência dos padrões. A estrutura garante que os padrões tenham suas frequências calculadas mais rapidamente que o método do Apriori através de travessias recursivas na árvore. A busca de padrões é feita de maneira completa e os padrões ficam codificados na própria estrutura de árvore.

3.1.2 Não-Completa

A busca não-completa de padrões procura maneiras rápidas de encontrar padrões frequentes nos dados através de heurísticas. Segundo Dass & Mahanti, 2006, duas das maiores áreas de desenvolvimento de heurísticas de busca de padrões são: heurísticas que tentam prever a frequência de padrões de tamanho k; Heurísticas que tentam prever o número máximo de padrões candidatos e até que nível (tamanho dos padrões) podem ocorrer padrões de frequência mínima.

Em [Dass & Mahanti, 2006], foi proposto uma heurística de busca de padrões frequentes em tempo-real. O algoritmo funciona como o Apriori, gerando uma lista de padrões candidatos e verificando se eles são frequentes (ou heuristicamente frequentes). Após determinado o valor mínimo de suporte, o algoritmo lista todos os padrões frequentes de tamanho 1 e 2 contidos nos dados. A heurística então utiliza estes padrões para montar uma lista de padrões candidatos de tamanho 3. Para cada padrão candidato calculam-se valores de *lower* e *uper-bound* (sendo este último calculado heuristicamente). A lista de padrões frequentes de tamanho $k + 1$ é formada não somente por padrões frequentes (como acontece no Apriori), mas também pelos denominados heuristicamente frequentes. Um padrão é denominado heuristicamente frequente se seu valor de *lower-bound* é superior ao valor mínimo de suporte e se nenhum dos subconjuntos deste padrão for infrequente.

3.2 Classificação associativa

Diversos estudos [Li et al., 2001; Machado, 2003; Thabtah et al., 2004] evidenciam que classificadores formados por conjuntos de regras de decisão são competitivos com os constituídos por árvore de decisão [Quinlan, 1993], regras de indução [Quinlan & Cameron-Jones, 1993; Cohen, 1995] e abordagens probabilísticas [Duda et al., 1973].

A maioria dos algoritmos de classificação associativa trabalham de modo *eager*, isto é, procuram por todos os padrões frequentes contidos nos dados e geram um classificador composto por regras que é usado para classificar qualquer nova instância de dado. Alguns algoritmos trabalham de modo *lazy*. Estes esperam até o momento em que uma instância deve ser classificada para buscar por padrões e gerar seus modelos de regra. Assim, enquanto algoritmos *eager* geram um modelo para predizer qualquer nova instância, os algoritmos *lazy* geram um modelo para cada instância de teste.

3.2.1 Modo *eager*

O CBA [Ma, 1998; Liu et al., 2001] é um algoritmo de classificação associativa que usa apenas uma das regras geradas para classificar cada instância. O algoritmo foi um dos primeiros a utilizar o Apriori na busca por padrões. Após encontrar todos os padrões frequentes, o conjunto de regras gerado é ordenado de acordo com os valores de confiança das regras (desempatadas por seus suportes). A classe de uma instância é dada pela primeira regra, em ordem, que a cobrir. Esta estratégia é chamada de "melhor regra". Tal abordagem afeta a acurácia do classificador, uma vez que a regra utilizada pode não ser a mais indicada para classificar a instância [Li et al., 2001; Gambhir & Gondaliya, 2012]. Para um problema com apenas duas classes, é possível que a "melhor regra" classifique uma instância como sendo de uma classe, enquanto a maior parte das regras restantes que cobrem a instância classificariam-na como sendo da outra classe.

De fato, não é uma tarefa fácil selecionar a regra que melhor classifica cada instância, por isso muitos algoritmos optam por usar conjuntos de regras para determinar a classe de cada instância, por exemplo o CMAR [Li et al., 2001]. O método busca por padrões usando o FP-growth [Han et al., 2000]. Dada uma instância a ser classificada, o algoritmo seleciona um conjunto pequeno de regras de decisão que cobrem a instância, que possuem valor mínimo de confiança e χ^2 (uma métrica que mede a correlação entre o padrão da regra e a classe da regra), determinados pelo usuário. A classe da instância é dada por uma métrica empírica denominada *weighted* χ^2 (vide Li et al., 2001), os autores atestam que esta métrica busca minimizar o viés que existe em favor

da menor classe causado pelo próprio χ^2. O CMAR, em geral, tem maior precisão que o CBA e o C4.5 [Quinlan, 1993] (vide Li et al., 2001). Contudo, é sabido que a abordagem utilizada pelo CMAR trás consigo problemas de velocidade (vide Gambhir & Gondaliya, 2012).

Outro algoritmo *eager* de classificação associativa é o HARMONY [Wang & Karypis, 2005]. A ideia por trás do HARMONY é a construção da "melhor regra" possível a partir de cada instância do conjunto de treinamento. A qualidade da regra é dada por sua confiança. O conjunto de regras é dividido por classes e cada subconjunto é ordenado pela confiança e, se houver empate, pelo suporte. Dada uma instância a ser classificada, o algoritmo computa dentre as regras que cobrem a instância: 1. o valor de confiança da primeira regra (lembrando que as regras estão em ordem); 2. a soma das confianças de todas as regras; 3. e a soma das confianças das primeiras k regras, onde o valor de k é definido pelo usuário da aplicação. A classe predita é a que possuir a maior soma dos três quesitos.

3.2.2 Modo *lazy*

De acordo com Wettschereck et al., 1997, a maioria dos algoritmos *lazy* vêm de variações do Naïve Bayes (como o LBR [Webb et al., 2005] e o LBN [Jiang & Guo, 2005]) ou do KNN [Dasarthy, 1990]. Algoritmos de classificação *lazy* têm em comum o fato de ser necessário manter o conjunto de treinamento em memória durante toda a sua vida útil para reaprenderem os modelos. Enquanto isso, algoritmos de classificação *eager* aprendem o modelo apenas uma vez e não necessitam mais do conjunto de dados.

DeEPs [Li et al., 2004] é um algoritmo *lazy* de classificação associativa que utiliza padrões emergentes em suas regras. Para isso, o conjunto de dados é particionado em vários subgrupos de acordo com suas classes e a diferença entre o suporte de cada padrão em cada subgrupo é calculado. Aqueles padrões cujo valor calculado está acima de um limiar predefinido são entendidos como emergentes, ou seja, este captura uma informação de discriminação de classe. Sempre que uma nova instância está sendo classificada, o DeEPs usa um filtro para remover, do conjunto de treinamento, instâncias que não tenham atributos em comum com a instância em classificação. Em seguida, os padrões emergentes são encontrados. O algoritmo classifica a instância somando os suportes dos padrões em cada classe e verificando qual classe possui o maior valor acumulado.

O LAC se enquadra nessa categoria de algoritmos e, de fato, também faz uma filtragem nos dados de treinamento como o DeEPs. Contudo o LAC não utiliza padrões emergentes e usa um método diferente para classificar as instância. Mais detalhes sobre

o LAC serão vistos na Seções 4.

3.3 Classificação por conjuntos de classificadores

Dada uma instância a classificar é possível usar um conjunto de classificadores, combinando suas classificações individuais (tipicamente através de voto), para classificar a instância [Dietterich, 2000]. Bagging [Breiman, 1996] e Boosting [Schapire & Singer, 1999] são exemplos de algoritmos que usam conjuntos de classificadores.

Para que o conjunto de classificadores seja mais acurado que qualquer um dos classificadores individuais, é necessário que cada classificador do conjunto seja *moderadamente preciso* e *diversificado* [Hansen & Salamon, 1990]. Um classificador é dito moderadamente preciso se este consegue predizer com uma taxa de acerto maior que 50% a classe de instâncias de fora do treinamento. Dois classificadores são ditos diversos/diversificados se estes erram a classificação de diferentes instâncias. Em outras palavras, se um classificador erra onde outro acerta.

Em Sun et al., 2006 é proposto o emprego de regras de associação e técnicas de *boosting* na construção de modelos de classificação. O trabalho consiste em usar o HPWR [Wang, 1998] (que gera regras de alta ordem a partir de análise residual) juntamente com o Adaboost [Schapire, 1990]. Em seus experimentos, os autores, empiricamente, testam modificações na maneira como as classificações individuais são combinadas a fim de determinar a decisão final do conjunto de classificadores. Seus resultados mostraram uma pequena melhora de acurácia. O classificador gerado pelo Adaboost foi melhor quando comparados ao C4.5, RIPPER [Cohen, 1995] e Naïve-Bayes. Os autores justificam que o Adaboost tende ao *overfitting* (super aprendizado ou memorização das instâncias do treinamento) mais rapidamente à medida que os classificadores escolhidos são mais complexos, isto é, as regras são de maior ordem. Esta abordagem difere-se do nosso trabalho porque trabalhamos em modo *lazy* (gerando apenas regras que interessam à classificação de cada instância específica) e não fazemos distinção entre as ordens das regras geradas. Nos interessa usar o *boosting* para melhor combinar as regras do modelo.

O uso da técnica do *boosting* não está limitada a apenas o uso desta com algoritmos de classificação por regra. De fato o *boosting* é uma metaheurística capaz utilizar diferentes algoritmos de aprendizado de máquina. Por exemplo, em [Drucker et al., 1993] encontramos o uso do *boosting* utilizando redes neurais. Este trabalho é focado em melhorar o uso de conjuntos de redes neurais (*ensambles*) em problemas de reconhecimento de caracteres. De fato, os autores não usaram um conjunto de mais

que três redes, tanto pela complexidade computacional da classificação (tornando o processo lento), mas também porque conseguiram melhorar seus resultados em fator de três com apenas estas poucas redes neurais. Já em [Tieu & Viola, 2004] vemos a utilização do *boosting* em tarefas de busca de imagens (*image retrieval*) à partir de consultas de palavras chave. O trabalho consiste usar o *boosting* para selecionar um conjunto de filtros primitivos de imagens à fim de classificar imagens em repositórios não indexados de maneira eficiente.

Por último devemos citar o XGBoost [Chen & Guestrin, 2016], um *boosting* de árvores escalável que admite dados esparsos, criado para utilizar múltiplas GPUs. Este algoritmo em especial tem recebido bastante reconhecimento em vários sites de problemas de aprendizado de máquina como o Kaggle em tarefas como predição de vendas, classificação de texto na web, predição de comportamento de clientes, detecção de movimento, etc. O XGBoost utiliza o *gradient tree boosting*, técnicas de regularização de modelos de árvore, amostragem de colunas/atributos e técnicas de pararelismo [Friedman, 2001; Breiman, 2001; Chen & Guestrin, 2016]. Apesar de seu sucesso, o XGBoost é citado aqui como uma forma de preconizar o ganho que pode ser trazido da aplicação de técnicas de *boosting*.

Capítulo 4

Lazy Associative Classifier

LAC (*Lazy Associative Classification*) é um algoritmo de aprendizado de máquina que constrói classificadores associativos de regras de modo *lazy*. Isto significa que o algoritmo escolhe treinar, isto é, formar o seu classificador somente após observar a instância a classificar. Em outras palavras, o trabalho do algoritmo é orientado a instância e o classificador construído é usado apenas para classificar a instância que deu início ao processo do algoritmo. Este modo de funcionamento difere-se do método mais convencional de geração classificadores, o modo *eager*, que constrói um único classificador e posteriormente utiliza este para classificar qualquer nova instância de dado.

Em Friedman et al., 1996, afirma-se que construir um único classificador que seja bom em predizer todas as instâncias dos dados pode resultar em classificadores complexos, uma vez que estes não tiram vantagem das características das instâncias a classificar. De outra forma, poderia-se construir um classificador capaz de nos dar uma explicação extremamente curta sobre a classificação da instância em mãos.

O LAC consegue usufruir da vantagem de ser *lazy* reduzindo o conjunto de treinamento e estreitando o espaço de busca dos padrões. Dada uma instância a classificar t (ou seja, uma instância de classe desconhecida), o LAC passa por três etapas até a classificação desta instância: Projeção dos dados, responsável por reduzir os dados de treinamento com base em t; Busca por Padrões; e Geração de Regras, cujo propósito é predizer a classe de t.

4.1 Projeção dos dados

A projeção dos dados é a primeira fase do LAC. O objetivo desta fase é filtrar os dados de treinamento, isto é, reduzir os atributos que descrevem as instâncias do treinamento

à somente aqueles atributos presentes na instância a classificar (mantendo as suas respectivas informações de classe). A projeção dos dados não afeta a busca dos padrões já que são processos independentes (vide Seção 4.2). Ela serve apenas para facilitar o cálculo das métricas de qualidade de padrões e regras usadas pelo LAC.

Dado X', o conjunto de atributos de t, uma instância a classificar, e S, o conjunto de dados de treinamento, a projeção P_t é dada por:

$$P_t = \{(X' \cap X, c) \mid (X, c) \in S \wedge X' \cap X \neq \emptyset\}. \tag{4.1}$$

Via de regra a projeção possui menos elementos e menor dimensionalidade que o conjunto de dados de treinamento. Para exemplificar, considere mais uma vez o conjunto de dados exibido na Tabela 2.1. A projeção dos dados a partir da instância t (última linha da tabela) é exibida na tabela abaixo. Pode-se perceber que tanto instâncias quanto atributos foram removidos.

#	Classe	a_2	a_5	a_{10}
2	+			×
3	-	×	×	
8	-		×	
9	-	×		×
10	+			×

Tabela 4.1: Conjunto de dados projetados a partir de $X' = \{a_2, a_5, a_{10}\}$.

A projeção é, de fato, um passo importante do processo de aprendizado do LAC porque essa tem o potencial de minimizar o custo computacional vinculado à contagem de instâncias para calculo de métricas de qualidade. Na prática, apenas um simples vetor com os índices das instâncias poderia ser usado para representar a projeção. Contudo, precisaremos calcular valores de suporte e confiança para os padrões e regras, então fez-se mais didático e conveniente que a projeção fosse definida nos mesmos moldes de S a fim de que possamos usar as equações de confiança e suporte definidas na Seção 2.

4.2 Busca por padrões

A maneira como os padrões são encontrados não é propriamente o foco desta pesquisa e, de fato, a estratégia usada para encontrar os padrões frequentes não interessa ao LAC, a este somente importa ter os padrões em mãos para que as regras sejam construídas.

O LAC não busca todos os padrões contidos nos dados, somente aqueles que têm algum atributo em comum com a instância t e que sejam frequentes (isto é, que possuam

um valor de suporte considerado aceitável pelo usuário da aplicação). O conjunto de padrões que estamos interessados é:

$$F_t = \{A \mid A \subseteq X' \wedge supp(A, P_t) \geq s\}, \tag{4.2}$$

onde s é o valor de suporte mínimo predefinido pelo usuário da aplicação. Na prática, s é obrigatoriamente um valor maior que zero. Perceba que nesta fase não nos interessa a informação de classe contida na projeção (como exibido na Figura 4.1) porque o cálculo do suporte não leva em consideração esta informação.

Todos os padrões em F_t estão contidos na instância a classificar. Isto significa que qualquer regra gerada a partir destes padrões cobre a instância t e, portanto, tem a capacidade de classificá-la. Quaisquer outras regras não cobririam a instância e, logo, não poderiam ser usadas para classificar t.

No pior caso, o número de padrões encontrados será $|F_t| = 2^{|X'|}$. Isto ocorreria quando $s \approx 0$. À medida que aumentamos o valor de s, o número de padrões em F_t diminuirá. Ainda assim o número de padrões em F_t tende a ser menor que o número de padrões totais existentes nos dados ($2^{|I|}$, para $s \approx 0$). A diferença está no fato de buscarmos apenas os padrões que se aplicam a X' em t. Assim, a complexidade da busca por padrões depende do número de atributos em t, não do número total de atributos em I.

No exemplo da Tabela 2.1, onde $X' = \{a_2, a_5, a_9, a_{10}\}$. O número máximo de padrões em F_t acontece quando $s \approx 0$. Assim, F_t possui $2^4 = 16$ padrões. Em contrapartida, o número total de padrões nos dados é de $2^{10} = 1024$.

No decorrer de várias chamadas ao algoritmo de busca de padrões, um mesmo padrão pode ser encontrado mais que uma vez. Para isso basta pelo menos duas instâncias a classificar tenham atributos em comum. Portanto, seria possível economizar processamento guardando-se, por exemplo, os padrões encontrados mais frequentemente. Todavia, esta é uma discussão na qual não nos aprofundaremos.

Por fim, como já argumentado, diferentes algoritmos poderiam ser utilizados para encontrarmos os padrões em F_t, por exemplo, o Apriori, sendo somente necessário fazer com que os padrões candidatos da primeira iteração sejam apenas os subconjuntos unitários de X'.

4.3 A Geração de regras

De fato, o LAC utiliza um método simples para gerar suas regras. O algoritmo associa cada padrão em F_t às classes de Y. O conjunto de regras é dado como a seguir:

$$R_t = \{A \to c \mid A \in F_t \ \wedge \ c \in Y \wedge conf(A \to c) \geq q\} \tag{4.3}$$

onde q é o valor de confiança mínima predefinia pelo usuário da aplicação. Cada regra em R_t cobre a instância t, uma vez que todos os padrões em F_t estão contidos em t. Portanto, todas as regras geradas podem classificar a instância. A cardinalidade de R_t depende do número de padrões em F_t e do número de classes em Y, de modo que $|R_t| = |F_t| \times |Y|$. Ainda assim, dependendo do número de atributos em t, é possível que existam centenas de regras em R_t.

Na prática, a não ser que o usuário da aplicação queira saber quais regras foram usadas para classificar determinada instância, não é necessário guardar as regras em memória, pois estas serão usadas apenas para classificar a instância que deu início ao processo do algoritmo.

4.4 Classificação

O LAC utiliza todas as regras encontradas para classificar a instância que deu início ao processo do algoritmo. Seja $R_t^c = \{(A \to c) \in R_t \wedge c \in Y\}$ o conjunto de todas as regras de R_t que apontam para a classe c (ou, em outras palavras, cuja classe é c). Dados o conjunto de regras R_t geradas e a respectiva projeção P_t, a classe predita pelo LAC é a classe cuja média das somas das confianças das regras é majoritária como segue:

$$\arg\max_{c \in Y} \left(\frac{1}{|R_t^c|} \sum_{(A \to c) \in R_t^c} conf(A \to c, P_t) \right) \tag{4.4}$$

Empates são decididos escolhendo-se a classe mais frequente em P_t.

4.5 Algoritmo

O algoritmo do LAC é exibido na Figura 4.1. O algoritmo projeta o conjunto de treinamento a partir de t, formando P_t. Em seguida, a função *MineraPadrões* busca por padrões frequentes, formando F_t. Dois laços de repetição são utilizados para formar

R_t combinando as classes do conjunto de dados, os padrões encontrados e verificando o valor de confiança mínima determinado pelo usuário da aplicação. No algoritmo abaixo R_t foi armazenado em memória pela didática deste documento, mas apenas a soma das confianças maiores que q e o número de regras geradas em cada classe seriam suficientes para determinar a classe da instância t.

Entrada: S, um conjunto de treinamento; t, a instância a ser classificada;
: s, suporte mínimo; q, confiança mínima.

$P_t := \{(X' \cap X, c) \mid (X, c) \in S \wedge X' \cap X \neq \emptyset\}$; `// Gerando projeção`
$F_t := \mathit{MineraPadrões}(P_t, s)$; `// Minerando padrões frequentes`

for $c \in Y$ **do**
  for $A \in F_t$ **do**
    $r := (A \rightarrow c)$;
    if $conf(r, P_t) \geq q$ **then**
      $R_t := R_t \cup r$;
    end
  end
end

Retorne:

$$\arg\max_{c \in Y} \left(\frac{1}{|R_t^c|} \sum_{(A \rightarrow c) \in R_t^c} conf(A \rightarrow c, P_t) \right)$$

Figura 4.1: Lazy Associative Classifier.

Para exemplificar, fixemos o limiar de suporte $s = 2/5$ e consideremos a Tabela 4.1. Teremos que $F_t = \{\{a_2\}, \{a_5\}, \{a_{10}\}, \{\emptyset\}\}$. Assim:

Classe	F_t	Suporte	R_t	Confiança
$+$	$\{a_2\}$	$2/5$	$(\{a_2\} \rightarrow +)$	0
$-$			$(\{a_2\} \rightarrow -)$	1
$+$	$\{a_5\}$	$2/5$	$(\{a_5\} \rightarrow +)$	0
$-$			$(\{a_5\} \rightarrow -)$	1
$+$	$\{a_{10}\}$	$3/5$	$(\{a_{10}\} \rightarrow +)$	$2/3$
$-$			$(\{a_{10}\} \rightarrow -)$	$1/3$
$+$	$\{\emptyset\}$	1	$(\{\emptyset\} \rightarrow +)$	$2/5$
$-$			$(\{\emptyset\} \rightarrow -)$	$3/5$

Tabela 4.2: Lista de regras.

Para este exemplo, as médias da soma das confianças são $(0 + 0 + 2/3 + 2/5)/4 = 16/45 \approx 0,267$ para a classe '+' e $(1 + 1 + 1/3 + 3/5)/4 = 44/60 \approx 0,733$ para a classe '−'. Desta forma o LAC classifica a instância t como sendo da classe '−'.

Capítulo 5

Boosting

Boosting é um método geral que integra a ideia da combinação de um conjunto de classificadores fracos (não acurados) a fim de gerar um classificador forte (acurado). Um classificador fraco é, por definição, um classificador que consegue predizer corretamente pouco mais que 50% das instâncias do treinamento – fraco também pode ser entendido como *moderadamente impreciso*. Via de regra, tais classificadores são mais facilmente gerados (ou inferidos) que classificadores precisos, principalmente no tocante ao processamento computacional. O *boosting* funciona iterativamente, escolhendo apenas um classificador fraco por iteração para fazer parte do modelo final do algoritmo. o método oferece um controle quanto ao tamanho do conjunto de classificadores fracos utilizados. As vantagens deste controle serão argumentadas quando estivermos discutindo a nossa abordagem final, o BLACk. A predição final do modelo do *boosting* combina a predição individual de cada classificador fraco escolhido. Estas predições são combinadas por escalares que indica a importância individual da predição de cada classificador fraco na predição do modelo como um todo. O *boosting* é conhecido por conseguir melhorar a acurácia do modelo classificador final.

Nesta seção discutiremos um algoritmo publicado no final da década de 90 por Schapire chamado Confidence-Rated Adaboost (CRA). O algoritmo é uma metaheurística que segue a filosofia do *boosting* e pode ser usado em conjunto com um ou mais algoritmos de classificação. Resumidamente, buscaremos entender dois aspectos principais do *boosting*: como o algoritmo escolhe; e como o algoritmo combina os classificadores fracos para formar o seu classificador final. Posteriormente, descreveremos uma versão do CRA que usa classificadores que se abstêm. Então discutiremos um pouco sobre o SLIPPER. Este algoritmo utiliza o CRA para gerar e combinar um conjuntos de regras e serviu de inspiração para o BLACk.

Durante o restante desta seção usaremos a palavra modelo para simbolizar o *status*

final do processo do *boosting*, isto é, o conjunto de classificadores fracos escolhidos pelo algoritmo juntamente com o conjunto de pesos associados a estes. A partir daqui "classificador" significa "classificador fraco".

5.1 Confidence-Rated AdaBoost

O CRA é um algoritmo que trabalha de forma iterativa, selecionando classificadores fracos para montar seu modelo final. Um peso (valor real positivo) é associado a cada instância do treinamento. O processo assume que, a cada iteração, um conjunto de classificadores é provido ao algoritmo por um módulo chamado *base learner*. Um classificador, dentre os retornados pelo *base learner*, é escolhido em cada iteração para se tornar parte do modelo final do CRA. Escolhe-se o classificador que "melhor" classificar as instâncias do treinamento em cada iteração. Veremos isso com mais detalhes mais à frente. Ao final da iteração os pesos das instâncias são modificados, diminuindo-se (aumentando-se) os pesos das instâncias corretamente classificadas (incorretamente classificadas). O modelo final é um modelo aditivo dos classificadores individuais escolhidos nas iterações do algoritmo.

Para entender melhor, é preciso primeiramente definir alguns termos. Seja T o número de iterações do nosso algoritmo, $S = \{(X_1, y_1), \ldots, (X_m, y_m)\}$ o nosso conjunto de treinamento, onde $X_i \in \mathcal{X}$ e $y_i \in \{-1, +1\}$, e D o conjunto de valores reais positivos, chamados de pesos, associados às instâncias de S, onde D_t representa os pesos associados às instâncias de S na t-ézima iteração e $D_t(i)$ o peso associado à instância (X_i, y_i) na t-ézima iteração. Ao iniciar o algoritmo temos que $D_1(i) = 1/|S|$.

Temos que o *base learner* retorna um conjunto de classificadores na forma $h : \mathcal{X} \to \mathbb{R}$ a cada iteração. A saber, o sinal do valor retornado pelos classificadores indica a classe prevista por este e o valor absoluto é um grau de certeza da predição do classificador. Na t-ézima iteração escolheremos o classificador h, dentre os retornados pelo *base learner*, que minimiza:

$$Z = \sum_{i=1}^{m} D_t(i) \exp(-\alpha y_i h(X_i)). \tag{5.1}$$

Este valor representa o erro do classificador h. A certeza da classificação, a exatidão da predição e o peso da instância classificada influenciam o valor de Z para um dado classificador h. De fato, toda a ideia por trás deste algoritmo de *boosting* está em minimizar Z a cada iteração. Por enquanto, deixaremos para discutir sobre α mais a diante, pois ele é encontrado de maneira diferente para cada tipo de classificador.

Neste ponto, basta saber que ele é um valor real a ser calculado.

Denominamos h_t o classificador escolhido na iteração t, e Z_t o seu respectivo valor de Z calculado como acima. Ao fim da t-ézima iteração, atualizamos os pesos das instâncias como segue:

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(X_i))}{Z_t}. \tag{5.2}$$

Atualizando os pesos desta forma, a soma dos novos pesos será igual a 1. Esta atualização dos pesos é um passo importante do algoritmo. Pois procura fazer com que as instâncias corretamente classificadas tenham seus pesos reduzidos e que as instâncias incorretamente classificadas tenham seus pesos aumentados. Assim, dado que os pesos das instâncias classificadas incorretamente subirão a cada iteração, os próximos classificadores escolhidos serão aqueles que classificarem corretamente as instâncias incorretamente classificadas por seus antecessores. A escolha do classificador em cada iteração e a atualização dos pesos estão interligada.

Demonstraremos uma maneira geral de se encontrar α posteriormente. Contudo, neste ponto, para que possamos definir o modelo final do CRA, é importante saber que α denota a confiança nas predições de cada classificador escolhido. O modelo final do CRA ao fim de T iterações para o nosso problema de classificação binária é dado por:

$$H(X) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(X)\right). \tag{5.3}$$

Logo, dado X, o conjunto de atributos de uma instância a classificar, a decisão final do algoritmo é dada pelo sinal das decisões individuais multiplicado por seus respectivos valores de α. Um valor de α maior implica em dizer que as predições de certo classificador é mais relevante ou mais confiável.

Um esquema geral do Confidence-Rated Adaboost é apresentada na Figura 5.1. O algoritmo tem como entrada o conjunto de dados de treinamento S e um número de iterações T. Os pesos associados às instâncias do treinamento são representados por D. A saída final do algoritmo é H.

O algoritmo inicia todos os pesos em D com um mesmo valor $(1/|S|)$. Em seguida o algoritmo entra em *loop*. Cada iteração chama o *base learner*, encarregado de prover um ou mais classificadores ao algoritmo. Escolhemos h_{j_t} que minimiza Z pela equação (5.1) e o denominamos h_t (respectivamente α_t e Z_t). Após escolhido o classificador da iteração, os pesos das instâncias são atualizados e normalizados (vide Equação (5.2)). Ao final das T iterações, o algoritmo retorna o modelo H composto pela combinação

de todos os α_t e h_t escolhidos.

Entrada: $S = \{(X_1, y_1), \ldots, (X_m, y_m)\}$ onde $X_i \in \mathcal{X}$ e $y_i \in \{-1, +1\}$;
: T, número de iterações.

Inicialize $D_1(i) = 1/|S|$

for $t = 1 \ldots T$ **do**
  $H' = BaseLearner(D_t, S)$
  for $h_j \in H'$ **do**
    $Z_j = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_j y_i h_j(X_i))$
  end
  Encontre:
  $$j_t = \arg\min_{1 \leq j \leq N} Z_j$$
  Seja $h_t = h_{j_t}, \alpha_t = \alpha_{j_t}, Z_t = Z_{j_t}$
  Atualize:
  $$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(X_i))}{Z_t}$$
end

Retorne o modelo final:

$$H(X) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(X)\right).$$

Figura 5.1: Visão geral do Confidence-Rated Adaboost.

5.1.1 Limites do erro de treinamento

Em Schapire & Freund, 2012, demonstra-se que o erro de treinamento do modelo classificador H resultante do algoritmo acima é no máximo o produtório dos erros em cada iteração:

$$\prod_{t=1}^{T} Z_t. \tag{5.4}$$

Isto mostra que minimizando Z_t em cada iteração, melhoramos a acurácia do modelo H. Então esta é a motivação do algoritmo. Para isso, poderíamos sempre escolher α_t e h_t em cada iteração do algoritmo de maneira a minimizar Z_t na Equação (5.1). Na

prática, escolhemos h de forma gulosa a cada iteração do *boosting* e definiremos uma única fórmula para α.

5.2 Uma versão do CRA para uso de regras

Nesta seção descreveremos uma versão do CRA mais adequada para o tipo de classificador usado no BLACk (isto é, regras de decisão). O *base learner* na Figura 5.1 retorna classificadores que podem usar qualquer valor real para indicar o grau de certeza em suas predições. Isto inclui a possibilidade de um classificador retornar o valor 0 como predição de uma dada instância, o que significa abster-se da predição. O classificador pode abster-se de uma predição se este não sabe ou não tem condições de predizer a classe de certa instância. Como veremos, esta é uma habilidade desejável.

As regras de decisão não retornam valores reais, mas valores pertencentes a Y. Porém, uma medida da qualidade da regra (por exemplo, a confiança) pode ser usada como grau de certeza da predição realizada. Contudo, nos interessa mais usar uma versão do CRA cujos classificadores possam retornar um valor de classe pertencente a Y ou se abster e em cujo próprio α indica o grau de certeza de cada regra escolhida. Discutiremos esta versão do CRA nesta seção. Isto implica em mudar o formato do classificador h definido na Seção 5.1.

Assuma um classificador h cujas predições estão restritas aos os valores $\{-1, 0, +1\}$. Em outras palavras, além das classes originais de um problema binário de classificação, o classificador pode predizer o valor 0, isto é, reconhecer que não sabe a classe da instância e abster-se da predição. Como antes, o objetivo do CRA, a cada iteração, é minimizar Z:

$$Z_t = \sum_{i=1}^{m} D(i) \exp(-\alpha_t y_i h_t(X_i)).$$

Para um t fixo e para um classificador h qualquer, definimos W_0, W_-, W_+ como a seguir:

$$W_b = \sum_{i:y_i h(X_i)=b} D(i) \tag{5.5}$$

para $b \in \{-1, 0, +1\}$. Assim W_0 representa a soma dos pesos das instâncias cujo classificador se absteve da predição ($y_i h(X_i) = 0$), W_+ representa a soma dos pesos das instâncias corretamente classificadas ($y_i h(X_i) = 1$), W_- representa a soma dos pesos das instâncias incorretamente classificadas ($y_i h(X_i) = -1$) e $W_0 + W_+ + W_- = 1$.

Podemos agora reescrever Z:

$$\begin{aligned} Z &= \sum_{i=1}^{m} D(i)e^{-\alpha y_i h_t(X_i)} \\ &= \sum_{i:y_i h(X_i)=0} D(i) \; + \sum_{i:y_i h(X_i)=-1} D(i)e^{\alpha} \; + \sum_{i:y_i h(X_i)=1} D(i)e^{-\alpha} \\ &= W_0 + W_- e^{\alpha} + W_+ e^{-\alpha}. \end{aligned} \tag{5.6}$$

Assim dividimos Z de acordo com as instâncias que h se abstém/erra/acerta.

Neste ponto, precisaremos definir o valor de α. Isto é feito a à partir de $Z'(\alpha) = 0$ (Schapire & Singer [1999]). Segue que:

$$\begin{aligned} Z'(\alpha) = \frac{dZ}{d\alpha} &= 0 + (-W_+ e^{-\alpha}) + W_- e^{\alpha} = 0 \qquad\qquad (\div\ (-W_+ e^{-\alpha})) \\ &\Rightarrow 1 + \frac{W_- e^{\alpha}}{-W_+ e^{-\alpha}} = 0 \Rightarrow W_- e^{\alpha} = W_+ e^{-\alpha} \\ &\Rightarrow \frac{e^{\alpha}}{e^{-\alpha}} = \frac{W_+}{W_-} \Rightarrow e^{2\alpha} = \frac{W_+}{W_-} \\ &\Rightarrow \alpha = \frac{\ln(W_+)}{2} - \frac{\ln(W_-)}{2} \\ &\Rightarrow \alpha = \frac{1}{2}\ln\left(\frac{W_+}{W_-}\right). \end{aligned} \tag{5.7}$$

Podemos então reescrever α nas seguintes formas[1]:

$$\alpha = \frac{1}{2}\ln\left(\frac{W_+}{W_-}\right) = \begin{cases} \ln\sqrt{\frac{W_+}{W_-}} \\ \\ -\ln\sqrt{\frac{W_-}{W_+}} \end{cases} \tag{5.8}$$

[1]Vide Anexo B para suporte das propriedades matemáticas utilizadas nesta etapa.

Então, aplicando α em Z temos:

$$\begin{aligned}
Z &= W_0 + W_- e^{\alpha} + W_+ e^{-\alpha} \\
&= W_0 + W_- e^{\ln\sqrt{\frac{W_+}{W_-}}} + W_+ e^{\ln\sqrt{\frac{W_-}{W_+}}} \\
&= W_0 + W_-\sqrt{\frac{W_+}{W_-}} + W_+\sqrt{\frac{W_-}{W_+}} \\
&= W_0 + W_-\frac{\sqrt{W_+}}{\sqrt{W_-}} + W_+\frac{\sqrt{W_-}}{\sqrt{W_+}} \\
&= W_0 + W_-\frac{\sqrt{W_+}}{\sqrt{W_-}}\cdot\frac{\sqrt{W_-}}{\sqrt{W_-}} + W_+\frac{\sqrt{W_-}}{\sqrt{W_+}}\cdot\frac{\sqrt{W_+}}{\sqrt{W_+}} \\
&= W_0 + \cancel{W_-}\frac{\sqrt{W_+W_-}}{\cancel{W_-}} + \cancel{W_+}\frac{\sqrt{W_-W_+}}{\cancel{W_+}} \\
&= W_0 + 2\sqrt{W_+W_-} \\
&= 1 - (W_+ - 2\sqrt{W_-W_+} + W_-) \\
&= 1 - (\sqrt{W_+} - \sqrt{W_-})^2. \qquad (5.9)
\end{aligned}$$

Logo, $Z = 1-(\sqrt{W_+}-\sqrt{W_-})^2$ o que significa que maximizando $|\sqrt{W_+}-\sqrt{W_-}|$, Z será minimizado. Em outras palavras, a cada iteração estamos interessados no classificador (isto é, na regra) que maximiza a diferença entre os pesos das instâncias corretamente e incorretamente classificadas.

É possível que o valor de W_- ou W_+ na Equação (5.7) um valor muito pequeno ou nulo, o que levaria α a ter um valor muito alto ou infinito. Na prática, tal valor pode causar problemas computacionais. Desta forma, Schapire & Singer 1999 sugerem a adição de um valor, a saber $1/2.|S|$, a ambos os termos da fração. Como segue:

$$\widehat{\alpha} = \frac{1}{2}\ln\left(\frac{W_+ + \frac{1}{2.|S|}}{W_- + \frac{1}{2.|S|}}\right), \qquad (5.10)$$

Este novo alfa (chamado de alfa amortizado) age limitando a amplitude de valores que o alfa pode atingir [Schapire & Singer, 1999; Schapire & Freund, 2012].

A Figura 5.2 exibe o algoritmo do CRA. O funcionamento é análogo à do algoritmo apresentado na Figura 5.1. Seja $G = |\sqrt{W_+} - \sqrt{W_-}|$. A cada iteração, procuramos na lista de classificadores o h_{j_t} que maximiza G (pois assim estaremos

minimizando Z). Em seguida calculamos o valor de $\widehat{\alpha}$ a partir da Equação 5.10. Então usamos $\widehat{\alpha}$ e h para atualizar os valores dos pesos das instâncias. Por fim, os pesos são normalizados.

Entrada: $S = \{(X_1, y_1), \ldots, (X_m, y_m)\}$ onde $X_i \in \mathcal{X}$ e $y_i \in \{-1, +1\}$;
: T, número de iterações;
: Lista de classificadores $h_1, \ldots, h_N$

Inicialize $D(i) = 1/|S|$

for $t = 1 \ldots T$ **do**

  for $j = 1 \ldots N$ **do**

    $$W_b^j = \sum_{i: y_i h_j(X_i) = b} D(i), \; \forall b \in \{-1, +1\}$$

    $$G_j = \left| \sqrt{W_+^j} - \sqrt{W_-^j} \right|$$

  end

  Encontre:

  $$j_t = \arg\max_{1 \leq j \leq N} G_j$$

  Calcule:

  $$\widehat{\alpha}_t = \frac{1}{2} \ln \left(\frac{W_+^{j_t} + \frac{1}{2.|S|}}{W_-^{j_t} + \frac{1}{2.|S|}} \right)$$

  Atualize:

  $$D_t(i) \leftarrow D_t(i) e^{-\widehat{\alpha}_t y_i h_{j_t}(X_i)}$$

  Normalize:

  $$D_{t+1}(i) \leftarrow D_t(i)/Z_t, \text{ onde } Z_t = \sum_{i=1}^{|S|} D_t(i)$$

end

Retorne o modelo final:

$$H(X) = \text{sign} \left(\sum_{t=1}^{T} \widehat{\alpha}_t h_{j_t}(X) \right). \tag{5.11}$$

Figura 5.2: Confidence-Rated Adaboost para regras de decisão.

No momento da atualização de pesos no algoritmo acima, qualquer instância em que $h_{j_t}(X_i) = 0$ não terá seu peso alterado, pois teríamos que $D(i) \leftarrow D(i).e^0 \Rightarrow D(i) \leftarrow D(i)$. Ou seja, podemos atualizar os pesos das instâncias iterando apenas sobre as instâncias cobertas pelo classificador h_{j_t}.

5.3 SLIPPER

O SLIPPER foi o primeiro algoritmo de geração de regras a usar *boosting*. O algoritmo usa o Confidence-Rated Adaboost para guiar o processo de geração de regra. Cada classificador do *boosting* é representado por uma regra no SLIPPER, devido a isto usaremos a letra h para representar um regra.

As regras do SLIPPER são compostas por condições. Cada condição possui uma das seguintes formas: $a_n = v$, onde a_n é um atributo nominal e v é um valor nominal que ocorre nos dados para este atributo; ou $a_c \leq \theta$ ou $a_c \geq \theta$, onde a_c é um atributo contínuo e θ é um valor qualquer que ocorre para a_c nos dados para este atributo. Uma regra cobre uma instância se a instância retorna verdadeiro para todas as condições da regra. Dizemos que se uma regra h cobre uma instância $x_i = (X_i, y_i)$ então $x_i \in h$ ($x_i \notin h$, caso contrário)[2,3]. A partir deste ponto procuraremos usaremos a palavra "classificador" em vez de regra.

Para o SLIPPER um classificador h classifica uma instância como segue:

$$h(x_i) = \begin{cases} \alpha, & \text{se } x_i \in h \\ 0, & \text{se } x_i \notin h \end{cases} . \tag{5.12}$$

Dito isto, temos que para um classificador h, seu respectivo Z é:

$$\begin{aligned} Z &= \sum_{x_i \notin h} D(i) + \sum_{x_i \in h} D(i) \exp(-y_i h(X_i)) \\ &= \sum_{x_i \notin h} D(i) + \sum_{x_i \in h : y_i = 1} D(i) e^{-\alpha} + \sum_{x_i \in h : y_i = -1} D(i) e^{\alpha} \\ &= W_0 + W_+ e^{-\alpha} + W_- e^{\alpha}. \end{aligned}$$

onde:

$$W_b = \sum_{x_i \in h : y_i = b} D(i)$$

e $b \in \{-1, 0, +1\}$. A equação acima é, de fato, a mesma Equação (5.6), mas os valores de W_+ e W_- representam as somas dos pesos das instâncias cobertas pelo classificador h das classes positiva e negativa.

[2]Estamos aqui relaxando esta notação para simplificar e facilitar o entendimento do restante da explicação.

[3]Usamos aqui a mesma notação de instância binária que definimos anteriormente, mas o SLIPPER aceita instâncias com valores nominais e contínuos

Perceba que acima não utilizamos o valor de α na primeira equação de Z. Esta é uma estratégia chamada de *folding* por Shapire (Schapire & Freund [2012]). O que de fato estamos fazendo é usar o valor que calcularemos para o α como o valor que indica a certeza (e a classe) resultante da classificação de uma instância pela regra (isto é, pelo classificador). Se fizermos com que o classificador h retorne apenas 1 ou 0 em (5.12), a notação estaria idêntica à definida na seção passada.

Neste ponto precisaríamos encontrar uma fórmula para α que minimiza Z, mas esta fórmula já foi descrita em (5.7). Logo o valor de Z para o SLIPPER é o mesmo da Equação (5.9), isto é:

$$Z = 1 - (\sqrt{W_+} - \sqrt{W_-})^2.$$

No algoritmo do Confidence-Rated Adaboot (Figura 5.2) procurávamos a cada iteração maximizar $G = |\sqrt{W_+} - \sqrt{W_-}|$. O SLIPPER, porém, se restringe a encontrar regras positivas apenas. Portanto seja $\widetilde{G} = \sqrt{W_+} - \sqrt{W_-}$ o valor a ser maximizado nas iterações do *boosting* no SLIPPER. Logo queremos que o peso das instâncias de classe positiva cobertas por h seja maior que o peso das instâncias de classe negativa cobertas por este. Podemos agora explicar como o SLIPPER gera as suas regras utilizando o algoritmo de *boosting*.

5.3.1 Geração de regras

O *base learner* utilizado é responsável por receber um conjunto de instâncias e um conjunto de pesos associados a estas e construir uma única regra que será o nosso classificador da iteração t.

O processo do *base learner* inicia dividindo o conjunto de instâncias em dois conjuntos disjuntos: `GrowSet` e `PruneSet`. Os tamanhos dos conjuntos são de 2/3 e 1/3, respectivamente. O *base learner* então invoca uma rotina chamada de `GrowRule`. Esta rotina inicia com uma regra vazia (sem condições) e segue um processo iterativo. A cada iteração do `GrowRule`, escolhe-se uma condição que, quando adicionada a regra sendo montada, maximiza $\widetilde{G}$. Por sua vez, o calculo dos valores de W_+ e W_- é feito somente sobre as instâncias do `GrowSet`. A rotina do `GrowRule` é encerrada quando o classificador deixa de cobrir instâncias negativas ou nenhuma condição consegue melhorar o valor de $\widetilde{G}$.

Frequentemente, a regra resultante do `GrowRule` é muito específica, isto é, cobre uma pequena porção dos dados. Regras muito específicas não generalizam bem as informações contidas nos dados e tendem ao superaprendizado. Por tanto, ao fim da

rotina de `GrowRule` a rotina de `PruneRule` é iniciada com a função de remover condições da regra resultante do `GrowRule`. A rotina de `PruneRule` procura remover condições na ordem inversa em que elas foram adicionadas. Cada sequência de remoção de condições gera uma regra nova. Procura-se a regra que minimiza:

$$(1 - V_+ - V_-) + V_+ e^{-\alpha} + V_- e^{\alpha}. \tag{5.13}$$

onde V_+, V_-, e são calculados como W_+, W_- (respectivamente) sobre as instâncias do `PruneSet` e α é calculado usando V_+ e V_-.

Ao fim da rotina de `PruneRule`, a regra resultante é comparada à regra vazia; e o *base learner* retorna, dentre estas duas regras, aquela que minimiza Z na Equação (5.9) (usando-se todas as instâncias dos dados). A regra vazia cobre todos as instâncias do dado e ela é necessária porque o algoritmo gera apenas regras positivas. Sem a regra vazia não seria possível classificar qualquer instância como negativa.

A regra retornada pelo *base learner* se torna o classificador h_t cujo valor de seu α é calculado sobre todas as instâncias do treinamento.

5.3.2 Algoritmo

O algoritmo do SLIPPER é apresentado na Figura 5.3. Na notação usamos o valor amortizado de α, isto é $\widehat{\alpha}$ pelos mesmos motivos discutidos na seção anterior.

O algoritmo faz chamada do *base learner* que é responsável por retornar um classificador. Este classificador a representação de uma regra de decisão construída como acabamos de discutimos na Seção 5.3.1. Então, o classificador retornada pelo *base learner* pode ser a regra gerada pelo processo de geracão de regras ou a regra vazia. Independente disto, demos que o classificador retornado é o classificador h_t escolhido para aquela iteração. Calculamos $\widehat{\alpha}_t$ como fizemos no Confidence-Rated Adaboost. Atualizamos os pesos de todas as instâncias cobertas por h_t (afinal a saída de h_t para as instâncias não cobertas é 0, o que resultaria nenhuma mudança no valor do peso da instância). Posteriormente as instâncias do treinamento têm seus pesos normalizados para que a soma dos pesos seja 1. Ao final do algoritmo retorna-se H como sendo o modelo classificador resultante do aprendizado.

Assim o SLIPPER consegue gerar um conjunto de regras, tendo como propósito, a cada iteração do *boosting* montar uma regra que minimiza Z.

Entrada: $S = \{(X_1, y_1), \ldots, (X_m, y_m)\}$ onde $X_i \in \mathcal{X}$ e $y_i \in \{-1, +1\}$;
: T, número de iterações;
Inicialize $D(i) = 1/|S|$

for $t = 1 \ldots T$ **do**
Chame o *base learner* passando D_t:

1. Divida os dados em `GrowSet` e `PruneSet`.
2. `GrowRule`: Inicie com uma regra vazia e gulosamente adicione condições à regra a fim de maximizar $\widetilde{G}$.
3. `PruneRule`: Inicie com a regra final do `GrowRule` e remova condições da regra a fim de minimizar Equação (5.13).
4. Retorne a regra final do `PruneRule` ou a regra vazia. Escolha a que minimiza a Equação (5.9).

Seja h_t o classificador retornado pelo *base learner* e $\widehat{\alpha}_t$ seu respectivo alfa amortizado.

Atualize:

$$D_t(i) \leftarrow D_t(i)e^{-y_i\widehat{\alpha}_t}, \forall x_i \in h_t$$

Normalize:

$$D_{t+1}(i) \leftarrow D_t(i)/Z_t, \text{ onde } Z_t = \sum_{i=1}^{|S|} D(i)$$

end
Retorne o modelo final:

$$H(x) = \text{sign}(\sum_{t:x\in h_t} h(x)). \tag{5.14}$$

Figura 5.3: SLIPPER: *Simple Learner with Iterative Pruning to Produce Error Reduction.*

5.3.3 Validação-Cruzada-Interna

O SLIPPER só possui um parâmetro a ser determinado pelo usuário, a saber o número de iterações do *boosting*. Os criadores do algoritmo utilizam uma validação-cruzada-interna de 5-dobras para determinar o valor de T. Os dados de treinamento são fatiados em 5 conjuntos disjuntos de instâncias e o algoritmo da Figura 5.3 é executado 5 vezes para um T_{max} predefinido de iterações (este valor é definido pelo usuário). Cada execução do algoritmo utiliza 4 fatias para treinar e é testado na fatia restante (chamaremos isto de dobra). O número de iterações T^* cuja média das acurácias em todas as dobras for a maior é o valor determinado de T para treinar no conjunto completo de dados de treinamento (empates são determinados pelo menor valor de T^*).

Capítulo 6

Boosted Lazy Associative Classifier

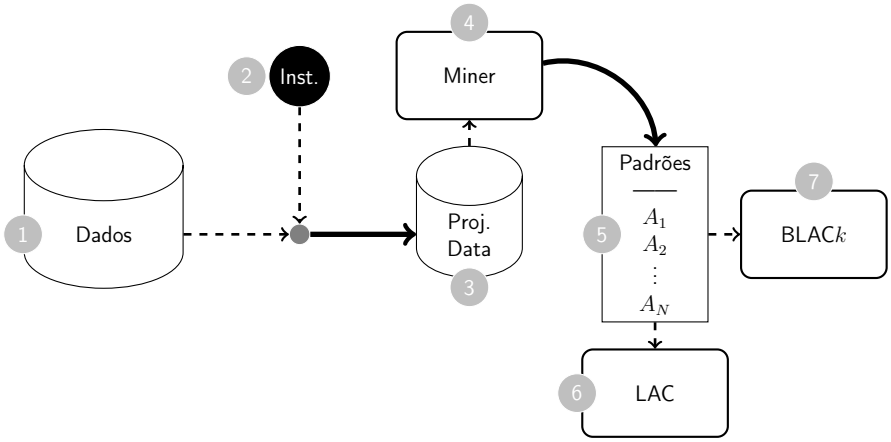
Nesta seção discutiremos sobre o BLACk (*Boosted Lazy Associative Classifier of k rules*). O BLACk constrói um modelo composto por um conjunto de regras de decisão e um conjunto de valores reais associados a estas. O algoritmo possui um processo semelhante ao do LAC, mas utiliza a ideia de minimização de Z presente no *boosting* para avaliar um conjunto de padrões frequentes $F = \{A_1 \dots A_N\}$, formar regras que serão adicionadas ao seu modelo final e calcular valores que representam a certeza da predição de cada regra.

A Figura 6.1 nos dá uma ideia geral do funcionamento do BLACk e do LAC e mostra em que ponto as duas abordagens se diferem. Ambos algoritmos recebem os dados de treinamento (1) e uma instância de classe desconhecida (2). Os dados são projetados (3) e então o *Miner* (4) se encarrega de buscar os padrões frequentes (5). A partir deste ponto, o LAC (6) e o BLACk (7) usam estratégias diferentes para montarem seus conjuntos de regras.

6.1 Primeiros conceitos

Antes de iniciarmos a explicação do algoritmo, precisamos conectar os elementos do BLACk que também estão presentes no LAC com o *boosting*:

Conjunto de dados de treinamento O conjunto de dados usado pelo BLACk é proveniente do processo de projeção. Como já mencionado, a projeção dos dados é a primeira fase do LAC. O objetivo desta fase é filtrar os dados de treinamento, isto é, reduzir os atributos que descrevem as instâncias. A projeção foi definida em (4.1) com uma nomenclatura específica. Contudo, para que continuemos utilizando a nomenclatura definida na seção de *boosting*, usaremos a letra $S = \{(X_1, y_1), \dots, (X_m, y_m)\}$



1. Dados de treinamento; 2. Instância de classe desconhecida; 3. Dados projetados; 4. Mineração de padrões frequentes; 5. Lista de padrões; 6. LAC; 7. BLACk (Confidence Adaboost)

Figura 6.1: Relação entre o BLACk e o LAC
Fonte: Autoria Própria.

para representar o conjunto de treinamento do BLACk onde as instâncias em S são as mesmas instâncias da projeção.

Regras de decisão As regras do BLACk assumirão a forma $h := A \to \alpha$, onde $A \in F$ é um padrão frequente e α será encontrado por meio do processo do *boosting*. A classe da regra é dada por $\text{sign}(\alpha)$ (assuma $Y = \{-1, +1\}$). Novamente o sentido de "regra" e de "classificador" se sobrepõe, portanto usaremos apenas a palavra "classificador" a partir deste ponto.

Uma classificador h do BLACk classifica uma instância binária $x = (X, y)$ como segue:

$$h(X) = \begin{cases} \alpha, & \text{se } A \subseteq X \\ 0, & \text{se } A \not\subset X. \end{cases} \tag{6.1}$$

Este é o mesmo tipo de classificador do SLIPPER.

Boosting Usaremos toda a notação definida no Confidence-Rated Adaboots. Assim, seja D o conjunto de valores reais positivos, chamados de pesos, associados às instâncias do treinamento, onde $D(i)$ é o peso associado à instância $x_i = (X_i, y_i)$. Dito isto, temos

que para h seu respectivo Z é:

$$\begin{aligned} Z &= \sum D(i)\exp(-y_i h(X_i)) \\ &= \sum_{A \not\subset X_i} D(i) + \sum_{A \subseteq X_i} D(i)\exp(-y_i h(X_i)) \\ &= \sum_{A \not\subset X_i} D(i) + \sum_{A \subseteq X_i : y_i = 1} D(i)e^{-\alpha} + \sum_{A \subseteq X_i : y_i = -1} D(i)e^{\alpha} \\ &= W_0 + W_+ e^{-\alpha} + W_- e^{\alpha}. \end{aligned} \tag{6.2}$$

Em palavras, Z pode ser subdividido na soma dos pesos das instâncias não cobertas, cobertas e de classe igual a $+1$ multiplicado por $e^{-\alpha}$, e cobertas e de classe igual a -1 multiplicado por e^{α}. Neste ponto, precisaríamos encontrar α tal que minimiza $Z'(\alpha) = 0$. Segue-se que:

$$\alpha = \frac{1}{2}\ln\left(\frac{W_+}{W_-}\right),$$

como já havíamos encontrado anteriormente. Aplicando α em Z temos:

$$Z = 1 - (\sqrt{W_+} - \sqrt{W_-})^2.$$

Logo buscaremos maximizar $G = |\sqrt{W_+} - \sqrt{W_-}|$ em cada iteração do *boosting* a fim de minimizar Z.

Para evitar problemas computacionais no cálculo de α definiremos um α amortizado como fizemos antes:

$$\widehat{\alpha} = \frac{1}{2}\ln\left(\frac{W_+ + \frac{1}{2.|S|}}{W_- + \frac{1}{2.|S|}}\right).$$

Lembrando que agora $h := A \to \widehat{\alpha}$ e que as instâncias em S são as instâncias resultantes do processo de projeção dos dados.

Como é possível notar todas estas equações já haviam sido definidas anteriormente, mas foram repetidas aqui para facilitar a leitura e as discussões subsequentes. Por fim perceba que apenas com o padrão A já é possível determinar os valores de W_+ e W_-. Isto significa que precisamos apenas iterar sobre os padrões disponíveis, onde o padrão que maximizar G será o escolhido para formar o classificador h e fará parte do modelo final do nosso algoritmo. Isto ficará mais claro quando visualizarmos o algoritmo do BLACk.

6.2 Algoritmo

Assim como o LAC, o BLACk é um algoritmo *lazy*. No processo completo do algoritmo, sempre que deseja-se classificar determinada instância realizamos o processo de projeção dos dados e de busca por padrões fechados antes de gerarmos nossas regras e por fim o nosso modelo final. O modelo final é então usado para classificar a instância que deu início ao processo. O algoritmo do BLACk pode ser visto na Figura 6.2. Por conveniência, partimos do momento em que a projeção e a mineração dos padrões fechados já aconteceu. Isto porque podemos facilmente optarmos por não fazermos a projeção e porque não precisamos especificar qual minerador de padrões o usuário deve usar.

O algoritmo recebe o conjunto de treinamento S (que é a própria projeção), o conjunto de padrões fechados F e o número máximo de iterações do algoritmo T. Este último parâmetro é definido pelo usuário. Os pesos em D são inicializados com um mesmo valor e de modo que a soma destes seja igual a 1. Então o algoritmo entra no seu laço principal. A cada iteração buscaremos um padrão para formarmos uma regra (classificador).

Logo no início de cada iteração temos os valores de W_+ e W_- calculados para cada padrão A_j. Assim, para o padrão A_j temos W_+^j e W_-^j. Com estes dois valores calculamos $G_j = \left|\sqrt{W_+^j} - \sqrt{W_-^j}\right|$. Após estes cálculos iniciais escolhemos j tal que G_j seja o maior valor calculado para a iteração t. Definimos assim j_t como sendo o j escolhido na iteração t. Em outras palavras A_{j_t} é o padrão escolhido para compor h_t que fará parte do modelo final do algoritmo.

Com os valores de $W_+^{j_t}$ e $W_-^{j_t}$ calculamos o valore de $\widehat{\alpha}$ e finalmente construímos $h_t := A_{j_t} \rightarrow \widehat{\alpha}$. Atualizamos os pesos das instâncias do treinamento a partir das predições de h_t e posteriormente normalizamos os pesos para que a soma destes seja igual a 1.

O modelo final do BLACk é dado pela Equação (6.3). Dada o padrão X de uma instância não classificada a classe predita é dada pelo sinal da soma das predições dos classificadores escolhidos nas iterações do algoritmos. Contudo, dado o modo de operação *lazy* do algoritmo, este modelo só será usado para classificar a instância que deu início ao processo do algoritmo. Isto é, a instância usada para gerar a projeção e de onde partiu a mineração de padrões. Logo, todas os classificadores escolhidos cobrem esta instância em questão. Assim a predição do algoritmo seria simplesmente o sinal da soma dos $\widehat{\alpha}_t$.

Entrada: $S = \{(X_1, y_1), \ldots, (X_m, y_m)\}$, as instâncias da projeção;
: $F = \{A_1, \ldots, A_N\}$, lista de padrões frequentes.
: T, número de iterações;

Inicialize $D_1(i) = 1/|S|$

for $t = 1 \ldots T$ **do**

for $j = 1 \ldots N$ **do**

$$W_+^j = \sum_{A_j \subseteq X_i : y_i = 1} D_t(i)$$

$$W_-^j = \sum_{A_j \subseteq X_i : y_i = -1} D_t(i)$$

$$G_j = \left| \sqrt{W_+^j} - \sqrt{W_-^j} \right|$$

end

Encontre:

$$j_t = \arg\max_{1 \leq j \leq N} G_j$$

Calcule:

$$\widehat{\alpha}_t = \frac{1}{2} \ln \left(\frac{W_+^{j_t} + \frac{1}{2.|S|}}{W_-^{j_t} + \frac{1}{2.|S|}} \right)$$

Construa:

$$h_t := A_{j_t} \rightarrow \widehat{\alpha}_t$$

Atualize:

$$D_t(i) \leftarrow D_t(i) e^{-y_i h_t(X_i)}$$

Normalize:

$$D_{t+1}(i) \leftarrow D_t(i)/Z_t, \text{ onde } Z_t = \sum_{i=1}^{|S|} D_t(i)$$

end

Retorne o modelo final:

$$H(X) = \text{sign} \left(\sum_{t=1}^{T} h_t(X) \right) \tag{6.3}$$

Figura 6.2: Boosted Lazy Associative Classifier of k rules.

6.3 Exemplo

Nesta seção apresentamos um pequeno exemplo de execução do nosso algoritmo para complementar o entendimento do funcionamento desde por parte do leitor. Todos os resultados intermediários estão descritos na Tabela 6.1.

O nosso exemplo utiliza instâncias projetadas apresentadas na Tabela 4.1 como

conjunto de treinamento, e os padrões apresentados na Tabela 4.2. As instâncias foram renumeradas, de modo que a instância 2 na Tabela 4.2 agora é a instância 1 na Tabela 6.1, a instância 3 na Tabela 4.2 agora é a instância 2 na Tabela 6.1 e assim por diante até a instância 10 da Tabela 4.2 que agora é a instância 5 da Tabela 6.1. A classe de cada instância é exibida ao lado de seu índice.

A primeira coluna da tabela informa a iteração do algoritmos de *boosting*. A cada iteração exibe-se os valores de D_t e ao lado o valor de G calculado para cada padrão. Na ultima coluna exibimos a o classificador escolhido h_t (isto é a regra) que fará parte do modelo final. Ainda na iteração t exibimos os pesos atualizados a partir do classificador h_t logo abaixo dos pesos de D_t e exibimos o valor de Z_t que usado para normalizar os pesos de D_t. Por sua vez os pesos normalizados são exibidos na linha $t = t + 1$.

Começando pela linha $t = 1$, exibimos os pesos de iniciais das 5 instâncias do conjunto de treinamento (D_1) todas inicializada para $1/|S| = 0,2$. Em seguida calculamos para o padrão $j = 1$ ($A_1 = \{a_2\}$) seus valores de $W_+^1 = 0$ e $W_-^1 = D_1(2) + D_1(4) = 0,4$ (o padrão A_1 cobre apenas as instâncias 2 e 4 que são de classe negativa). Calculamos $G_1 = |\sqrt{0} - \sqrt{0,4}| = 0,632$. Este processo é repetido para $j = 2, \ldots, 4$. Os valores de cada G da iteração $t = 1$ são exibidos na tabela logo abaixo dos padrões. Escolhemos aleatoriamente G_1 como sendo o máximo G encontrado na iteração (**na tabela o G escolhido em cada iteração aparece em negrito**) e formamos a regra $h_1 =: A_1 \to \widehat{\alpha}_1$, onde $A_1 = \{a_2\}$ e $\widehat{\alpha}_1 = -0,805$.

Utilizamos então h_1 para atualizar os pesos das instâncias. Modificamos assim apenas os pesos das instâncias 2 e 4 (**na tabela as instâncias modificadas têm seus pesos originais sublinhados**). A soma dos pesos após a atualização é dado por $Z_1 \approx 0,779$. Normalizamos então todos os pesos para obtermos D_2.

Após estas 5 iterações temos um total de 5 regras (4 negativas e 1 positiva) e podemos determinar a classificação da instância por parte do algoritmo por meio da Equação (6.3). Lembrando que todas as regras geradas cobrem a instância que originou o processo do algoritmo, a classe predita pelo algoritmo é dado pelo sinal da soma de todos os $\widehat{\alpha}_t$. Assim:

$$\begin{aligned} H(X) &= \text{sign}(\widehat{\alpha}_1 + \widehat{\alpha}_2 + \widehat{\alpha}_3 + \widehat{\alpha}_4 + \widehat{\alpha}_5) \\ &= \text{sign}(-0,805 - 0,775 - 0,581 - 0,548 + 0,737) \\ &= \text{sign}(-1,972). \end{aligned}$$

Logo, temos que a classe da instância que deu início ao processo do algoritmo é negativa.

t		Instâncias					Padrões (G)				Regra
		1 (+)	2 (−)	3 (−)	4 (−)	5 (+)	$\{a_2\}$	$\{a_5\}$	$\{a_{10}\}$	$\{\emptyset\}$	
1	D_1	0,200	0,200	0,200	0,200	0,200	**0,632**	0,632	0,185	0,142	$(\{a_2\} \to \widehat{\alpha}_1)$
		0,200	0,089	0,200	0,089	0,200		$Z_1 \approx 0,779$			$\alpha_1 \approx -0,805$
2	D_2	0,257	0,115	0,257	0,115	0,257	0,479	**0,610**	0,378	0,019	$(\{a_5\} \to \widehat{\alpha}_2)$
		0,257	0,053	0,118	0,115	0,257		$Z_2 \approx 0,800$			$\alpha_2 \approx -0,775$
3	D_3	0,330	0,068	0,152	0,147	0,330	0,464	**0,469**	0,428	0,206	$(\{a_5\} \to \widehat{\alpha}_3)$
		0,330	0,038	0,085	0,147	0,330		$Z_3 \approx 0,930$			$\alpha_3 \approx -0,581$
4	D_4	0,355	0,041	0,091	0,159	0,355	**0,447**	0,364	0,444	0,303	$(\{a_2\} \to \widehat{\alpha}_4)$
		0,355	0,024	0,091	0,092	0,355		$Z_4 \approx 0,916$			$\alpha_4 \approx -0,548$
5	D_5	0,387	0,026	0,100	0,100	0,387	0,355	0,354	**0,564**	0,405	$(\{a_{10}\} \to \widehat{\alpha}_5)$
		0,387	0,012	0,100	0,048	0,387		$Z_5 \approx 0,934$			$\alpha_5 \approx 0,737$

Tabela 6.1: Exemplo de execução do BLACk.

6.4 Complexidade computacional

A complexidade do BLACk depende diretamente do número de iterações escolhidas, do número de padrões disponíveis e do número de instâncias no conjunto de treinamento (no caso, a projeção). Seja T o número de iterações do BLACk, $m = |S|$ o tamanho do conjunto de treinamento e N o número de padrões frequentes encontrados, onde N pode ser exponencial no número de atributos da instância a classificar.

Subdividiremos o algoritmo da Figura 6.2 em 3 partes principais: Calcular os valores de G para cada um dos padrões (laço mais interno do algoritmo); Encontrar o G máximo; e atualizar e normalizar os pesos das instâncias. Na primeira parte do algoritmo verificamos para cada padrão quais instância estes cobrem a fim de calcularmos seus respectivos valores de G. Isto nos dá uma complexidade de de pior caso de $O(Nm)$. Isto porque no pior caso os padrões cobrem todas as instâncias do treinamento.

Na segunda parte do algoritmo buscamos o índico do máximo valor de G encontrado dado os pesos da iteração atual. A busca é feita sobre um conjunto não ordenado de valores, portanto teremos complexidade $O(N)$ para determinar G_j máximo. Na terceira parte do algoritmo precisamos atualizar e normalizar os pesos. Ambas tarefas tem complexidade $O(m)$ pois iteramos sobre todas as instâncias do treinamento. Logo, a complexidade em função da instância a classificar do processo inteiro da Figura (6.2) é $O(TNm)$, onde N pode ser exponencial no número de atributos da instância a classificar.

6.5 BLACk multiclasse

Nesta seção descreveremos a versão multiclasse do BLACk. Removendo o *folding* aplicado ao nosso classificador, redefinimos h como segue:

$$h(X) = \begin{cases} c, & \text{se } A \subseteq X \\ 0, & \text{se } A \not\subset X. \end{cases} \tag{6.4}$$

Agora podemos definir Z em termos de cobertura e classe das instâncias. Assim temos que Z:

$$Z = \sum_{A \not\subseteq X_i} D(i) + \sum_{A \subseteq X_i : y_i = 1} D(i)e^{-\alpha} + \sum_{A \subseteq X_i : y_i = 2} D(i)e^{\alpha} + \cdots + \sum_{A \subseteq X_i : y_i = n} D(i)e^{\alpha}$$

Note que quando $y_i = 1$ mantemos o valor de α negativo, isto porque arbitrariamente queremos gerar uma regra que aponte para a classe 1. Podemos simplificar Z como segue:

$$Z = W_0 + W_1 e^{-\alpha} + W_2 e^{\alpha} + \cdots + W_n e^{\alpha}$$

Agora usando a primeira derivada de Z descobriríamos um valor para α bem similar ao que já havíamos encontrado:

$$\begin{aligned}
Z'(\alpha) = \frac{dZ}{d\alpha} &= 0 + (-W_1 e^{-\alpha}) + (W_2 + \cdots + W_n)e^{\alpha} = 0 && (\div\ (-W_1 e^{-\alpha})) \\
&\Rightarrow 1 + \frac{(W_2 + \cdots + W_n)e^{\alpha}}{-W_1 e^{-\alpha}} = 0 \Rightarrow (W_2 + \cdots + W_n)e^{\alpha} = W_1 e^{-\alpha} \\
&\Rightarrow \frac{e^{\alpha}}{e^{-\alpha}} = \frac{W_1}{W_2 + \cdots + W_n} = e^{2\alpha} = \frac{W_1}{W_2 + \cdots + W_n} \\
&\Rightarrow \alpha = \frac{\ln(W_1)}{2} + \frac{\ln(W_2 + \cdots + W_n)}{2} \\
&\Rightarrow \alpha = \frac{1}{2} \ln \left(\frac{W_1}{W_2 + \cdots + W_n} \right).
\end{aligned}$$

Contudo existe uma falha nesta abordagem. Anteriormente sempre que o valor de α fosse positivo o classificador estaria apontando para a classe positiva e sempre que o valor de α fosse negativo o classificador estaria apontando para a classe negativa. Enquanto, usando o α descrito acima temos apenas a ideia que se α for positivo então a regra aponta para $c = 1$, mas se α for negativo podemos apenas afirmar que a classe **não aponta** para $c = 1$. Assim, precisaríamos sempre que $W_1 > W_2 + \ldots W_n$, onde a

classe 1 foi arbitrariamente escolhida.

Logo uma abordagem multiclasse mais apropriada seria utilizarmos o mesmo algoritmo da Figura 6.2 substituindo apenas W_+ e W_- pelo maior e segundo maior valor em $\{W_1, \ldots, W_n\}$, respectivamente, calculando o valor de α da mesma maneira que anteriormente e fazendo as devidas adaptações para o modo multiclasse. Em outras palavras escolheremos c e d tal que W_c é o maior valor em $\{W_1, \ldots, W_n\}$ e W_d é o segundo maior valor de maneira a seguir uma estratégia OvA (*one versus all*). Isto é, dentre as classes disponíveis escolheremos c como classe da regra e d como a classe que mais se confunde com d ou que a regra tem dificuldade em classificar.

A Figura 6.3 exibe as adaptações realizadas no BLACk para torná-lo multiclasse. As principais modificações aconteceram no laço interno do algoritmo onde agora calculamos os valores W de cada padrão para cada classe e escolhemos c_j e d_j tal que W_{c_j} e W_{d_j} sejam o maior e o segundo maior valor dentre os valores W calculados para o padrão A_j. Como $W_{c_j} >= W_{d_j}$ pudemos remover o módulo da fórmula de G.

Outra modificação significante aconteceu na parte da atualização dos pesos. Ainda que estamos tomando em consideração apenas W_c e W_d para calcularmos o α das regras, os erros cometidos por esta se aplicam a todo o conjunto de treinamento. Logo, se a regra cobre a instância e a classe real da instância é a mesma classe da regra então reduzimos o peso da instância:

$$D_t(i) \leftarrow D_t(i)e^{-\widehat{\alpha}_t},$$

caso o classificador cubra a instância mas a classe real da instância não seja a mesma da classe da regra então incrementamos o seu peso:

$$D_t(i) \leftarrow D_t(i)e^{\widehat{\alpha}_t}.$$

Por fim, modificamos o modelo retornado pelo algoritmo para refletir o uso de mais que duas classes como segue:

$$H(X) = \underset{c \in Y}{\arg\max} \sum_{t=1}^{T} \widehat{\alpha}_t \ 1\{h_t(X) = c\},$$

onde $1\{\cdot\}$ é uma função que retorna 1 se a expressão passada for verdadeira e 0 caso contrário. Assim, dado o padrão X uma instância a classificar retornamos a classe de maior soma de $\widehat{\alpha}$.

Entrada: $S = \{(X_1, y_1), \ldots, (X_m, y_m)\}$, as instâncias da projeção;
: $F = \{A_1, \ldots, A_N\}$, lista de padrões frequentes.
: T, número de iterações;

Inicialize $D_1(i) = 1/|S|$

for $t = 1 \ldots T$ **do**

for $j = 1 \ldots N$ **do**

Calcule:

$$W_c^j = \sum_{A_j \subseteq X_i : y_i = c} D_t(i), \forall c \in Y$$

Seja c_j tal que $W_{c_j}^j$ é o maior valor em $\{W_1, \ldots W_n\}$

Seja d_j tal que $W_{d_j}^j$ é o segundo maior valor em $\{W_1, \ldots W_n\}$

Calcule:

$$G_j = \sqrt{W_{c_j}^j} - \sqrt{W_{d_j}^j}$$

end

Encontre:

$$j_t = \arg\max_{1 \leq j \leq N} G_j$$

Calcule:

$$\widehat{\alpha}_t = \frac{1}{2} \ln \left(\frac{W_{c_{j_t}}^{j_t} + \frac{1}{2.|S|}}{W_{d_{j_t}}^{j_t} + \frac{1}{2.|S|}} \right)$$

Construa:

$$h_t := A_{j_t} \rightarrow c_{j_t}$$

Atualize:

- $D_t(i) \leftarrow D_t(i) e^{-\widehat{\alpha}_t}$, se $A_{j_t} \subseteq X_i \wedge y_i = c_{j_t}$
- $D_t(i) \leftarrow D_t(i) e^{\widehat{\alpha}_t}$, se $A_{j_t} \subseteq X_i \wedge y_i \neq c_{j_t}$

Normalize:

$D_{t+1}(i) \leftarrow D_t(i)/Z_t$, onde $Z_t = \sum_{i=1}^{|S|} D_t(i)$

end

Retorne o modelo final:

$$H(X) = \arg\max_{c \in Y} \sum_{t=1}^{T} \widehat{\alpha}_t \, 1\{h_t(X) = c\} \tag{6.5}$$

Figura 6.3: BLACk multiclasse.

6.6 O uso de padrões fechados

Em qualquer iteração, dois padrões que cobrem um mesmo subconjunto de instâncias resultam nos mesmos valores de W_+, W_- e, consequentemente G, ainda que estes padrões possuam atributos binários diferentes. Isto implica em dizer que ambos os padrões levam à mesma atualização de pesos se escolhidos para montar a regra iteração. Logo todos os padrões que cobrem um mesmo subconjunto das instâncias pode ser substituídos por um só deles sem que haja alteração na classificação.

Dado um conjunto de instâncias, a interseção do conjunto de atributos que descrevem estas instâncias é denominado de padrão fechado. Uma padrão A é dito fechado se A é frequente e não existe nenhum super-padrão $U \supset A$ que cobre as mesmas instâncias que A [Pasquier et al., 1999].

Existem algoritmos que mineram somente os padrões fechados. O uso de tais algoritmo pelo BLACk pode levar a uma diminuição da complexidade computacional. Enquanto o número total de padrões é exponencial no número de atributos da instância a classificar, o número de padrões fechados é exponencial no min(atributos na projeção, atributos da instância a classificar). Assim teremos que a complexidade do nosso algoritmo usando padrões fechados é dado por $O(Tm.2^{\min(\text{atributos na projeção, atributos da instância a classificar})})$. Logo optaremos por trabalhar com padrões fechados em nossos experimentos.

6.7 Validação-Cruzada-Interna

Assim como no SLIPPER, esperamos determinar automaticamente por meio de validação-cruzada-interna o número iterações T. Obviamente usar uma validação interna traz mais custo ao processo de aprendizado, especialmente porque este mesmo processo é repetido para cada instância a ser classificada, contudo, para aplicações onde tempo não é uma questão a técnica pode ser usada na automatização do algoritmo. Na Seção 7 exibiremos os resultados da validação interna.

6.8 Intâncias de fora da projeção

Um leitor mais curioso pode se perguntar se existe algum efeito positivo ou negativo no uso das instâncias de fora da projeção para o LAC ou para o BLACk. As instâncias de fora da projeção são aquelas que têm intersecção vazia com a instância a classificar. De fato, esta é uma discussão válida, especialmente para o BLACk.

O BLACk tem a disposição todos os padrões do LAC. Isto inclui o padrão vazio. Este padrão cobre todas as instâncias do treinamento. Os demais padrões disponíveis cobrem somente as instâncias da projeção.

Considere que incluímos as instâncias de fora da projeção no conjunto de treinamento do BLACk. Todas estas instâncias adicionadas são cobertas apenas pelo padrão vazio. Por consequência nenhuma regra que utiliza qualquer padrão que não o vazio é capaz de classificar tais instância.

A cada iteração as instâncias cobertas pelo padrão da regra recebem novos valores e ao fim da normalização tanto as instâncias não cobertas pela regra quanto as instâncias cuja regra classifica erroneamente têm seus pesos incrementados. Isto significa que a cada iteração o valor de G para o padrão vazio vai ser maior e que a chance de escolhermos o padrão vazio para montarmos a nossa regra será maior. Por sua vez, a regra de padrão vazio sempre vai predizer a classe majoritária dentre as instâncias do treinamento (isto inclui as instâncias de fora da projeção nesse momento). A classe majoritário do conjunto completo de treinamento pode diferir da classe majoritária das instâncias da projeção o que pode levar a perda de acurácia por parte do nosso algoritmo.

6.9 Comparações

Dentre outras coisas, o BLACk oferece maior controle que o LAC quanto à quantidade de regras usadas para gerar cada modelo. Isto porque o processo do *boosting* nos habilita a determinar pelo máximo quantos classificadores serão usados para compor H controlando o número de iterações do algoritmo ("pelo máximo" porque uma mesma regra pode ser selecionada várias vezes). Em contraste, o LAC gera até $|Y|.2^{|X'|}$ regras (usando a mesma notação da Seção 4) e a única maneira de controlar o número de regras geradas é a partir de limiares de suporte e de confiança mínima. Outra diferença é que o BLACk utiliza o valor dos α's calculados para determinar a classe da instância que deu início à execução do algoritmo, enquanto o LAC utiliza as médias das confianças (vide Seção 4.4). Enquanto não existem teorias que justificam a agregação das confianças de todas as regras (suficientemente confiantes) aplicáveis à instância a ser classificada como forma de determinação da classe, Schapire & Freund [2012] comprovaram que o boosting minimiza o erro de treinamento a medida que T aumenta.

A principal diferença entre o SLIPPER e o BLACk é o modo de operação. O BLACk operar em modo *lazy* e tem a possibilidade de considerar todas as regras que se aplicariam à instância sendo classificada. Em contraponto, o SLIPPER opera em modo

eager e embora na teoria possa gerar todas as regras que se aplicam a determinada instância a ser classificada, na prática o SLIPPER não avalia todas as regras possíveis dado seu modo guloso de construção de regras.

Capítulo 7

Experimentos

Fizemos experimentos em 38 conjuntos de dados, 37 retirados do repositório público da Universidade da Califórnia em Irvine (UCI) e 1 retirado de uma competição do Kaggle. As descrições dos conjuntos de dados podem ser encontradas no Anexo A.

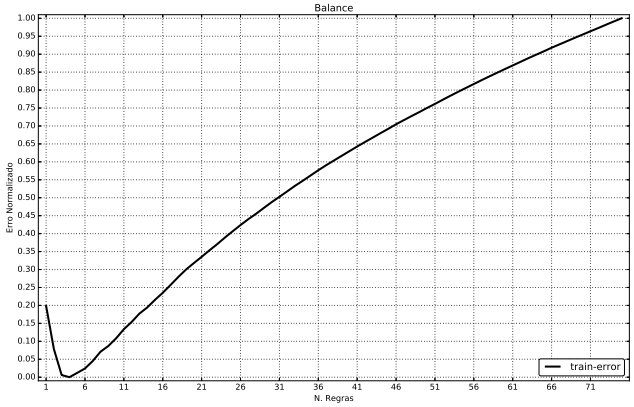
Em um primeiro momento, os nossos experimentos visam verificar o erro de treinamento e a complexidade dos padrões escolhidos pelo BLACk. Posteriormente verificaremos a qualidade do nosso algoritmo em termo de acurácia e tamanho dos modelos gerados. Por último, experimentaremos em um dado de mensagens do Twitter originário do Kaggle. Em especial, este último experimento visa comprovar o comportamento do BLACk em dados esparsos.

Para todos os testes do BLACk usamos 75 iterações de *boosting* e utilizamos apenas padrões fechados. Tanto para BLACk quanto para o LAC, escolhemos um valor pouco maior que 0 como limiar de suporte mínimo, pois queremos que todos os padrões sejam utilizados – os principais resultados publicados na literatura para o LAC utilizaram tanto o suporte mínimo dos padrões quanto a confiança mínima das regras com valores um pouco acima de 0, portanto existe uma coerência entre as experimentações. Para obter os resultados dos testes realizamos validação-cruzada de 10 fatias.

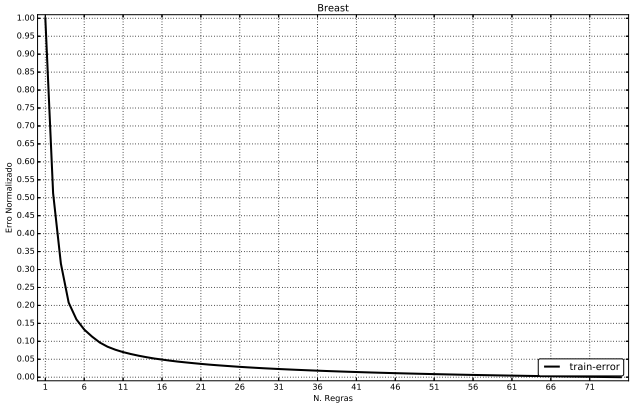
7.1 Erro de treinamento

O *boosting* tende a fazer com que o erro no treino decresça exponencialmente à medida que mais classificadores são adicionados ao modelo. Queremos verificar se o BLACk apresenta um efeito similar trabalhando em modo *lazy*. A Figura 7.1 mostra o comportamento do erro de treinamento dado pela Equação (5.4) de 15 conjuntos de dados.

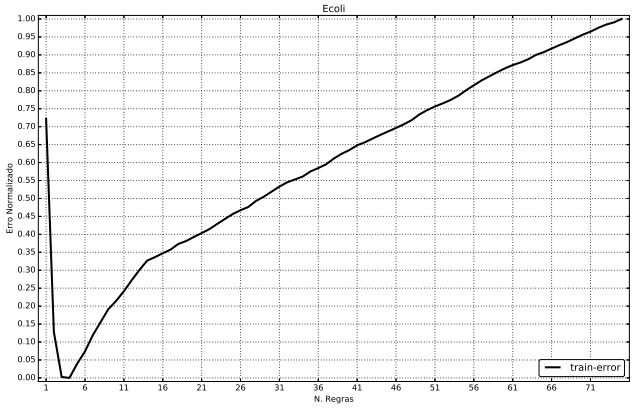
O eixo x indica o número de classificadores do modelo e o eixo y o valor normalizado do erro de treinamento.



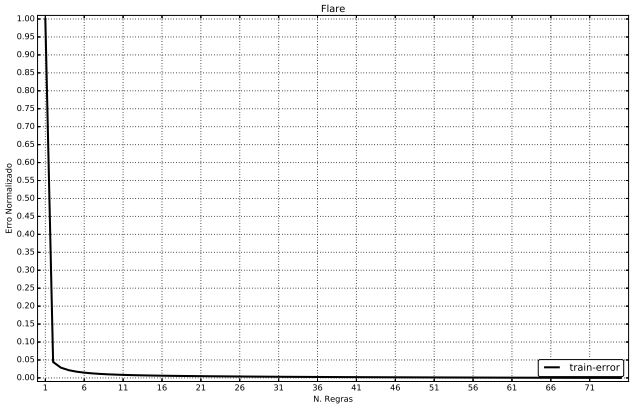
(a) Balance



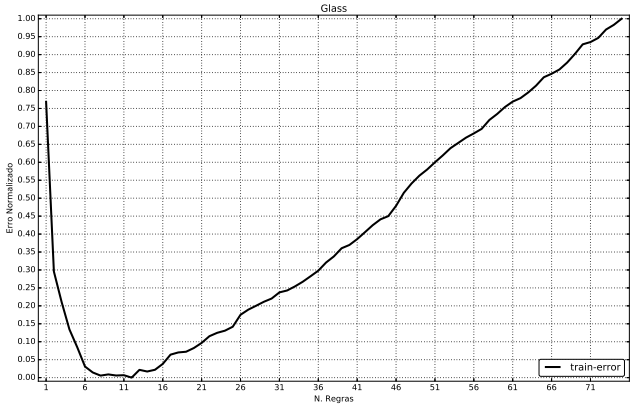
(b) Breast



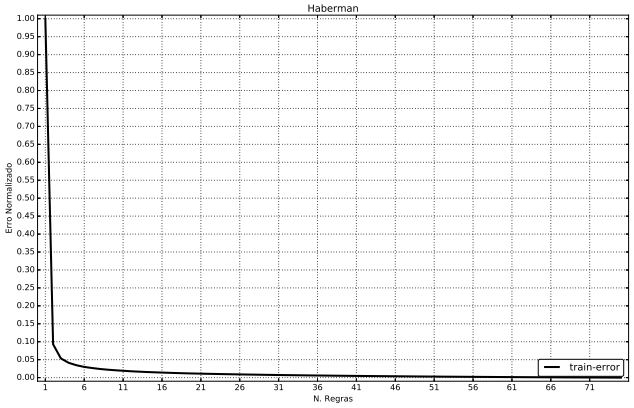
(c) Ecoli



(d) Flare

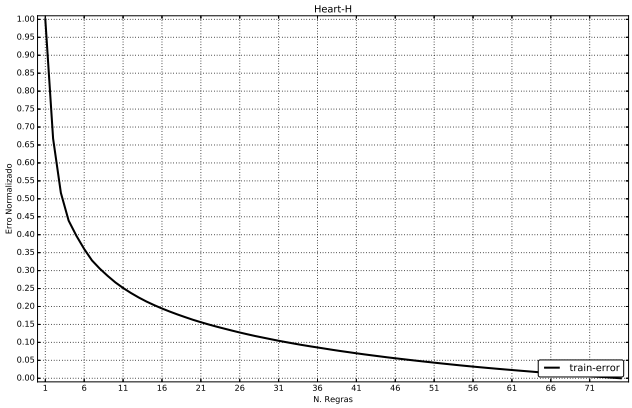


(e) Glass

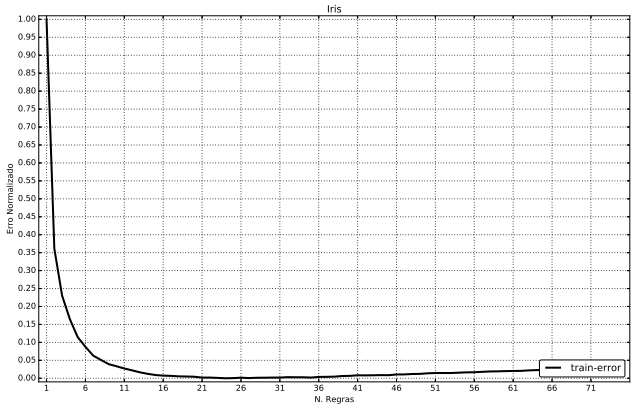


(f) Haberman

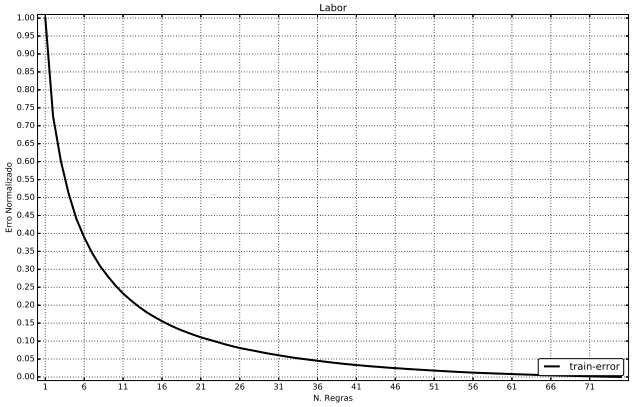
As imagens mostram o funcionamento esperado e que algumas curvas crescem logo



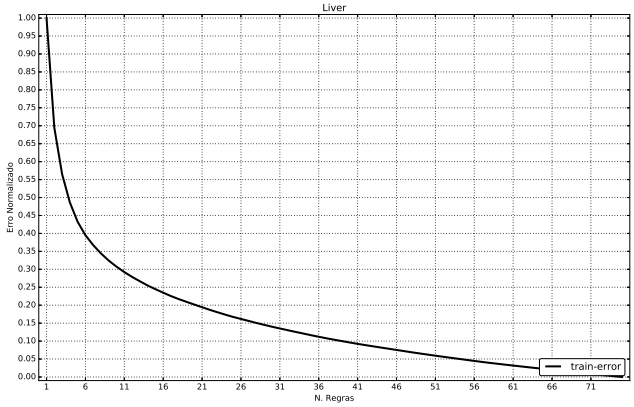
(g) Heart-h



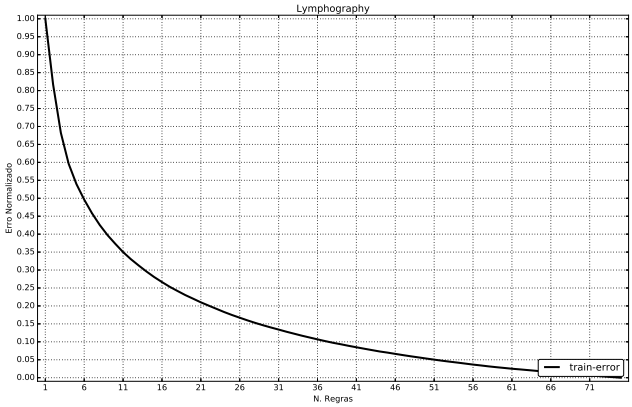
(h) Iris



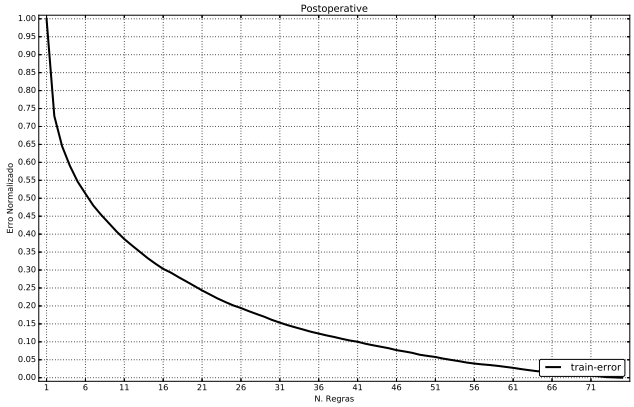
(i) Labor



(j) Liver

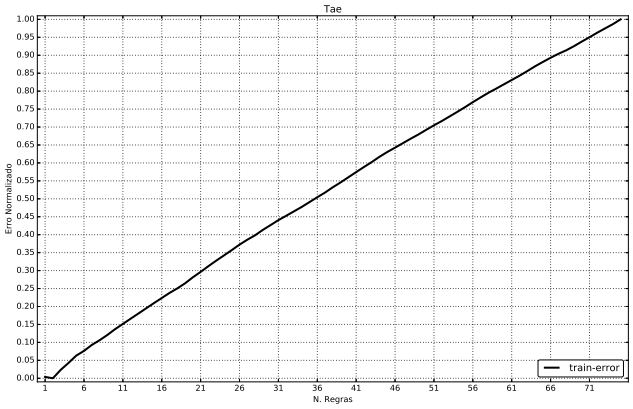


(k) Lymphography

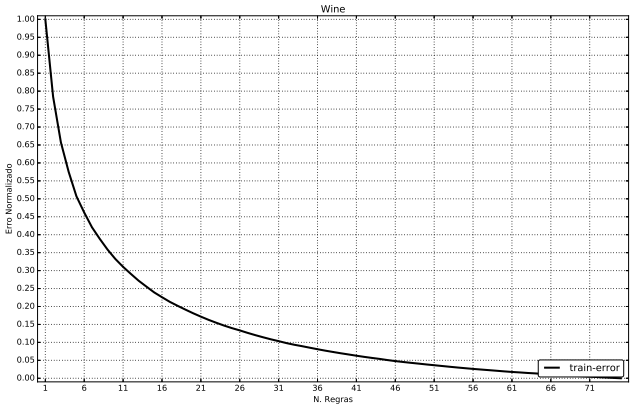


(l) Postoperative

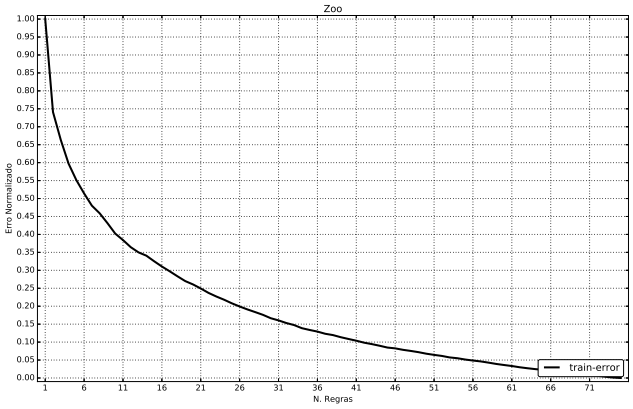
após poucas iterações. Isto indica que o algoritmo sofreu *overfitting* (superaprendeu) no conjunto de treinamento com apenas algumas regras. Caso quiséssemos usar o erro de treinamento como indicador de parada para o BLACk, estaríamos gerando



(m) Tae



(n) Wine



(o) Zoo

Figura 7.1: Erros de treinamento normalizados do BLACk.

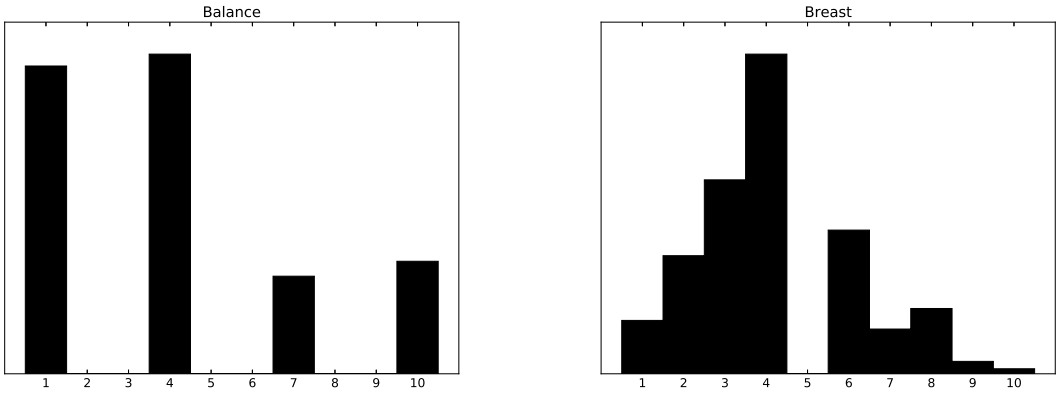
um classificador que explicaria de maneira bem simples (devido ao pequeno número de regras que seria humanamente interpretável) o motivo da classificação da instância. Contudo, não necessariamente, o erro de treinamento será um bom indicador de parada pois o *boosting* não possui uma relação muito forte entre o erro de treinamento e o erro de validação. Ou seja, mesmo que o algoritmo superaprenda no treinamento, ele pode estar melhorando a sua acurácia em instâncias não presentes no treinamento.

7.2 Caracterização dos padrões escolhidos

Nesta seção verificaremos quão complexos são as regras escolhidas pelo *boosting*. A complexidade de uma regra é dada pelo número de atributos em seu padrão. Para que possamos comparar a complexidade de regras em diferentes conjuntos de dados, assumimos, que a complexidade de uma regra é relativa ao modelo gerado. Isto é, dada

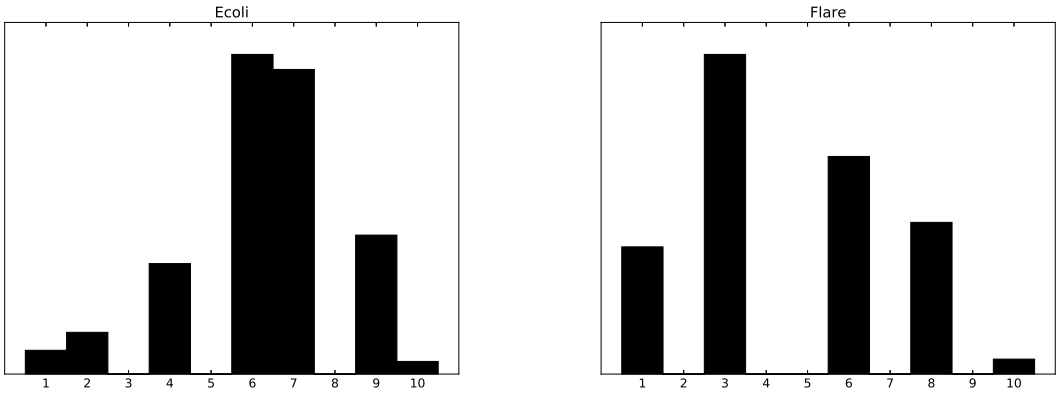
uma instância a ser classificada, o tamanho do maior padrão frequente encontrado a partir da instância a ser classificada dita o valor de complexidade máxima de regra para o modelo que será gerado para classificar aquela instância (o tamanho do menor padrão dita a complexidade mínima). Por exemplo, para uma instância composta de 10 atributos se o maior padrão frequente encontrado possui apenas 4 atributos, então 4 representa a complexidade máxima das regras encontradas para classificar aquela instância. Assim, independente do tamanho da instância sendo classificada, temos uma noção de regra complexa e regra simples.

Para a comparação dos resultados computamos os tamanhos das regras usadas nos modelos e normalizamos as complexidades para valores inteiros entre 1 e 10. Os resultados para 15 conjuntos do UCI podem ser observados nos histogramas da Figura 7.2.



(a) Balance

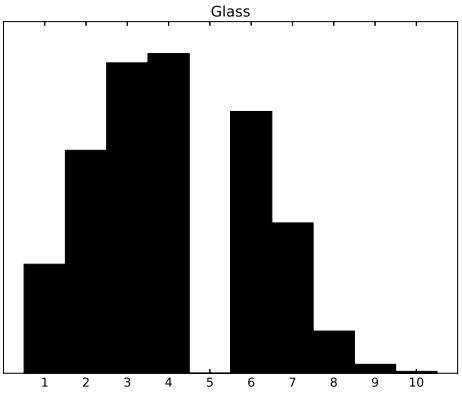
(b) Breast



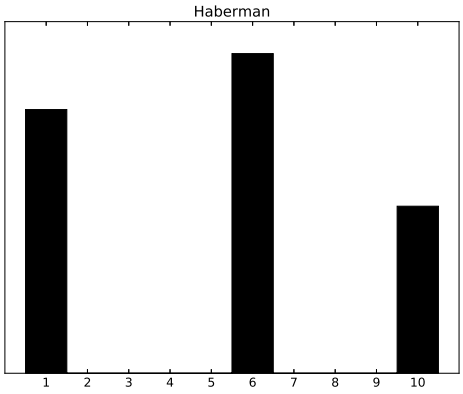
(c) Ecoli

(d) Flare

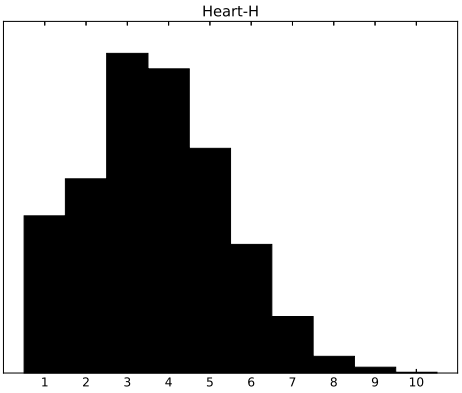
O algoritmo tende a escolher menos as regras de maior complexidade (isto é, regras com padrões maiores). Regras mais complexas são usadas para realizar ajustes finos nos pesos da distribuição visto que elas cobrem porções menor do espaço de dados



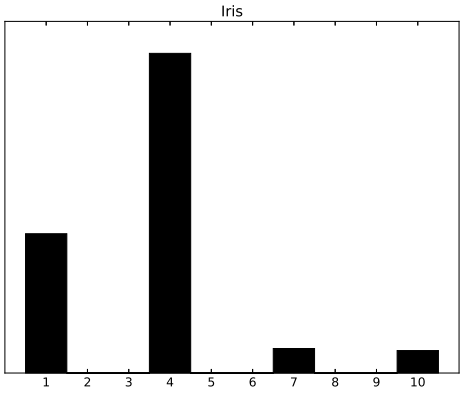
(e) Glass



(f) Haberman



(g) Heart-h

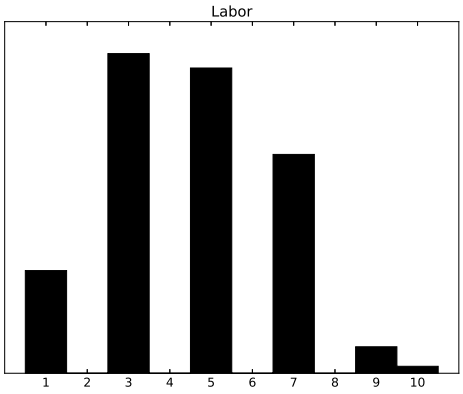


(h) Iris

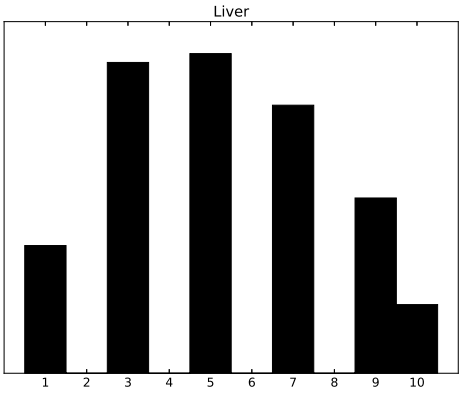
por serem mais específicas. É interessante notar que algumas imagens apresentam um aparente distribuição normal. Se necessário, poderíamos usar esta informação para que o minerador focasse seus esforços em determinados tamanhos de padrões, o que aceleraria a execução do algoritmo.

7.3 Acurácia e complexidades

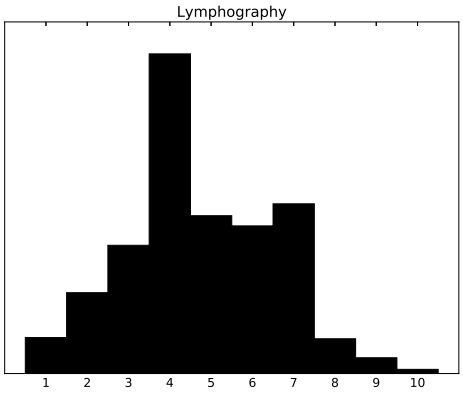
Alguns autores atestam que o *boosting* funciona melhor com classificadores mais simples, em nosso caso com regras cujos padrões possuem menos atributos. Tomando os mesmos 15 conjuntos de dados e os 10 níveis de complexidades definidos anteriormente verificamos qual o comportamento do BLACk à medida que ele tem acesso a padrões mais complexos. Ou seja, verificamos a acurácia do BLACk quando este tem acesso a apenas padrões de complexidade menor igual a k, $k \in \{1 \dots 10\}$. Este experimento têm o propósito de obter evidências que corroborem ou não a afirmação do início do



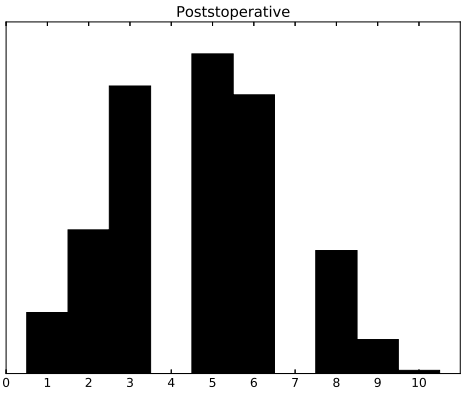
(i) Labor



(j) Liver



(k) Lymphography



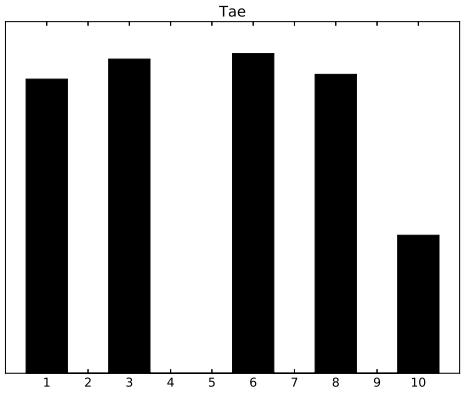
(l) Postoperative

parágrafo. A Figura 7.3 mostra os resultados das acurácias normalizadas. O eixo x representa o valor da complexidade máxima dos padrões k e o eixo y a acurácia normalizada dos modelos.

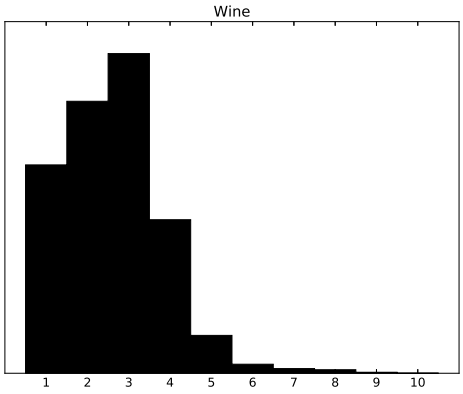
É possível verificar que algumas curvas chegam ao ponto máximo (ou seja, obtém a maior acurácia encontrada) antes de chegar à complexidade máxima (10). Isto indica que, não necessariamente, disponibilizar todos os padrões possíveis ao BLACk fará com que o algoritmo retorne o melhor modelo possível.

Não podemos afirmar que temos evidências que os melhores modelos são os que têm acesso somente aos padrões mais simples, mas também não podemos afirmar que os modelos são sempre mais acurados quando têm acesso a todos os padrões. De fato, apenas 6 das 15 curvas têm ponto máximo no ponto em que todos os padrões são disponibilizadas ao algoritmo.

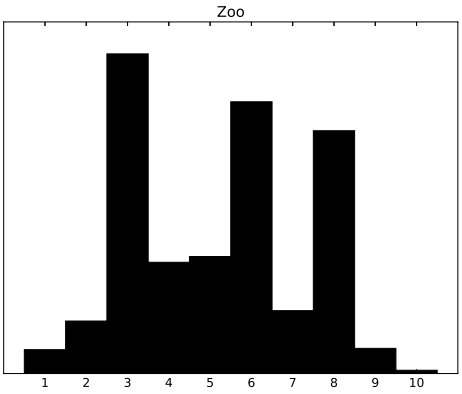
Portanto, seria vantajoso se o BLACk fosse configurado para usar uma faixar específica de complexidade de padrões para cada conjunto de dados. Para isso poderíamos mudar o funcionamento do *Miner* e economizar tempo de processamento



(m) Tae



(n) Wine



(o) Zoo

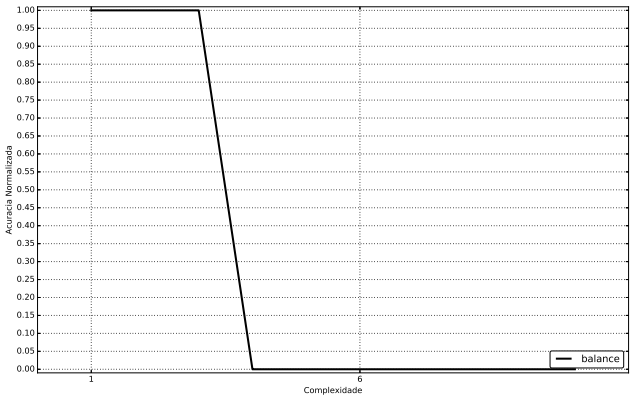
Figura 7.2: Complexidade dos padrões escolhidos

minerando padrões de tamanhos máximos e mínimos específicos. Contudo, determinar a faixa ideal de complexidade de padrões para cada um dos conjuntos de dados utilizados nos experimentos poderia ser custoso.

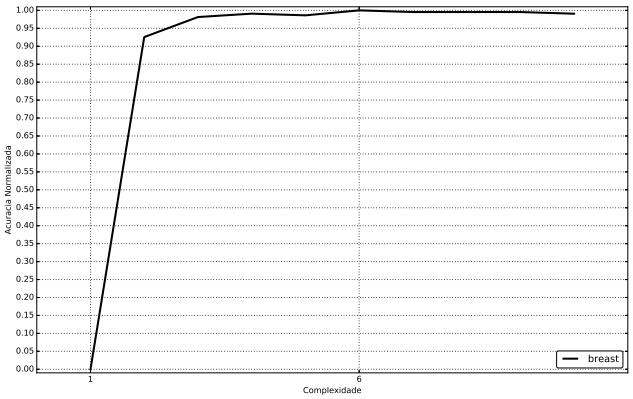
7.4 Instâncias de fora da projeção

Verificamos a diferença no número de vezes que a regra de padrão vazio é selecionada quando permitimos ou não que instâncias de fora da projeção façam parte dos dados de treinamento para os mesmos 15 conjuntos de dados que vinhamos utilizando nas seções acima. Em casos onde uma regra cobre todas as instâncias do conjunto de dados, isto é, uma regra que cobre as mesmas instâncias que a regra vazia, esta regra foi contada como regra vazia, visto que a regra vazia poderia ser substituída por esta regra.

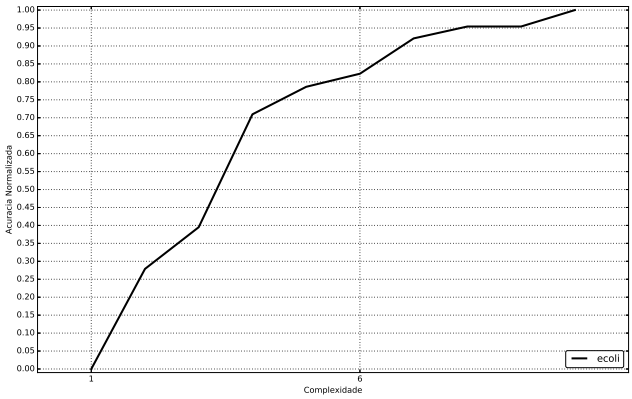
A Tabela 7.1 exibe os resultados dos experimentos. Nas colunas sob a coluna



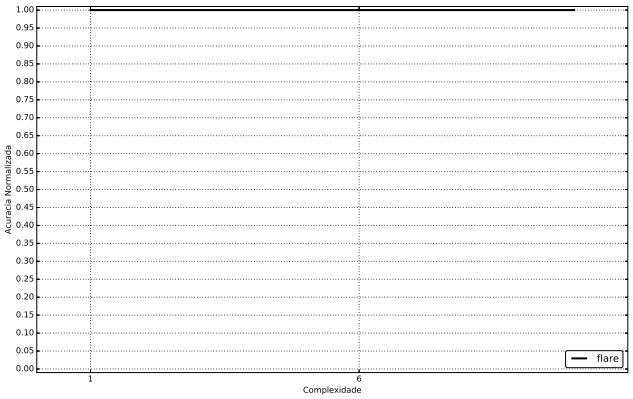
(a) Balance



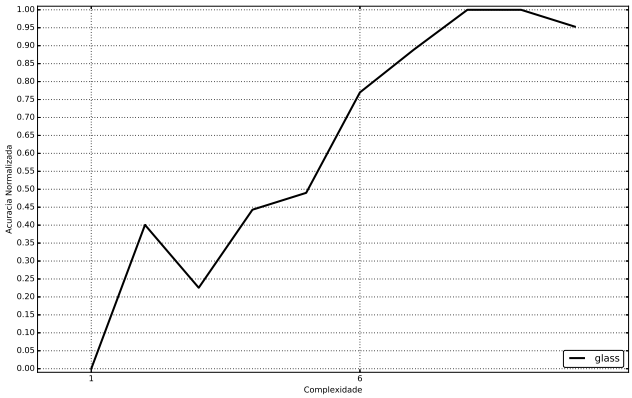
(b) Breast



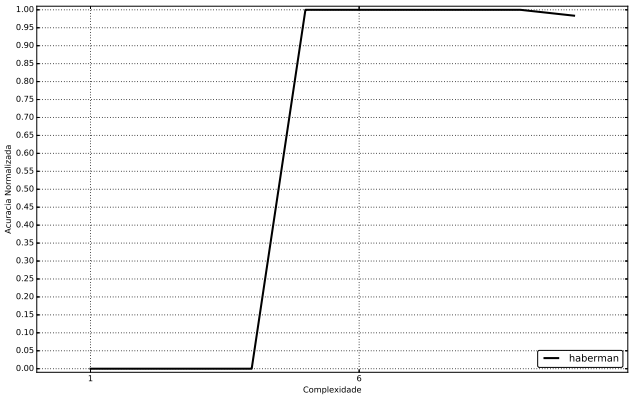
(c) Ecoli



(d) Flare

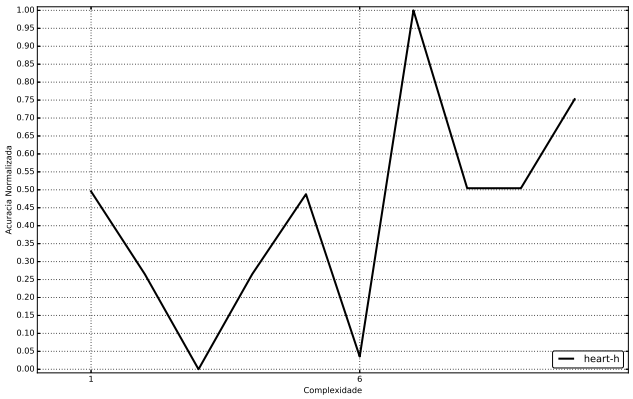


(e) Glass

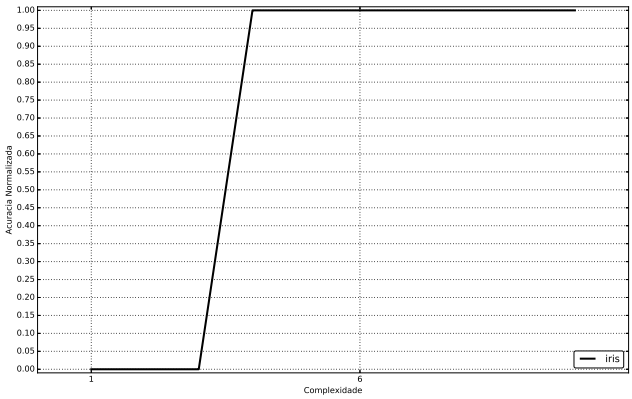


(f) Haberman

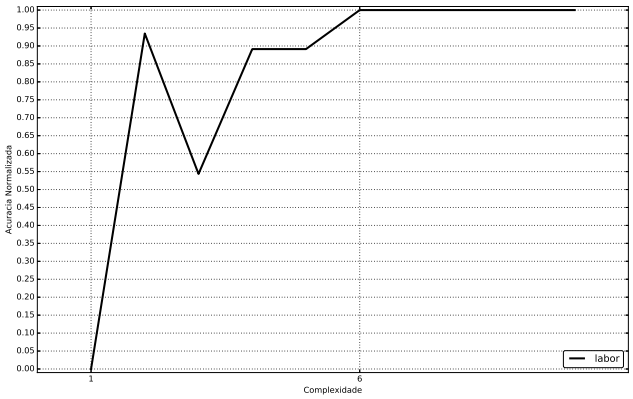
Acurácia, na coluna LAC exibimos a acurácia média da validação-cruzada do LAC, na coluna BLACk exibimos a diferença entre a acurácia média atingida pelo BLACk em relação ao LAC e na coluna BLACK-E exibimos a diferença entre a acurácia média



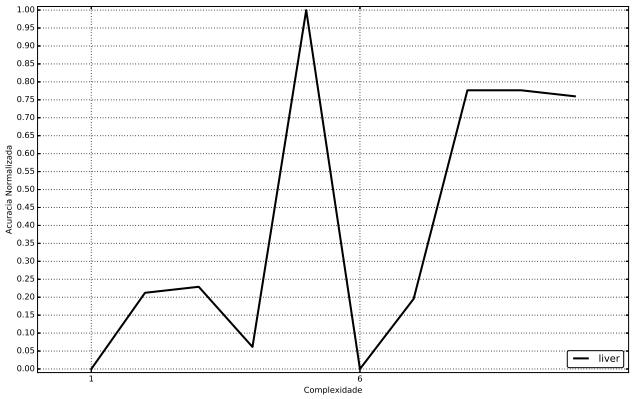
(g) Heart-h



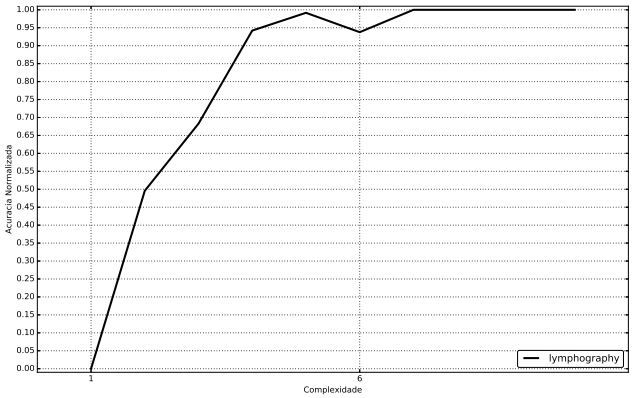
(h) Iris



(i) Labor



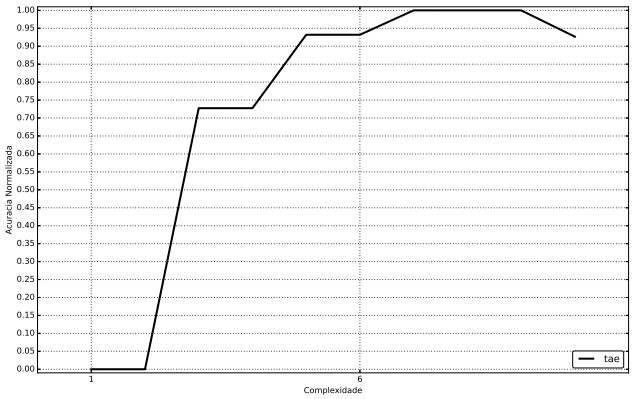
(j) Liver



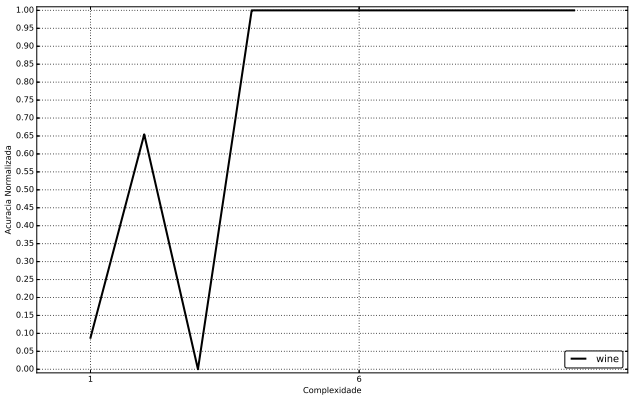
(k) Lymphography

(l) Postoperative

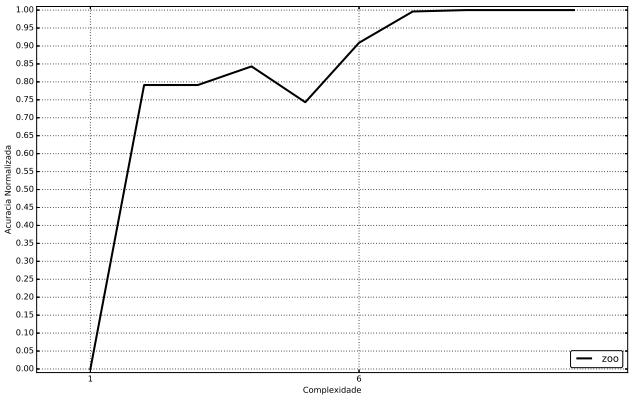
atingida pelo BLACk usando a instâncias de fora da projeção em relação ao LAC. Nas colunas sob a coluna *Regras P. Vazio (Média)* exibimos o número médio de vezes que a regras de padrão vazio foi selecionado para fazer parte do modelo final no BLACk e



(m) Tae



(n) Wine



(o) Zoo

Figura 7.3: Acurácia dada a Complexidade dos Padrões disponíveis

no BLACk-E.

Como é possível verificar, na maioria dos casos o BLACk-E utiliza mais regras vazias que o BLACk comum, já que à medida que o peso das instância vai sendo redistribuído e acumulado sobre as instâncias de fora da projeção a regra de padrão vazio vai obtendo um valor de G mais alto. Por um lado, talvez fosse interessante a alguma aplicação o uso das instâncias de fora da projeção, mas não encontramos evidência suficiente para afirmar que esta seja a melhor prática para a nossa abordagem. Além disso a acurácia do BLACk comum prevaleceu sobre a do BLACk-E.

7.5 Tamanho dos modelos

O BLACk oferece a possibilidade de limitar o tamanho de seu modelo, isto é, o número máximo de classificadores. Contudo, o número de iterações não é o único fator que

		Acurácia			Regras P. Vazio (Média)	
ID	Nome	LAC	BLACk	BLACk-E	BLACk	BLACk-E
1	balance	0,688	0,072	0,062	4,3	2,3
2	breast	0,959	0,001	0,004	3,7	14,2
3	ecoli	0,818	-0,012	-0,015	0,7	2,6
4	flare	0,978	0,000	0,000	8,9	11,9
5	glass	0,682	0,056	0,070	0,4	1,8
6	haberman	0,729	0,013	0,013	2,5	2,5
7	heart-h	0,827	0,014	0,003	1,0	8,7
8	iris	0,933	0,027	0,013	3,5	1,9
9	labor	0,930	0,000	-0,018	1,3	6,1
10	liver	0,600	0,012	0,020	0,8	6,3
11	lymphography	0,797	0,068	0,047	0,1	5,5
12	postoperative	0,711	0,000	0,000	0,4	5,1
13	tae	0,550	0,007	0,013	2,6	3,2
14	wine	0,933	0,022	0,017	1,6	4,4
15	zoo	0,871	0,099	0,099	0,0	3,5
		Acurácia	**0,025**	0,022		

Tabela 7.1: Média de uso de regras vazias pelo BLACk comum e pelo BLACk-E (cuja projeção contém instâncias vazias).

define o tamanho do modelo resultante do BLACk. Se uma regra for escolhida mais que uma vez, podemos compactar o conjunto de regras somando os valores dos *alpha*'s das regras idênticas. Chamaremos de BLACk-U o modelo do BLACk cujas regras idênticas são compactadas.

Neste experimento, assim como nos experimentos anteriores, escolhemos um total de 75 iterações para cada conjunto de dados. Observamos para o LAC, o BLACk e BLACk-U o número de regras em seus modelos e o número de atributos das regras dos modelos gerados para cada um dos 37 conjuntos de dados do UCI. Os resultados estão listados na Tabela 7.2. As colunas *Regras* informam o número médio de regras usadas pelos modelos gerados por cada algoritmo para classificar as instâncias das 10 fatias de validação, as colunas *Atribs.* informam o número médio de atributos presentes nos modelos gerados de cada algoritmo para classificação das instâncias e a coluna *Iter.* informa o número de iterações necessário para obter a máxima acurácia média para o BLACk em cada conjunto.

Note que para alguns conjuntos de dados o valor do *Iter.* é bastante inferior a 75, por exemplo `colic` = 1. Para que fique claro, é importante lembrar que o BLACk trabalha de forma *lazy* gerando um modelo para cada instância a ser classificada. Logo,

		LAC		BLACk			BLACk-U	
ID	Nome	Regras	Atribs.	Regras	Atribs.	Iter.	Regras	Atribs.
1	anneal	1963	13242	75	421	17	31	176
2	australian	5276	30011	75	334	6	63	283
3	autos	6074	43241	75	549	73	40	260
4	balance	48	99	75	152	2	14	27
5	breast	386	1917	75	293	12	27	106
6	car	252	747	75	177	14	26	65
7	cmc	891	3625	75	337	4	44	177
8	colic	6987	38714	75	293	1	58	229
9	credit-a	5329	32917	75	350	21	59	278
10	credit-g	17356	181281	75	763	16	67	678
11	crx	7809	50611	75	393	2	63	335
12	dermatology	579832	6029470	75	950	8	54	638
13	diabetes	330	1225	75	258	7	31	102
14	ecoli	229	828	75	327	42	16	56
15	flare	31	305	75	715	1	9	86
16	glass	697	2468	75	281	18	27	87
17	haberman	14	22	75	139	63	6	10
18	heart-c	2402	11364	75	268	3	53	189
19	heart-h	937	4118	75	276	17	39	139
20	heart-statlog	1578	8690	75	324	28	49	210
21	hepatitis	27549	197434	75	420	38	48	267
22	hypothyroid	655	15624	75	1672	5	16	352
23	iris	33	67	75	140	2	7	11
24	labor	189	779	75	207	3	15	43
25	liver	88	250	75	239	9	19	52
26	lymphography	10218	76349	75	485	33	44	276
27	mushroom	4729	43478	75	385	18	27	140
28	postoperative	361	1275	75	290	18	25	85
29	primary-tumor	258907	1808310	75	730	1	44	358
30	segment	58329	475717	75	546	26	55	382
31	sick	1512	33857	75	1620	21	21	467
32	tae	72	176	75	207	8	16	37
33	tictactoe	792	3259	75	211	2	53	155
34	vehicle	13639	85278	75	434	15	56	300
35	vote	21223	139545	75	200	21	42	125
36	wine	2917	13942	75	197	16	36	98
37	zoo	6749	39885	75	565	18	23	142

Tabela 7.2: Tamanho dos modelos.

dizer que o modelo que leva à máxima acurácia tem tamanho 1 significa dizer que apenas a primeira regra escolhida pelo *boosting* e o seu respectivo valor de α em cada par (instância a classificar, projeção) seriam suficientes para obter a melhor acurácia média nos conjuntos de validação.

Em média o LAC possui mais de 350 vezes o número de regras do BLACk e mais de 800 vezes o número de regras do BLACk-U. Em média o BLACk-U é menor em todos os conjuntos de dados. De fato, o número de iterações escolhidas para o *boosting*

foi bem pequeno (75, enquanto na literatura encontra-se *boostings* de mais de 1000 classificadores), ainda assim, como pode ser entendido pela coluna *Iter.*, o máximo valor de acurácia média dos modelos gerados acontece antes das 75 iterações. Ou seja, a diferença dos valores da tabela poderia ser ainda maior se considerássemos os valores de iterações específicos de cada conjunto de dados para o BLACk.

7.6 Acurácia dos modelos

O BLACk, assim como outros algoritmos de aprendizado de máquina, precisa ser ajustado a cada conjunto de dados, a saber o número de regras utilizadas pelo algoritmo deve ser escolhido. Uma maneira comum de se escolher tal valor é por meio de validação cruzada, testando-se vários valores para o número de iterações (isto é, o número máximo de regras) e escolhendo o valor cuja acurácia média se mostrar a maior. O ajuste do algoritmo é simples, demandando apenas tempo para o processamento de todas as iterações.

Na Tabela 7.3 exibimos na primeira coluna as acurácias médias do LAC para a série de conjuntos de dados, as demais colunas apresenta a diferença entre o valor média das acurácias em relação ao LAC. Os valores do BLACk referem-se à melhor acurácia média encontrada para um intervalo de 1 a 75 iterações. A acurácia média de cinco algoritmos de classificação que utilizam regras de associação também são exibidas na tabela, a saber: CBA, CMAR, CPAR, FOIL, PRM. Para o CBA e para o CMAR, nós experimentamos 15 configurações de valores de parâmetros de confiança e suporte mínimos e os melhores resultados foram reportados. Ainda reportamos resultados das acurácias médias do C4.5, RIPPER e do SLIPPER. As implementações destes algoritmos foram providas por uma *suite* de algoritmos chamada KEEL (usamos as configurações *default* da *suite* para todos estes algoritmos)[1]. A coluna de título BLACk-ICV refere-se ao BLACk com validação-cruzada interna que discutimos no final do capítulo 6. A última linha da tabela apresenta a média dos valores das colunas. Para os conjuntos de dados que não foi possível obter valor o símbolo "-" foi utilizado.

7.6.1 Comparando os modelos

Visto que temos cerca de 40 amostras, o teste de Shapiro-Wilk pode ser usado para determinar a normalidade dos resultados dos algoritmos. Os resultados deste teste são exibidos na Tabela 7.4 sob a coluna *W*. A maioria dos resultados atestam amostras

[1]Para os algoritmos do KEEL, os dados não foram normalizados e discretizados como para os demais algoritmos. A discretização pode levar à perda de informações.

ID	Nome	LAC	BLACk	BLACk-ICV	CBA	CMAR	CPAR	FOIL	PRM	C4.5	RIPPER	SLIPPER
1	anneal	0,763	0,198	0,194	0,145	0,073	0,185	0,223	0,189	0,191	0,230	0,211
2	australian	0,859	-0,003	-0,004	0,002	0,003	-0,004	-0,012	-0,004	-0,012	-0,029	-0,058
3	autos	0,622	0,167	0,151	-0,193	-0,312	-0,327	-0,234	-0,328	0,124	0,154	0,026
4	balance	0,688	0,072	0,070	0,011	0,005	-0,038	0,060	-0,038	0,119	-0,186	0,166
5	breast	0,959	0,001	-0,003	0,001	-0,013	-0,025	-0,213	-0,023	-0,006	-0,013	-0,007
6	car	0,719	0,237	0,198	0,075	-0,019	0,024	-0,449	0,019	0,134	0,175	0,201
7	cmc	0,520	-0,024	-0,015	-0,205	-0,233	-0,099	-0,196	-0,063	-0,018	0,018	-0,003
8	colic	0,832	0,022	0,022	0,010	0,005	-0,007	0,016	-0,007	-0,019	0,005	-0,196
9	credit-a	0,869	0,006	-0,004	0,004	-0,037	-0,004	-0,061	-0,004	-0,030	-0,056	-0,051
10	credit-g	0,701	0,041	0,047	0,009	-0,001	0,004	0,129	0,002	0,001	-0,019	-0,017
11	crx	0,861	0,006	-0,003	0,000	-0,042	-0,006	-0,018	-0,006	-0,008	-0,028	-0,074
12	dermatology	0,887	0,048	0,053	-0,349	-0,551	0,014	0,038	0,001	0,050	0,023	-0,111
13	diabetes	0,712	0,025	0,026	0,021	-0,035	0,033	0,017	0,038	0,013	-0,019	0,016
14	ecoli	0,819	-0,012	-0,006	-0,196	-0,318	-0,170	-0,165	-0,164	-0,015	-0,048	-0,104
15	flare	0,979	0,000	0,000	-0,115	0,000	-0,102	-0,006	-0,102	-0,009	-0,037	-0,022
16	glass	0,682	0,055	0,045	-0,064	-0,087	-0,024	-0,011	-0,043	-0,057	-0,026	0,023
17	haberman	0,729	0,013	0,009	0,010	-0,210	-0,117	0,010	-0,117	0,003	-0,121	-0,026
18	heart-c	0,843	-0,007	-0,001	-0,005	-0,001	-	-0,046	-	-0,047	-0,096	-0,080
19	heart-h	0,827	0,013	0,010	-0,012	-0,039	-	-0,126	-	-0,003	-0,068	-0,054
20	heart-statlog	0,852	-0,011	0,007	-0,007	-0,011	-0,059	-0,056	-0,063	-0,052	-0,104	-0,033
21	hepatitis	0,794	0,079	0,027	0,030	-0,005	-0,097	-0,006	-0,099	0,006	0,031	0,038
22	hypothyroid	0,924	0,011	0,011	-	-0,001	-	-0,001	-	0,067	0,067	0,050
23	iris	0,933	0,027	0,033	0,007	0,013	0,013	0,007	0,013	0,020	0,020	0,020
24	labor	0,923	0,003	-0,013	-0,117	-0,010	-0,140	-0,112	-0,140	-0,103	-0,013	-0,087
25	liver	0,600	0,011	0,020	-0,018	-0,096	-0,137	-0,037	-0,137	0,040	0,034	0,083
26	lymphography	0,797	0,068	0,061	0,047	0,034	-0,070	0,155	-0,056	-0,025	-0,053	0,021
27	mushroom	0,971	0,029	0,029	0,025	0,015	0,015	0,029	0,015	0,027	0,029	0,029
28	postoperative	0,715	0,000	0,000	-0,004	-0,560	-0,149	-0,071	-0,226	-0,126	-0,304	-0,093
29	primary-tumor	0,427	-0,032	-0,028	-0,026	-0,029	-0,009	0,295	-0,009	-0,029	-0,150	-0,044
30	segment	0,889	0,074	0,073	-0,774	-	-0,010	0,044	-0,007	0,064	0,061	0,072
31	sick	0,939	0,029	0,029	-	-0,850	-	0,001	-	0,039	0,043	0,041
32	tae	0,550	0,006	0,000	-0,081	-0,120	-0,060	-0,184	-0,060	-0,072	-0,079	0,053
33	tictactoe	0,774	0,216	0,216	-0,020	-0,275	-0,285	0,072	-0,282	0,101	0,203	0,209
34	vehicle	0,664	0,014	0,017	-0,015	-0,072	-0,017	0,039	-0,026	-0,010	0,024	0,085
35	vote	0,919	0,034	0,035	0,025	0,023	0,035	0,023	0,035	0,037	-0,032	0,028
36	wine	0,934	0,022	0,011	-0,096	-0,063	-0,098	-0,069	-0,098	-0,002	-0,008	0,010
37	zoo	0,874	0,098	0,079	-0,234	-0,471	0,009	0,048	-0,001	0,058	-0,013	0,028
		Média	**0,042**	0,038	-0,060	-0,119	-0,052	-0,023	-0,054	0,012	-0,010	0,010

Tabela 7.3: Resultados: Acurácias

	Shapiro-Wilk		Mann-Whitney			
Algoritmo	W	p-value	U	p-value	U	p-value
BLACk	0,875	0,001	-	-	-	-
BLACK-ICV	0,886	0,001	662,5	0,408	-	-
LAC	0,940	0,046	524,0	0,042	549,5	0,073
CBA*	0,902	0,005	396,5	0,006	413,0	0,010
CMAR*	0,917	0,010	381	0,001	393	0,002
CPAR*	0,940	0,067	359,0	0,009	369,0	0,012
FOIL	0,872	0,001	515,0	0,034	528,0	0,046
PRM*	0,937	0,055	354,0	0,007	369,0	0,012
C4.5	0,916	0,009	579,0	0,128	593,0	0,163
RIPPER	0,919	0,011	566,0	0,101	582,5	0,136
SLIPPER	0,938	0,040	574,0	0,117	591,0	0,157
			BLACk		BLACk-ICV	

*Cálculo de normalidade realizado com menos de 37 valores.

Tabela 7.4: Resultados: Teste de normalidade e teste pareado de diferença

de dados que não apresentam distribuição normal para um nível de confiança de 5% (p-value < 0.05). Logo, não poderemos usar o teste de Student para determinar se dois algoritmos são estatisticamente diferentes. Usaremos, portanto, o teste de Mann-Whitney, exibidos sob as colunas U foi utilizado para comparar todos os algoritmos ao BLACk e ao BLACk-ICV.

Observando o valor do p-value no cruzamento do BLACk com o LAC (p-value = 0,042), somos obrigados a rejeitar a hipótese nula, isto é, temos que o LAC é estatisticamente diferente do BLACk; e observando os resultados da Tabela 7.3, concluímos que existe um ganho de acurácia do BLACk em relação ao LAC. Quando comparados aos demais algoritmos de classificação associativa, o BLACk também foi estatisticamente superior.

Embora o BLACk tenha sido superior ao C4.5, RIPPER e SLIPPER de acordo com a tabela 7.3, não foi possível comprovar uma diferença estatística entre o BLACk e os algoritmos citados para o nível de confiança mais comumente aceito de 5%. A diferença estatística, seguida da comprovação de superioridade do BLACk para o SLIPPER, só poderia ser dada para uma confiança acima de 12%, por exemplo. Ainda assim, os dois algoritmos trabalham de modo diferente e, para algumas aplicações, pode-se aconselhar o uso do BLACk no lugar do SLIPPER. As mesmas observações valem para as comparações entre o BLACk e o RIPPER, e o BLACk e o C4.5.

O BLACk-ICV se mostrou estatisticamente igual ao BLACk para o nível de confiança de 5%, mas a acurácia média do BLACk foi maior. O BLACk-ICV foi superior ao LAC para um nível de confiança de 7,3% (p-value = 0,073). Com exceção do C4.5,

RIPPER e SLIPPER o BLACk-ICV se mostrou estatisticamente diferente e foi superior em acurácia a todos os algoritmos (com exceção do BLACk).

7.7 Aplicação em dados esparsos

O número de instâncias selecionadas para compor a projeção depende da frequência dos atributos da instância a ser classificada, de modo que é provável que em dados mais esparsos (onde a frequência dos atributos é considerada baixa) o tamanho da projeção seja menor que o tamanho dos dados completos. Nesta seção usaremos um conjunto de dados de textos do Twitter (rede social) que foram coletados pela Universidade de Michigan e publicados como parte de um dos desafios no Kaggle (comunidade de análise de dados e competições de tarefas de aprendizado de máquina). O conjunto de dados é composto por 7086 instâncias. O desafio é predizer o sentimento (positivo ou negativo) do texto com base nas palavras contidas nos *tweets*.

Textos de redes sociais costumam conter informalidades, regionalismos e/ou neologismos, além de outras características como partes do texto em maiúsculo, uso exacerbado de pontuações e repetição de uma mesma letra, além de erros ortográficos. Neste tipo de dado, por exemplo, é possível que `Gol` e `goool` tenham sentidos diferentes. Isto pressupõe que deve ser feito um trabalho cuidadoso de pré-processamento dos dados. Contudo, não é nosso objetivo conseguir capturar todos as nuances dos textos a fim de obter a melhor acurácia possível, estamos interessados apenas em verificar a qualidade do BLACk em dados esparsos onde não existe um super pré-processamento do texto.

Ao prepararmos o texto, todas as palavras foram transformadas em minúsculas e todas as pontuações foram removidas. Após isso, um valor de *hashing* foi atribuído a cada palavra. Perceba que o universo de palavras é, por si só, enorme, uma vez que pode conter qualquer palavra do dicionário da língua e `gol` e `gool` seriam entendias como palavras diferentes. É provável que uma dada palavra w aconteça em poucas instâncias, ou até que esta exista apenas em instâncias a ser classificada. Ainda assim, após esse rápido tratamento dos dados, ficamos com pouco mais de 2000 valores de *hashing* diferentes. Isto é possuímos pouco mais de 2000 atributos binários. Se uma palavra aparece em uma frase a ser classificada seu atributo binário correspondente tem valor 1, caso contrário seu atributo tem valor 0. Procurar por todos os padrões frequentes, considerando um valor de suporte próximo a zero, seria impossível se não estivéssemos trabalhando em modo *lazy* (existem 2^{2000} padrões possíveis nos dados).

Computamos a acurácia média da validação-cruzada para o LAC e para 75 itera-

ções no BLACk. Os dados podem ser visualizados na Figura 7.4. O eixo x apresenta o número de regras presentes no modelo e o eixo y o respectivo valor médio da acurácia. Em média, o LAC usa aproximadamente 181 regras para classificar cada instância. A partir da imagem, é possível verificar que o BLACk consegue uma melhor acurácia usando menos de 5 regras em cada modelo. Isto significa que o BLACk tem um modelo mais humanamente interpretável que o LAC. Além disso, evidenciamos que existe uma viabilidade no uso do BLACk em dados esparsos, especialmente se estivermos buscando explicações sobre o porquê de uma instância ter sido classificada como sendo de determinada classe.

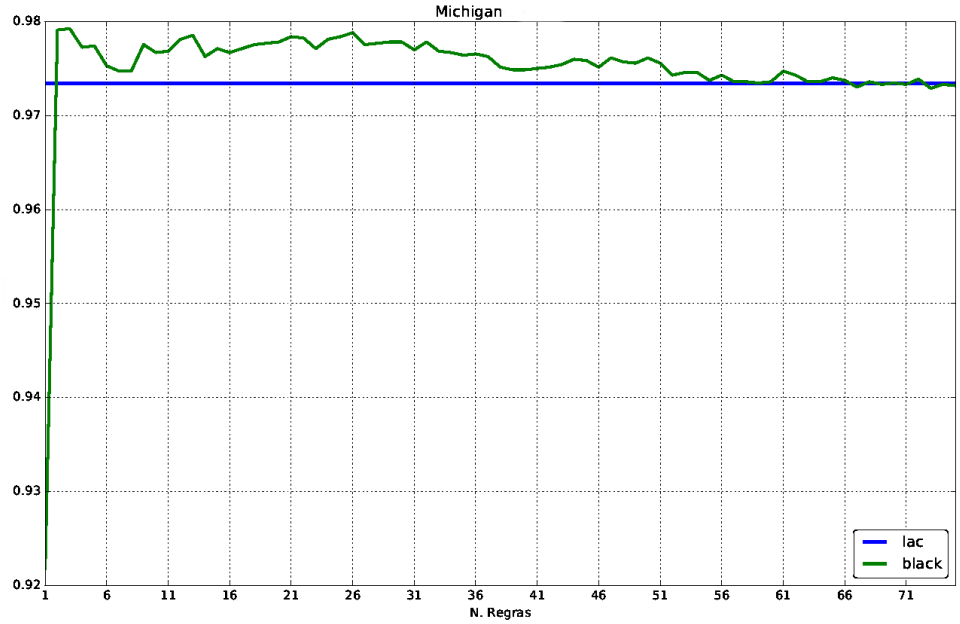


Figura 7.4: Conjunto de dados Twitter: Acurácia

Capítulo 8

Conclusão

Ao longo deste documento descrevemos o conceito de problemas de classificação, uma das mais notórias tarefas da área de aprendizado de máquina, e como este é um problema complexo. Discutimos que algoritmos de aprendizado de máquina usam conjuntos de dados de treinamento e resolvem problemas de classificação construindo modelos que mapeia instâncias às classe do problema. Dentre várias estratégias propostas na literatura focamos nossa discussão em algoritmos que usam regras de decisão por estas gerarem modelos classificadores que são humanamente interpretáveis. Descrevemos o *Lazy Associative Classifier* (LAC) como um algoritmo que tira vantagem de cada instância a se classificada projetando os dados e gerando apenas regras que cobrem a instância. A intuição chave por trás do LAC é que um problema complexo pode ser decomposto em vários sub-problemas bem mais simples e solucionáveis independentemente. Contudo, o número de regras geradas e o uso indiscriminado de todas as regras por parte do algoritmo foram apontados como características indesejáveis.

Propomos, então, o *Boosted Lazy Associative Classifier of k rules* (BLACk) como um algoritmo baseado em *boosting* de regras e que soluciona os problemas do LAC ao mesmo tempo que herda suas vantagens. Mostramos que o BLACk consegue reduzir o erro de treinamento, o que atesta o funcionamento normal do algoritmo. Mostramos que o BLACk é estatisticamente superior ao LAC (com uma confiança de 5%) no que diz respeito à acurácia do algoritmo e que o BLACk constrói modelos menores que o LAC. Sendo estas duas as motivações deste trabalho. O BLACk também se mostrou superior os outros algoritmos de classificação associativa. Quando comparado ao C4.5, ao RIPPER e ao SLIPPER não foi possível comprovar uma superioridade estatística do BLACk para 5% de confiança, apenas para um valor pouco maior que 12% de confiança. Aconselhamos o uso do BLACk em vezes destes três últimos algoritmos em situações onde o conjunto de dados está constantemente sendo modificado. Por

fim, demonstramos a qualidade e viabilidade do BLACk em problemas de análise de sentimentos em dados esparsos.

Concluímos que a decomposição do problema de classificação em vários subproblemas menores não impediu o *boosting* de melhorar a acurácia dos modelos, e que por isso o BLACk foi bem sucedido. O *boosting* também proporcionou a possibilidade do controle do tamanho dos modelos, o que é uma característica desejável para alguns tipos de dados. Sobretudo, comprovamos que o uso indiscriminado das regras e o modelo de predição do LAC afetam a qualidade do algoritmo, e que o BLACk é uma opção viável e facilmente implementável para substituir o LAC.

Referências Bibliográficas

Agrawal, R.; Srikant, R. et al. (1994). Fast algorithms for mining association rules. Em *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pp. 487--499.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123--140.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5--32.

Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754.

Cohen, W. W. (1995). Fast effective rule induction. Em *Proceedings of the twelfth international conference on machine learning*, pp. 115--123.

Cruz, J. A. & Wishart, D. S. (2006). Applications of machine learning in cancer prediction and prognosis. *Cancer informatics*, 2.

Dasarthy, B. (1990). Nearest neighbor pattern classification techniques.

Dass, R. & Mahanti, A. (2006). An efficient heuristic search for real-time frequent pattern mining. Em *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on*, volume 2, pp. 40b--40b. IEEE.

Dietterich, T. G. (2000). Ensemble methods in machine learning. Em *International workshop on multiple classifier systems*, pp. 1--15. Springer.

Drucker, H.; Schapire, R. & Simard, P. (1993). Improving performance in neural networks using a boosting algorithm. Em *Advances in neural information processing systems*, pp. 42--49.

Duda, R. O.; Hart, P. E. et al. (1973). *Pattern classification and scene analysis*, volume 3. Wiley New York.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189--1232.

Friedman, J. H.; Kohavi, R. & Yun, Y. (1996). Lazy decision trees. Em *AAAI/IAAI, Vol. 1*, pp. 717--724.

Galindo, J. & Tamayo, P. (2000). Credit risk assessment using statistical and machine learning: basic methodology and risk modeling applications. *Computational Economics*, 15(1):107--143.

Gambhir, S. & Gondaliya, N. (2012). A survey of associative classification algorithms. Em *International Journal of Engineering Research and Technology*, volume 1. ESRSA Publications.

Geng, L. & Hamilton, H. J. (2006). Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)*, 38(3):9.

Golbeck, J.; Robles, C. & Turner, K. (2011). Predicting personality with social media. Em *CHI'11 extended abstracts on human factors in computing systems*, pp. 253--262. ACM.

Han, J.; Pei, J. & Yin, Y. (2000). Mining frequent patterns without candidate generation. Em *ACM Sigmod Record*, volume 29, pp. 1--12. ACM.

Hansen, L. K. & Salamon, P. (1990). Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993--1001.

Jiang, L. & Guo, Y. (2005). Learning lazy naive bayesian classifiers for ranking. Em *17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)*, pp. 5--pp. IEEE.

Kononenko, I. (2001). Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in medicine*, 23(1):89--109.

Li, J.; Dong, G.; Ramamohanarao, K. & Wong, L. (2004). Deeps: A new instance-based lazy discovery and classification system. *Machine learning*, 54(2):99--124.

Li, W.; Han, J. & Pei, J. (2001). Cmar: Accurate and efficient classification based on multiple class-association rules. Em *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pp. 369--376. IEEE.

Liu, B.; Ma, Y. & Wong, C.-K. (2001). Classification using association rules: weaknesses and enhancements. Em *Data mining for scientific and engineering applications*, pp. 591--605. Springer.

Ma, B. L. W. H. Y. (1998). Integrating classification and association rule mining. Em *Proceedings of the fourth international conference on knowledge discovery and data mining*.

Machado, F. (2003). Cpar: Classification based on predictive association rules.

Moraes, C. & Abreu, C. (2006). A história das máquinas. *Abimaq/Sindimaq. Magma Editora Cultural, São Paulo*.

Pasquier, N.; Bastide, Y.; Taouil, R. & Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. Em *International Conference on Database Theory*, pp. 398--416. Springer.

Quinlan, J. R. (1993). C4.5: Programming for machine learning. *Morgan Kauffmann*, p. 38.

Quinlan, J. R. & Cameron-Jones, R. M. (1993). Foil: A midterm report. Em *European conference on machine learning*, pp. 1--20. Springer.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5(2):197–227. ISSN 0885-6125.

Schapire, R. E. (2003). The boosting approach to machine learning: An overview. Em *Nonlinear estimation and classification*, pp. 149--171. Springer.

Schapire, R. E. & Freund, Y. (2012). *Boosting: Foundations and algorithms*. MIT press.

Schapire, R. E. & Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297--336.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1--47.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 111--147.

Sun, Y.; Wang, Y. & Wong, A. K. (2006). Boosting an associative classifier. *IEEE Transactions on Knowledge and Data Engineering*, 18(7):988--992.

Thabtah, F. A.; Cowling, P. & Peng, Y. (2004). Mmac: A new multi-class, multi-label associative classification approach. Em *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pp. 217--224. IEEE.

Tieu, K. & Viola, P. (2004). Boosting image retrieval. *International Journal of Computer Vision*, 56(1-2):17--36.

Wang, J. & Karypis, G. (2005). Harmony: Efficiently mining the best rules for classification. Em *Proceedings of the 2005 SIAM International Conference on Data Mining*, pp. 205--216. SIAM.

Wang, Y. (1998). *High-order pattern discovery and analysis of discrete-valued data sets*. University of Waterloo.

Webb, G. I.; Boughton, J. R. & Wang, Z. (2005). Not so naive bayes: aggregating one-dependence estimators. *Machine learning*, 58(1):5--24.

Wettschereck, D.; Aha, D. W. & Mohri, T. (1997). A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, 11(1-5):273--314.

Anexo A

Conjuntos de Dados

ID	Origem	Título	Nome	Inst.	Atrib.	Classes	Bin.	Faltante
1	UCI	Annealing Database	anneal	798	38	6	83	S
2	UCI	Australian Credit Approval	australian	690	14	2	72	S
3	UCI	1985 Auto Imports	autos	205	26	7	262	S
4	UCI	Balance Scale Weight & Distance	balance	625	5	3	8	N
5	UCI	Wisconsin Breast Cancer	breast	699	10	2	18	S
6	UCI	Car Evaluation	car	1728	6	4	21	N
7	UCI	Contraceptive Method Choice	cmc	1473	10	3	43	N
8	UCI	Horse Colic database	colic	368	28	2	91	S
9	UCI	Credit Approval	credit-a	690	16	2	62	S
10	UCI	German Credit	credit-g	1000	20	2	85	N
11	UCI	Japanese Credit Screening	crx	125	16	2	52	S
12	UCI	Dermatology	dermatology	366	34	6	121	S
13	UCI	Pima Indians Diabetes	diabetes	768	8	2	85	N
14	UCI	Protein Localization Sites	ecoli	336	8	8	61	N
15	UCI	Solar Flare	flare	323	13	2	16	N
16	UCI	Glass Identification	glass	214	10	7	75	N
17	UCI	Haberman's Survival	haberman	306	4	2	20	N
18	UCI	Heart Disease Cleveland	heart-c	303	14	5	70	S
19	UCI	Heart Disease Hungarian	heart-h	294	14	5	54	S
20	UCI	Heart Statlog	heart-statlog	270	13	2	63	N
21	UCI	Hepatitis Domain	hepatitis	155	20	2	46	S
22	UCI	Thyroid Disease	hypothyroid	3772	30	4	36	S
23	UCI	Iris Plants	iris	150	5	3	12	N
24	UCI	Final Settlements In Labor Negotitions	labor	57	16	2	36	N
25	UCI	BUPA liver disorders	liver	345	7	2	33	N
26	UCI	Lymphography Domain	lymphography	148	19	4	50	N
27	UCI	Mushroom Database	mushroom	8124	22	2	88	S
28	UCI	Postoperative Patient Data	postoperative	90	9	3	23	S
29	UCI	Primary Tumor Domain	primary-tumor	339	18	22	37	S
30	UCI	Image Segmentation Data	segment	2310	19	7	70	S
31	UCI	Thyroid Records	sick	3772	30	2	39	S
32	UCI	Teaching Assistant Evaluation	tae	151	6	3	15	N
33	UCI	Tic-Tac-Toe Endgame	tictactoe	958	9	2	27	N
34	UCI	Vehicle Silhouettes	vehicle	946	18	4	117	N
35	UCI	1984 United States Congress Voting	vote	435	16	2	32	S
36	UCI	Wine Recognition	wine	178	13	3	37	N
37	UCI	Zoo database	zoo	101	18	7	33	N
38	Kaggle	Twitter Sentiment Analysis	twt	7086	-	2	2173	N

Tabela A.1: Informações gerais sobre os conjuntos de dados.

Anexo B

Propriedades matemáticas

B.1 Propriedades dos radicais

Abaixo estão algumas propriedades matemáticas usadas neste trabalho.

$$\sqrt[n]{a^m} = a^{\frac{m}{n}} \tag{B.1}$$

$$\frac{1}{\sqrt{\frac{a}{b}}} = \sqrt{\frac{b}{a}} \tag{B.2}$$

$$\frac{1}{2}\log a = \log a^{\frac{1}{2}} = \log \sqrt{a} \tag{B.3}$$

$$-\frac{1}{2}\log a = \log a^{-\frac{1}{2}} = \log \frac{1}{\sqrt{a}} \tag{B.4}$$