Universidade Federal de Pernambuco
Centro de Informática

Graduação em Ciência da Computação

Uma Abordagem para Gerar Justificativas Compreensíveis para Escores Produzidos por Redes Neurais

Christian D. A. Daniel

Trabalho de Graduação

Recife
01 de Dezembro de 2008

Universidade Federal de Pernambuco
Centro de Informática

Christian D. A. Daniel

Uma Abordagem para Gerar Justificativas Compreensíveis para Escores Produzidos por Redes Neurais

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Prof. Dr. Paulo J. L. Adeodato*

Recife
01 de Dezembro de 2008

Aos meus pais

Agradecimentos

À NeuroTech™ por ter me apresentado um problema a ser resolvido, e ter disponibilizado todos os subsídios e fontes de dados que foram necessárias para a realização desse trabalho. Freqüentemente suportando meus atrasos, faltas e sono durante o expediente de trabalho.

Aos meus pais que sempre me incentivaram e me deram suporte a todas as minhas escolhas (por mais incoerentes que fossem).

Aos meus colegas-amigos, que tornaram o curso de graduação muito mais agradável que realmente é. Em especial: Arthur, André Ricardo, André Vitor e Icamaan; que, além disso, tornaram possível a realização desse trabalho, cada um do seu jeito.

Aos meus amigos que me aturaram, sempre me fazendo espairecer quando necessário. E, em especial, Denise e Bruno que sempre pararam para escutar, demonstrando interesse, em tudo que eu fazia de novo em relação a este trabalho, sempre, assim, me incentivando a continuar.

E, especialmente, a Nathalie que, apesar de estar distante, conseguiu me obrigar, todo o dia, a continuar este trabalho até o fim, a fim de que eu consiga atingir meus grandes objetivos num futuro próximo.

"Nunca ande pelo caminho traçado, pois ele conduz somente aonde outros já foram."

—ALEXANDER GRAHAM BELL

Resumo

Redes Neurais (RN) é uma das abordagens mais comuns e robustas para sistemas de classificação não-lineares. A instância mais utilizada dessa abordagem é a Multi-Layer Perceptron (MLP), pois as suas hipóteses induzidas possuem um alto grau de generalização em relação a outros algoritmos. Entretanto, MLPs também estão entre as abordagens que produzem hipóteses difíceis de compreender, pois cada nodo representa uma combinação não-linear dos nodos da camada anterior. Uma característica importante de MLP's é o fato de seu output ser composto de escores, indicando a pontuação que cada classe recebeu da rede. É importante notar que se pode usar o fato de uma MLP dar uma pontuação, e não só determinar de que classe pertence o exemplo de entrada, para auxiliar no entendimento da decisão da RN.

Freqüentemente, estatísticos e especialistas no domínio da aplicação da RN questionam a decisão tomada pela solução. Quando isso acontece, existem poucos argumentos para convencer a esses profissionais da consistência desse resultado, ou mesmo explicar-lhes o porquê desse escore.

O foco deste trabalho é criar uma nova abordagem que possibilite a extração de informações (estatísticas e regras) da solução atual (RN) que sejam necessárias e suficientes para explicar um escore gerado.

Como proposta, devemos extrair regras da RN em conjunto com os exemplos de treinamento, e, após isso, aplicá-las aos exemplos cujos escores desejam ser justificados.

Palavras-chave: Redes Neurais, Mineração de Dados, Extração de Regras.

Abstract

Neural Networks (NNs) is one of most common and robust approaches for non-linear classifiers. Multi-Layer Perceptrons (MLPs) are often treated as the most used architecture of NNs, because their inductive hypotheses have a high degree of generalization comparing to other approaches. Although, MLPs are also under the set of hard-to-understand hypotheses, since each node represents a non-linear combination of the node outputs in the layer before.

A very important feature of MLPs is the fact of their outputs are composed by real-numbered values. By the means of understanding NNs decisions, it's important to notice that we can exploit the MLPs feature of outputting scores and not only the classifying results.

Statisticians and domain specialists sometimes doubt (or even disagree) the decision that NNs had taken. When this happens there are not sufficient arguments that convince these professionals of the results reliability, or the meaningfulness of scoring.

This work focuses is to create a new approach capable of extract information of the current solution (NN). That information must be necessary and sufficient to explain an outputted score.

Our propose is to extract rules from the NN by the training set and, after that, apply these rules to the examples whose the score must be justified.

Keywords: Neural Networks, Data Mining, Rule Extraction, Knowledge Discovery.

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

Uma das principais características da inteligência é a aprendizagem a partir de exemplos, e nas últimas duas décadas têm tido estudos frutíferos em inteligência artificial, estatística, ciência cognitiva e áreas correlatas. Foram desenvolvidos algoritmos que são capazes de aprender indutivamente a partir de exemplos. A aprendizagem indutiva é geralmente utilizada para duas metas principais: resultados e descoberta.

No primeiro caso, a meta é induzir um modelo que possa ser utilizado para resolver uma tarefa de interesse. Outra utilização para a aprendizagem indutiva é adquirir conhecimento sobre dados, construindo um modelo descritivo sobre eles. Mas, para isso, deve-se utilizar modelos que sejam humanamente compreensíveis e possam levar a um melhor entendimento do domínio do problema.

Aprendizagem indutiva com foco na compreensibilidade é uma das principais atividades nas áreas de aquisição de conhecimento em bases de dados e mineração de dados. Claro, um problema pode requerer tanto que uma tarefa seja cumprida e que algum conhecimento compreensível seja gerado a partir dos dados.

1.1 Contexto

Atualmente a utilização de modelos complexos, como redes neurais, para a tarefa de classificação tem se tornado mais corriqueira. A maioria desses modelos processa a entrada de forma não-linear e, aos olhos humanos, de forma às vezes imprevisível. Em geral, o resultado desses modelos tem uma representação numérica, que neste trabalho chamaremos de escore.

Freqüentemente, estatísticos e especialistas no domínio da aplicação da RN questionam a decisão tomada por um modelo complexo, especialmente em modelos baseados em redes

neurais. Quando isso acontece, existem poucos argumentos para convencer a esses profissionais da consistência desse resultado, ou mesmo explicar-lhes o porquê desse escore.

Intuitivamente, a abordagem mais direta para se entender um domínio é a inferência de regras lógicas. Este trabalho busca, então, extrair regras de modelos que são de difícil compreensão humana.

Existem diversas abordagens que tentam extrair regras de modelos complexos, a grande maioria, entretanto, é baseada na "forma" do modelo. Quando em redes neurais, baseia-se em sua arquitetura, topologia, funções de ativação, método de treinamento, etc. Quando num modelo estatístico, pode-se basear na forma de estimação das probabilidades, nas probabilidades condicionais, nas implicações, etc.

1.2 Objetivos

O foco deste trabalho é criar uma nova abordagem que possibilite a extração de informações (regras) de um modelo existente (RN) que sejam necessárias e suficientes para explicar um escore gerado. Essa abordagem deve poder induzir justificativas compreensíveis por seres humanos para escores gerados por redes neurais para exemplos reais de classificação.

Existem certas características que devem ser buscadas por qualquer algoritmos que extrai representações simbólicas de uma massa de dados:

- **Compreensibilidade (Interpretabilidade):** a representação simbólica deve ser facilmente compreensível por especialistas no domínio;
- **Fidelidade:** a representação deve ser eficaz quanto à aproximação do modelo;
- **Escalabilidade:** a abordagem deve ser dimensionável para modelos com muitas entradas e um processo intermediário com muita informação (e.g. redes neurais com muitos nós e muitas conexões);
- **Generalidade:** a abordagem deve atender o maior número possível de modelos. Por exemplo, uma abordagem para redes neurais (o que já é uma perda de generalidade) deve atender a arquiteturas genéricas e não deve demandar um regime especial de treinamento.

OBS.: o foco desse trabalho é uma abordagem para justificar escores de redes neurais, mas, como será visto posteriormente, o modelo não é mandatório podendo ser qualquer outro que possua um resultado real e contínuo.

1.3 Organização

Nos próximos capítulos serão mostradas breves introduções sobre Redes Neurais (capítulo 2) e Extração de Regras (capítulo 3), que são os fundamentos principais sobre a abordagem a ser apresentada. O capítulo 2 segue para mostrar como um modelo pode ser difícil de interpretar para um humano, exemplificando com as redes neurais. O capitulo 3 apresenta os fundamentos da extração de regras, métodos e problemas, apenas o necessário para o entendimento deste trabalho.

O capítulo 4 é a descrição detalhada da abordagem, por um algoritmo iterativo que exibe resultados de fácil compreensão. O capítulo 5 mostra alguns resultados da utilização da nossa nova abordagem. O capítulo 6 finaliza esta monografia numa discussão sobre a contribuição da mesma, limitações da abordagem apresentada e possíveis trabalhos futuros.

CAPÍTULO 2

Redes Neurais

Redes Neurais é uma das abordagens mais utilizadas atualmente para problemas de classificação, por ser robusta e conseguir aproximar qualquer tipo de função. Esta abordagem foi bem sucedida na criação de modelos para muitos dos problemas mais complexos do mundo real, tais como reconhecimento de imagens e caracteres, reconhecimento de voz, análise de crédito, etc [RWL94].

Neste capítulo, é apresentada uma contextualização histórica das Redes Neurais Artificiais, seguida de uma explicação simples e construtivista sobre Perceptrons de Múltiplas Camadas, finalizando com as conclusões relevantes para este trabalho.

2.1 Histórico

O campo de estudos das redes neurais artificiais teve inicio em 1943, quando Warren S. McCulloch e Walter Pitts publicaram seu conhecido trabalho "*A logical calculus of the ideas immanent in nervous activity*", que consistia numa rede neural simples modelada com circuitos elétricos. Em sua tese, McCulloch e Pitts mostraram que era possível construir uma rede neural apenas com matemática e algoritmos. Assim, iniciou-se um segmento completamente novo na pesquisa computacional e inteligência artificial. [MP43]

O conceito de treinamento de uma rede neural foi introduzido por Hebb em 1949, quando publicou o livro "*The organization of behavior*". Em 1962, Frank Rosenblatt apresentou uma abordagem para treinar uma rede de McCulloch e Pitts, tornou isso possível se utilizando da intensidade das sinapses dos neurônios. Para treinar a rede, Rosenblatt muda um pouco a intensidade das sinapses cada vez que a rede retorna um resultado errado. A forma de treinamento de redes neurais é uma área que ainda hoje é muito pesquisada. [Ros62]

Quando o campo estava em evidência, — pois agora era possível construir grandes redes e atribuí-las problemas que eram virtualmente impossíveis na programação convencional — Marvin Minsky e Seymour Papert mostraram em sua tese, em 1969, que a rede neural era incapaz de resolver a operação XOR, mais tarde superado pelo conceito de camadas escondidas. Minsky e Papert mostraram também que o aumento do número de entradas de uma rede neural aumentaria exponencialmente o tempo de treinamento da mesma, limitando a eficiência da rede. [MP69]

A descoberta desses problemas desacelerou a onda de pesquisas sobre redes neurais. Nesse tempo, o poder computacional cresceu, assim as redes neurais poderiam resolver mais problemas que antes. A dificuldade em treinar uma rede com camadas escondidas foi solucionado eficientemente com a introdução do algoritmo de back-propagation (i.e. retro-propagação), por Werbos em 1974. [Wer74]

Apesar dos computadores terem alcançado velocidades centenas de vezes maiores desde o início da pesquisa, ainda existem problemas com o treinamento quando um valor exacerbado de entradas é adicionado. O aumento de performance de Redes Neurais Artificiais assim como a criação de novos modelos das mesmas continuam sendo uma área forte de pesquisa.

2.2 Perceptrons

O perceptron é a forma mais simples de uma rede neural, pois é constituído de um único neurônio com os pesos das entradas ajustáveis e um *bias* (viés). O funcionamento de um perceptron é simples: cada atributo do exemplo de entrada corresponde a uma conexão de entrada do perceptron e cada conexão de entrada possui um peso; então o neurônio faz uma soma ponderada das conexões de entrada. Assim, o bias é uma constante que é sempre adicionada à entrada. Na verdade, antes de passar o resultado para a saída a soma passa por uma função de ativação, que modela a saída real em relação ao resultado da soma. Vide figura 2.1.

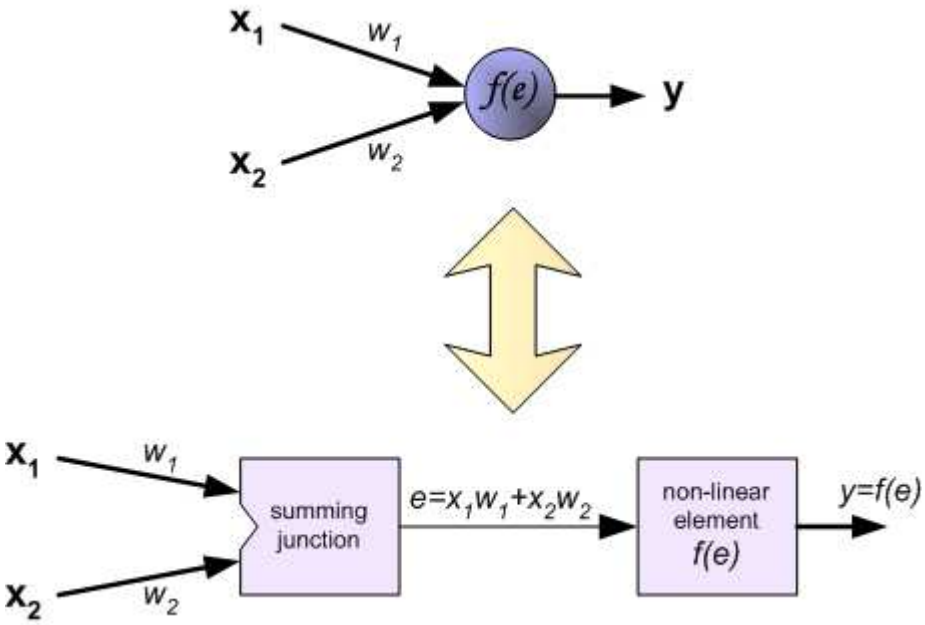

Figura 2.1 MLP — Funcionamento de um Neurônio

2.2.1 Funções de Ativação

A função de ativação (representada por $f(e)$ na figura 2.1), define a saída do neurônio, limitando-a e dando a forma adequada ao problema. Pode-se identificar três tipos básicos de funções de ativação:

2.2.1.1 Função Limiar (Degrau)

Uma função degrau é uma função não contínua que possui valores discretos distintos dependendo da entrada. Vide equação 2.1 e figura 2.2.

$$f(e) = \begin{cases} 1 & \text{se } e \geq 0 \\ 0 & \text{se } e < 0 \end{cases} \tag{2.1}$$

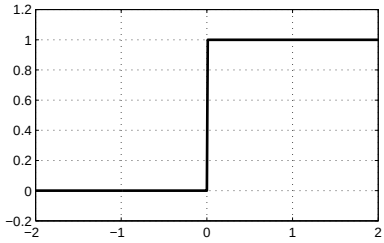

Figura 2.2 Função Limiar

2.2.1.2 Função Linear por Partes

Uma função linear por partes é uma função que é linear até atingir os valores de saturação (valores máximos e mínimos de saída), quando a saída se torna constante. A função linear por partes é contínua mas não diferenciável. Vide equação 2.2 e figura 2.3.

$$f(e) = \begin{cases} 1 & \text{se } e \geq +1 \\ e & \text{se } -1 > e > +1 \\ 0 & \text{se } e \leq -1 \end{cases} \tag{2.2}$$

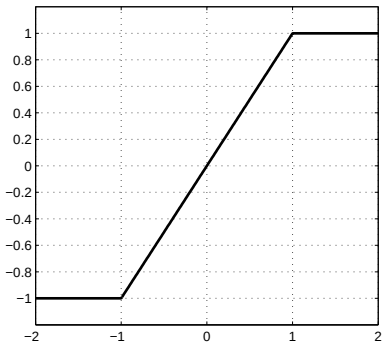

Figura 2.3 Função Linear por Partes

2.2.1.3 Função Sigmóide

Uma função sigmóide é uma função que possui um gráfico em forma de 'S', e é a forma mais utilizada de função de ativação em redes neurais. Uma função sigmóide é uma função contínua diferenciável monotônica estritamente crescente que possui um bom balanceamento

entre as características de funções de ativação lineares e não-lineares. As duas funções sigmóides mais comuns são a função logística (equação 2.3 e figura 2.4) e a função tangente hiperbólica (equação 2.4 e figura 2.5). Essas funções possuem como parâmetro também um valor de inclinação a.

$$f(e) = \frac{1}{1+\exp(-ae)} \tag{2.3}$$

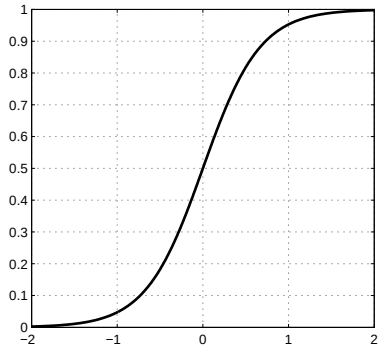

Figura 2.4 Função Logística — $a = 3$

$$f(e) = \frac{1-\exp(-ae)}{1+\exp(-ae)} = \tanh\left(\frac{ae}{2}\right) \tag{2.4}$$

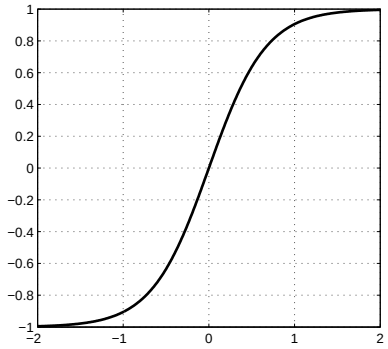

Figura 2.5 Função Tangente Hiperbólica — $a = 3$

2.3 Perceptrons de Múltiplas Camadas

O perceptron é uma estrutura muito simples e pode apenas modelar problemas linearmente separáveis, isso exclui os problemas mais complexos (inclusive o problema do XOR). Para resolver isso, utiliza-se perceptrons encadeados, em forma de rede — normalmente representados como grafos ponderados dirigidos, onde cada nodo é um perceptron e as arestas são suas conexões. A estrutura mais comum é a divisão dos perceptrons em camadas, denominada Perceptrons de Múltiplas Camadas (*Multi-Layer Perceptrons* — MLP), que é capaz de generalizar qualquer função [Cyb89].

A figura 2.6 mostra um exemplo de uma MLP com duas camadas escondidas e uma camada de saída. Em geral (como no exemplo) tratamos de redes completamente conectadas (i.e. cada nodo se conecta com todos os nodos da camada seguinte); além disso na MLP, o fluxo da informação (sinais/sinapses) é sempre para frente (no exemplo, da esquerda para a direita).

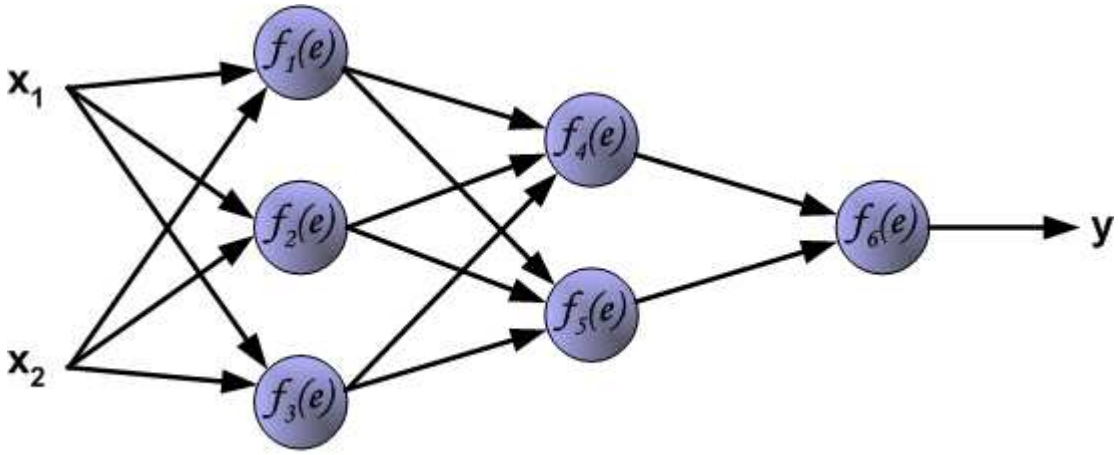

Figura 2.6 MLP

Com uma rede já construída, o seu funcionamento é simples: ao se colocar um exemplo na camada de entrada, a primeira camada da rede recebe as estradas pra cada neurônio; depois de computar o resultado de cada neurônio dessa camada, o mesmo se faz com a próxima camada (pois a saída da camada anterior é tratada como sinal de entrada para a camada seguinte); isso acontece iterativamente até termos o resultado da camada de saída. Vide esquemático nas figuras 2.7, 2.8 e 2.9.

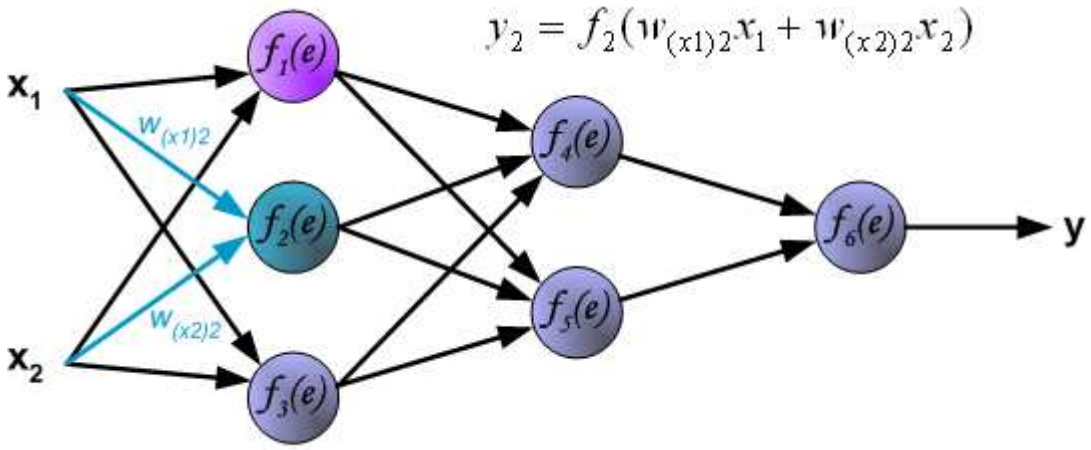

Figura 2.7 MLP — Entrada — Camada Escondida 1

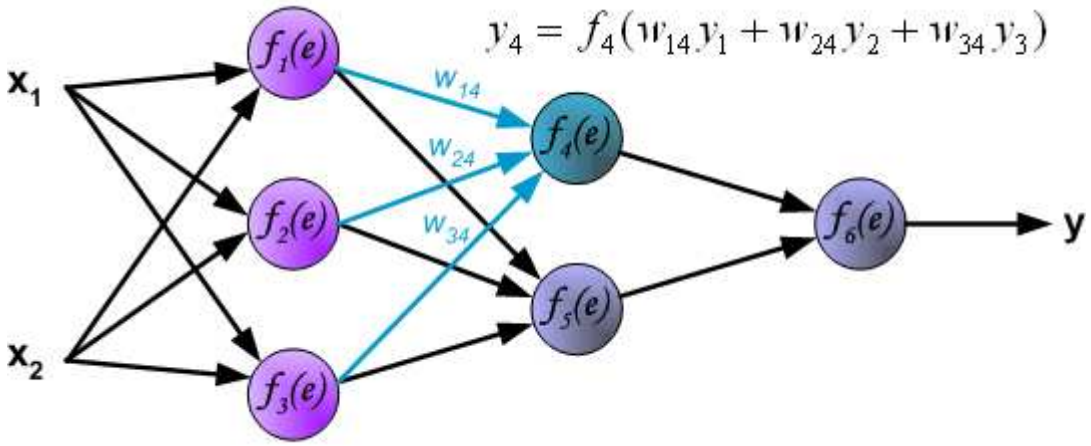

Figura 2.8 MLP — Entrada — Camada Escondida 2

2.4 Conclusões Relevantes

Como pode ser notado, um modelo baseado em redes neurais é de difícil compreensão humana, ele pode possuir várias camadas com muitos nodos, onde cada nodo possui diversas entradas combinadas que ainda passam por uma função de ativação. Extrair conhecimento diretamente dessa estrutura é uma tarefa não trivial, que ainda hoje não possui uma metodologia genérica que garanta um bom desempenho nos quesitos compreensibilidade, escalabilidade e fidelidade.

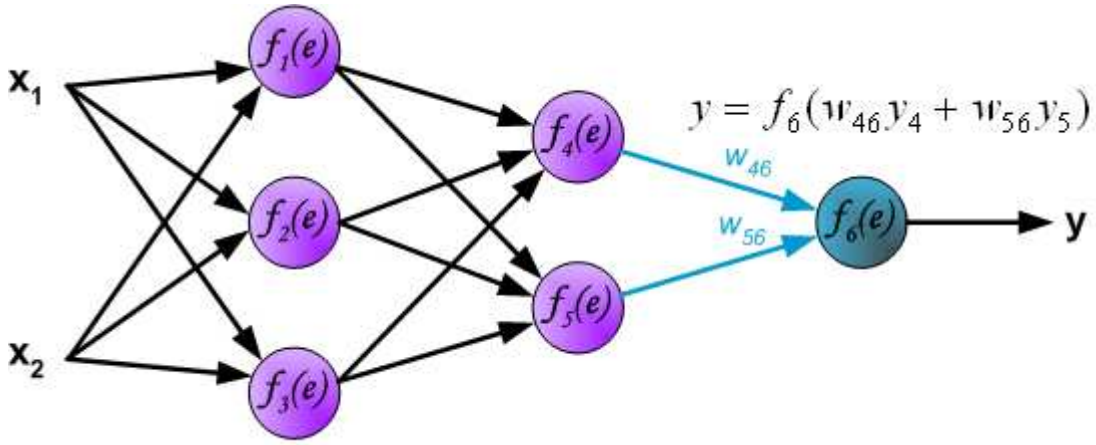

Figura 2.9 MLP — Entrada — Camada de Saída

CAPÍTULO 3

Extração de Regras

Para entender um modelo representado por uma rede neural, podemos tentar traduzi-lo para outro modelo em uma linguagem mais compreensível. O nome "extração de regras" aparece devido ao fato de que a indução de regras é realizada com o intuito de traduzir um outro modelo para a linguagem de regras.

Existem várias abordagens que tentam extrair regras diretamente da estrutura da rede neural, se utilizando das saídas possíveis dependendo dos pesos e da função de transferência. Algumas delas se utilizam de métodos de busca global com heurísticas [SN88, Thr95], outras se utilizam de métodos de busca local para procurar regras conjuntivas.

A maior limitação desses métodos é que eles costumam pecar na generalização e/ou na escalabilidade. Além disso, a grande maioria deles depende da arquitetura da rede neural e da função de transferência empregada na rede, ou seja, fazem exigências quanto à estrutura da rede.

Este capítulo trata de uma breve introdução à extração de regra de forma simples, com o objetivo de auxiliar no entendimento da metodologia deste trabalho. O capítulo cobre algumas medidas utilizadas para comparação de regras, depois como extrair regras com atributos quantitativos (i.e. numéricos), então uma breve introdução sobre árvores de decisão é apresentada e, após, o capítulo é concluído com um esclarecimento das limitações da utilização de regras como classificadores.

3.1 Medidas de Interesse (Interestingness)

3.1.1 Suporte (*Support*)

Suporte (também conhecido como cobertura — *coverage*) é a métrica que indica a abrangência de uma regra em relação aos exemplos da base de dados. É a percentagem de exemplos na base, que obedecem a uma dada regra.

$$support(A \Rightarrow B) = P(A \cap B) \tag{3.1}$$

3.1.2 Confiança (*Confidence*)

Confiança (também conhecida como precisão — *accuracy*) é a métrica que indica a taxa de acerto de uma regra. É a percentagem de exemplos que obedecem à regra em relação aos que obedecem apenas as condições da mesma.

$$\text{confidence}(A \Rightarrow B) = P(B|A) = \frac{P(A \cap B)}{P(A)} = \frac{support(A \Rightarrow B)}{P(A)} \tag{3.2}$$

3.2 Minerando Regras de Associação Quantitativas

Quando desejamos minerar regras de associação podemos (e vamos) nos deparar com atributos quantitativos (numéricos), que não são originalmente divididos em categorias. Atributos numéricos podem possuir uma quantidade exacerbada de valores distintos então, se gerássemos uma regra para cada valor distinto de um atributo numérico teríamos um número muito grande de regras com cobertura ínfima.

Regras de associação quantitativas são dinamicamente discretizadas durante a mineração dos dados por um processo chamado *Binning*. Binning consiste em particionar o espaço da variável numérica em faixas de valores (*bins*), então as estratégias de binning definem apenas

em quais pontos devem ser separadas essas faixas:

- **Faixas Eqüidistantes:** as faixas têm o mesmo tamanho;
- **Faixas Eqüifrenquentes:** as faixas comportam um número aproximado de exemplos;
- **Faixas Baseadas em Agrupamento:** as faixas são computadas executando um algoritmo de agrupamento (*clustering*), onde valores mais próximos são agrupados numa mesma faixa (baseado em várias medidas de distância).

3.3 Árvores de Decisão

Uma árvore de decisão é uma árvore utilizada para classificação de instâncias, cujos nodos não-terminais representam uma avaliação sobre algum atributo, e cada ramo a partir desse nodo representa um dos possíveis valores para esse atributo. E cada folha representa a decisão: a classificação a classificação da instância para uma das classes.

Então, para prever a classe de (classificar) uma instância, deve-se começar testando o atributo da raiz da árvore, e seguir o ramo que condiz com o atributo da instância, fazendo o mesmo com os próximos nós não terminais, até encontrar uma folha e essa afirma a classe a qual a instância pertence. O exemplo clássico de árvore de decisão é o problema de classificar se uma manhã de sábado é ou não propícia para se jogar tênis, o diagrama da árvore encontra-se na figura 3.1, e a base de dados utilizada para a criação desta árvore na tabela 3.1.

3.3.1 Aprendizagem por Árvores de Decisão

O algoritmo de contrução de uma árvore de decisão costuma ser uma busca gulosa (*greedy*) *top-down*, no espaço de possíveis árvores de decisão. O algoritmo base utilizado é o ID3 [Qui86]. O ID3 funciona iterativamente escolhendo o melhor atributo para classificar os exemplos da base de dados que pertencem ao ramo atual (na raiz, todos os exemplos), essa escolha é baseada em testes estatísticos.

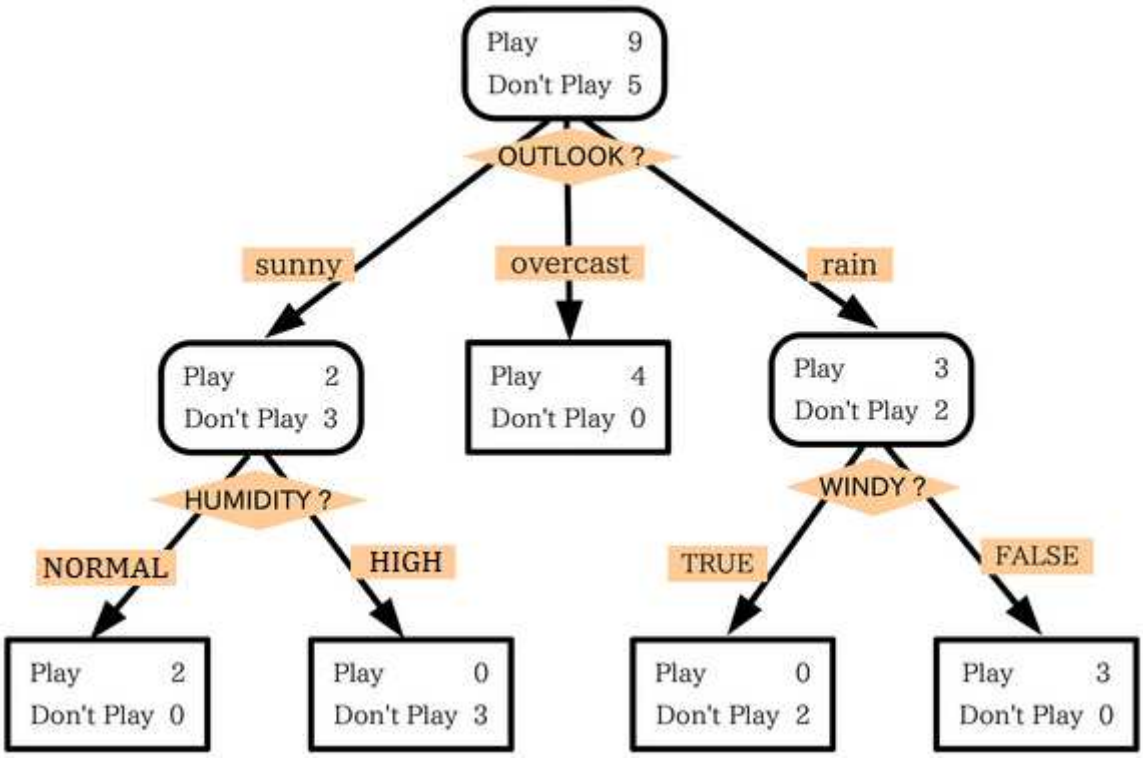

Figura 3.1 Exemplo de Árvore de Decisão — Jogar Tênis

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	Don't Play
D2	Sunny	Hot	High	Strong	Don't Play
D3	Overcast	Hot	High	Weak	Play
D4	Rain	Mild	High	Weak	Play
D5	Rain	Cool	Normal	Weak	Play
D6	Rain	Cool	Normal	Strong	Don't Play
D7	Overcast	Cool	Normal	Strong	Play
D8	Sunny	Mild	High	Weak	Don't Play
D9	Sunny	Cool	Normal	Weak	Play
D10	Rain	Mild	Normal	Weak	Play
D11	Sunny	Mild	Normal	Strong	Play
D12	Overcast	Mild	High	Strong	Play
D13	Overcast	Hot	Normal	Weak	Play
D14	Rain	Mild	High	Strong	Don't Play

Tabela 3.1 Dados para a Árvore da Figura 3.1 — Jogar Tênis

A principal escolha para algoritmo ID3 é selecionar a melhor heurística para a seleção de atributos, isto é escolher num dado nodo qual o atributo melhor separa as classes do problema.

Segue alguns testes para seleção de atributos.

3.3.1.1 Entropia e Ganho de Informação

A entropia mede a impureza (ou aleatoriedade) de um conjunto de dados, baseando-se no trabalho de Shannon sobre teoria da informação. Isso significa dizer que quanto menor a entropia maior a diferença das classes (em termos de quantidade) no conjunto de dados. A entropia pode ser quantizada como

$$Entropy(D) \equiv \sum_{i \in Classes} -p_i \log_2(p_i) \tag{3.3}$$

onde D é o conjunto de dados,e p_i é a proporção de D que pertence à classe i.

Para a escolha do melhor atributo deve-se escolher o atributo que melhor separa as classes, em relação à partição do espaço correspondente ao ramo atual. Ou seja, deve-se escolher o atributo que apresente o melhor ganho de informação sobre os dados do ramo (no caso da raiz, toda a base de dados). O ganho de informação, aqui, é definido como a diminuição da entropia e pode ser quantizado como

$$Gain(D,A) \equiv Entropy(D) - \sum_{v \in Values(A)} \frac{|D_v|}{|D|} Entropy(D_v) \tag{3.4}$$

onde D é a partição dos dados relativa ao ramo, A é o atributo que se deseja mensurar o ganho de informação (caso ele seja selecionado para particionar o espaço), $Values(A)$ é o conjunto de possíveis valores do atributo A, D_v é o conjunto de dados de D que possuem v como valor do atributo A.

Então o atributo escolhido para o nodo é simplesmente o atributo que apresente maior ganho de informação.

3.3.1.2 Índice de Gini

O índice de Gini mede a nível de impureza de um conjunto de dados como

$$Gini(D) = 1 - \sum_{i \in Classes} p_i^2 \tag{3.5}$$

onde D é o conjunto de dados,e p_i é a probabilidade de uma instância em D pertencer à classe i. p_i costuma ser estimado simplesmente como $\frac{|C_{i,D}|}{|D|}$.

O índice de Gini considera um particionamento binário de cada atributo, dividindo um atributo em dois subconjuntos do mesmo. Então, se um atributo possui n valores distintos, existem $2^{n-1} - 1$ partições possíveis. Por exemplo, no caso da Temperatura no exemplo da tabela 3.1 temos três valores distintos (Hot, Mild e Cold), resultando em três possíveis particionamentos: {Hot} e {Mild, Cold}; {Hot, Mild} e {Cold}; {Hot, Cold} e {Mild}.

Considerando o particionamento binário, computa-se a soma ponderada das partições (em relação ao tamanho delas) para se calcular o índice de Gini:

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2) \tag{3.6}$$

onde D é o conjunto de dados, D_1 e D_2 são a primeira e a segunda partição de D respectivamente, e $Gini_A(D)$ é o índice de Gini sobre o atributo A no conjunto de dados D (dado um particionamento binário de D como D_1 e D_2).

Para cada atributo, cada possível particionamento binário é considerado, e o particionamento com o menor índice de Gini é escolhido. Então o atributo (e, conseqüentemente, particionamento) a ser escolhido como nodo da árvore é o que possui a maior redução de impureza:

$$\Delta Gini(D,A) = Gini(D) - Gini_A(D) \tag{3.7}$$

3.3.2 Extração de Regras de Árvores de Decisão

Extrair regras de uma árvore de decisão consiste, em percorrer cada caminho da árvore da raiz até uma folha. Dessa forma, cada nodo não-terminal representa uma cláusula da regra, onde as

cláusulas são conectadas por conjunções (operador “E”) e a conseqüência da regra (operador “ENTÃO”) é a classe que a folha pertence.

Vale ressaltar que regras extraídas de uma árvore de decisão são mutuamente exclusivas e exaustivas. Serem mutuamente exclusivas implica que nenhum exemplo pode pertencer ao mesmo tempo a duas regras distintas. Serem exaustivas implica que todo exemplo pertence a pelo menos uma regra, e não é necessária uma regra padrão para o caso que um exemplo não encaixe em nenhuma regra.

Com essa metodologia, conseguimos extrair exatamente uma regra por folha. Contudo, a árvore gerada pode ser muito grande (número grande de nodos entre a raiz a as folhas), o que provoca regras complexas demais e muito específicas (baixo suporte), dificultando o objetivo de entendimento humano. Pode-se, então, podar (*prune*) as regras para que elas satisfaçam tais exigências. A poda funciona removendo qualquer antecedente ou precondição cuja remoção não prejudica a taxa de acerto (que não diminui a confiança).

3.4 Limitações

O principal dilema ao escolher a abordagem de classificação está entre a interpretablidade do modelo e a fidelidade do modelo para com o problema. Enquanto redes neurais possuem alta capacidade de fidelização ao problema (podendo generalizar qualquer função), elas são de difícil compreensão humana. Por outro lado, regras são de fácil compreensão humana, mas não conseguem ser fieis a qualquer problema mantendo a generalização e escalabilidade.

A principal limitação das regras é o fato do modelo ser uma conjunção de soluções lineares. Na verdade, um modelo baseado em regras é uma conjunção de hiper-cubos com arestas paralelas aos eixos, como está exemplificado na figura 3.2: então, para se aproximar do problema real (imagem da esquerda), a única solução direta (sem transformação dos dados) é criar mais hiper-cubos (imagem da direita) cada vez menores quando se aproxima das bordas.

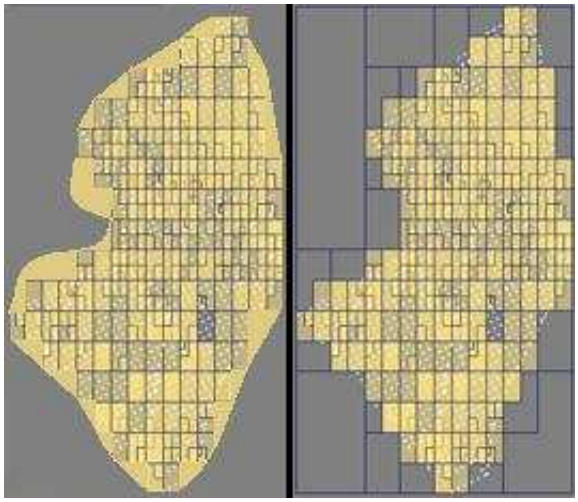

Figura 3.2 Exemplo de Limitação de Regras como Classificadores

CAPÍTULO 4

Abordagem

Enfim, este capítulo apresenta em detalhes como funciona a abordagem proposta para justificar resultados de modelos mais complexos. Cada seção do mesmo representa uma etapa do processo da abordagem, que devem ser seguidas em ordem: Pré-Processamento, Tratamento de Atributos, Seleção de Atributos, Construção das Árvores e Análise Iterativa de Regras.

A abordagem apresentada neste trabalho busca resolver dois problemas:

- existe uma consulta que apresentou um escore muito diferente do esperado pelo especialista no domínio. Nesse caso, a abordagem deve encontrar regras que justifiquem esse escore ou essa classificação como Bom ou Mau;
- existem duas consultas cujos valores de escores eram esperados como valores próximos, mas foram preditos pelo modelo atual como valores muito diferentes. Nesse caso, a abordagem deve encontrar regras que diferenciem as duas consultas, não só isso, mas também diferenciando pela diferença de escore (com predições piores para a de mais baixo escore e melhores para de mais alto).

4.1 Pré-Processamento

Pré-processamento de dados é definido como qualquer tipo de processamento realizado dos dados brutos para prepará-los para outro procedimento de processamento. É utilizado como uma prática preliminar de mineração de dados, transformando os dados num formato que seja mais efetivo e mais facilmente processado por um modelo. Esta fase pode ser constituída de várias técnicas que tratam de casos distintos nos dados, por exemplo: tratar valores ausentes (*missing values*), remoção de ruídos nos dados (*denoising*), selecionar uma amostra representativa dos dados (*sampling*), separar variáveis categóricas em atributos distintos, normalização

de variáveis numéricas, etc.

Como cada modelo possui suas próprias características, normalmente o pré-processamento é orientado ao modelo (*model driven*), dessa forma realiza-se um pré-processamento diferente para cada abordagem. Mas como a tentativa da abordagem é de assemelhar o comportamento da rede neural (ou de outro modelo qualquer), nada mais natural que se aproveitar do pré-processamento já existente para o modelo atual para a abordagem de justificativas.

4.2 Tratamento de Atributos

Quanto aos campos categóricos, a abordagem assume que o pré-processamento do modelo separa as categorias em vários atributos como atributos binários, pois esse é o caso para grande parte dos modelos. Então, esses atributos continuam sendo considerados binários.

Quanto aos campos numéricos, devemos transformá-los por meio de binning. Como explicado na seção 3.2 , existem algumas formas de categorizar variáveis numéricas. Neste trabalho utilizamos Binning com faixas eqüidistantes (com três e com cinco faixas) e com faixas baseadas em agrupamento (também com três e com cinco faixas).

No caso das faixas baseadas em agrupamento é utilizado o algoritmo clássico K-means, com K igual ao número de faixas desejadas. Vale lembrar que o algoritmo é utilizado unidimensionalmente, isto é, é utilizado em uma variável por vez, sem a influência de outras variáveis nem da variável *a posteriori*.

4.3 Seleção de Atributos

Quando um modelo é construído, deve-se filtrar os dados para que se trabalhe apenas com os dados mais importantes e passíveis de serem utilizados pelo modelo, evitando assim problemas de generalização do modelo — essa é uma tarefa que costuma ser atribuída ao pré-processamento. Todavia, o pré-processamento desta abordagem já está pronto e nada foi realizado quanto à remoção de atributos não interessantes ao problema de diferenciar consultas.

Existem duas opções para a utilização desta abordagem: a primeira é a tentativa de explicar o porquê do resultado de uma consulta única; a segunda é tentar diferenciar duas consultas distintas. Na primeira, não há muito que fazer quanto à seleção de atributos, pois utilizamos todos eles e esperamos que o modelo da abordagem escolha os mais significativos.

Na segunda opção partimos da hipótese que a interação entre variáveis não é tão importante para o modelo da abordagem ou da hipótese que mesmo sendo importante, o modelo da abordagem não conseguiria usufruir bem da característica de interação entre as variáveis. Assim, é possível descartar todas as variáveis que não possuem diferenças depois da etapa de tratamento de atributos — isto é, apenas as variáveis que estão em faixas diferentes nas duas consultas são mantidas.

4.4 Construção das Árvores

Tendo a lista de atributos selecionados para a construção da árvore, falta apenas escolher o teste para formação da árvore e a variável alvo. O teste escolhido foi o índice de Gini, pois a entropia tende a formar árvores maiores e com mais ramos. Ambas as variáveis Score (escore, numérica) e Status (Bom/Mau, binária) foram mantidas, isso faz com que sejam criadas duas árvores para cada combinação de dados que já tínhamos — ao total temos oito árvores: (Eqüidistante ou Agregado) $*$ (três ou cinco faixas) $*$ (Score ou Staus).

4.5 Análise Iterativa de Regras

Assim que as árvores são criadas, devemos verificar se existem regras que "predizem" as instâncias das consultas da forma que desejamos, a classificação pode ser realizada nas oito árvores de forma indistinta. Se uma árvore não produzir uma classificação interessante para o nosso problema —ou seja, produzir um resultado diferente do desejado ou o mesmo resultado para duas consultas, no problema de diferenciar duas consultas —devemos tratar a árvore, numa tentativa de recuperar informação relevante da mesma.

A solução encontrada para o tratamento da árvore, para que ela seja esmiuçada antes de

perdida, foi a exclusão iterativa dos nós raiz. Logo, quando uma árvore não apresenta bons resultados, a variável que corresponde à raiz dela é excluída da lista de seleção de atributos para aquela árvore, então uma nova árvore é construída baseada nas variáveis que restaram. Esse processo é realizado até que tenhamos regras suficientes pra justificar as consultas ou a árvore em questão não possa ser mais re-construída (por falta de atributos que separe bem as classes).

Após isso, as regras são extraídas diretamente da árvore de decisão para a justificativa. Agora, o especialista do domínio possui argumentos suficientes para redigir ou explicar o porquê daquela escolha incomum para resultados das consultas.

CAPÍTULO 5

Resultados

Alguns testes foram realizados e dois deles foram escolhidos para exemplificar este capítulo de resultados. A base de dados e as consultas a serem justificadas foram extraídas da massa de dados de uma empresa de concessão de créditos com cheques de porte nacional, e o modelo atual (baseado em redes neurais) foi cedido pela NeuroTech™.

Ambas as justificativas são duplas (possuem duas consultas), com uma diferença razoável de escore entre elas. Cada consulta de uma mesma justificativa é correspondente a uma compra com cheque(s) de um mesmo emitente e a decisão que a rede neural toma é se o cheque deve ou não ser aceito (baseado na variável *a posteriori* Status: bom pagador ou mau pagador).

A primeira justificativa é um evento considerado de dificuldade moderada, isto é, um especialista no domínio do problema trabalha entre oito e dezesseis horas para estudar, compreender e criar essa justificativa. A segunda justificativa é considerada de dificuldade alta, ou seja, um especialista, em geral, não consegue reunir argumentos suficientes para explicar as consultas.

Para cada árvore de decisão gerada para a justificativa, são exibidas na tabela a seguir quantas remoções de raízes são necessárias para encontrar a primeira regra que explique e diferencie as duas consultas, como apresentado na seção 4.5. Para essa regra, são calculados o suporte (que mede a freqüência de exemplos que obedecem às precondições da regra) e a confiança (a freqüência que esta regra acerta a variável alvo).

Existem árvores que mesmo depois das remoções de raízes não é possível encontrar nenhuma regra que satisfaça o problema (e ficam sem atributos candidatos a raízes), para esses casos a tabela é preenchida com o valor NA (*Not Avaliable*). As informações completas sobre as árvores estão disponíveis nos apêndices deste trabalho.

Segue a tabela 5.1, que é um sumário sobre a justificativa de dificuldade moderada supracitada, e a tabela 5.2, sumário sobre a justificativa de dificuldade alta:

Faixas	Alvo	Binning	Eliminações Necessárias	Suporte	Confiança
3 Faixas	Status	Equidistantes	2	0,04995	0,6797
		Agrupamento	0	0,2893	0,7128
	Score	Equidistantes	2	0,26195	0,78126
		Agrupamento	0	0,0483	0,5725
5 Faixas	Status	Equidistantes	0	0,42005	0,6837
		Agrupamento	0	0,4676	0,6637
	Score	Equidistantes	5	0,05115	0,41251
		Agrupamento	0	0,4676	0,41595

Tabela 5.1 Resultados de uma Justificativa — Dificuldade Moderada

Faixas	Alvo	Binning	Eliminações Necessárias	Suporte	Confiança
3 Faixas	Status	Equidistantes	NA	NA	NA
		Agrupamento	NA	NA	NA
	Score	Equidistantes	NA	NA	NA
		Agrupamento	NA	NA	NA
5 Faixas	Status	Equidistantes	1	0,18935	0,704
		Agrupamento	NA	NA	NA
	Score	Equidistantes	NA	NA	NA
		Agrupamento	1	0,7056	0,3585

Tabela 5.2 Resultados de uma Justificativa — Dificuldade Alta

Algumas características interessantes podem ser notadas:

- Em justificativas de dificuldade maior é difícil prever quais árvores terão boas regras a serem extraídas;
- em geral, a confiança é mais baixa quando a variável alvo é o escore — isso se dá devido à faixa de probabilidade dividir-se em mais faixas que quando o alvo é o status (que é binário);
- mesmo em justificativas as quais o especialista não consegue destrinchar, a abordagem consegue extrair algumas (poucas) regras que satisfaçam as exisgências o problema.
- a maior diferença, em relação à dificuldade de encontrar as regras, se dá devido ao Binning e não ao tipo de alvo escolhido.

CAPÍTULO 6

Conclusões

Redes neurais são uma das abordagens mais utilizadas para problemas de classificação, por possuir excelente desempenho e eficácia na predição em vários problemas de diferentes domínios. Entretanto, elas sofrem de uma limitação significativa, seu modelo é geralmente incompreensível por seres humanos. Para resolver essa limitação, muitos desenvolveram técnicas para a extração de regras de redes neurais. Em geral, a extração de regras é vista como uma tentativa de aproximar a função que a rede neural treinada representa, com representação simbólica específica.

A principal diferença deste pra outros trabalhos é que a abordagem é a mais generalizada possível, se afastando completamente da estrutura interna do modelo que gera o resultado duvidoso. Então, esta abordagem é constituída de uma extração de informação dos próprios dados, e essa informação é cruzada com o resultado do modelo atual para formar as justificativas.

Observamos resultados satisfatórios e a possibilidade de aumentar a eficiência do trabalho de profissionais em estatística e especialistas no domínio da aplicação. Faz-se necessária uma ressalva, pois esta metodologia possui limitações relativas ao ajuste da não-linearidade dos problemas.

6.1 Trabalhos Futuros

Existe um grande potencial teórico e prático nesta linha de pesquisa. Um projeto futuro poderia integrar uma bateria de testes comparativos dos conjuntos de regras extraídas mais ampla, no intuito de comprovar experimentalmente a eficácia e estabilidade do método. Poder-se-ia também, nesse *benchmark* nos utilizar de todo um universo de métricas sobre regras [HH99].

Além disso, existem diversos testes que podemos fazer para melhorar o desempenho da

abordagem, por exemplo quanto ao agrupamento podemos utilizar técnicas de agrupamento diferentes, que façam agrupamentos mais "inteligentes" (e.g. sem um número predeterminado de centros ou com agrupamentos hierárquicos); podemos fazer agrupamentos bidimensionais — utilizando as variáveis *a posteriori* para fazer grupos que dependam delas, e então esses centros seriam novamente projetados para a variável que se deseja agrupar; etc.

Além disso, uma pesquisa maior e mais específica na literatura poderia indicar uma perda de generalidade (aplicando a abordagem apenas às redes neurais) que compense em eficácia.

Poderíamos também nos utilizar das regras que não condizem com o resultado original do modelo, como uma tentativa de avaliar ou mesmo combater o modelo atual. Nesse caso talvez a ajuda de especialistas no domínio possa ser necessária para validar as regras extraídas, apesar de a base de dados já possuir informação suficiente para a argumentação em geral.

APÊNDICE A

Consulta Dupla — Dificuldade Moderada

```
> summary(dadosTreeStatusCut3)

Classification tree:
tree(formula = paste("STATUS_1  ", stringAttCut3, -STATUS_1
-STATUS_2 -SCORE"),
data = dadosCut3)
Variables actually used in tree construction:
[1] "QTD_DIAS_ULTIMO_CHEQUEQTD_DIAS_PRIMEIRO_CHEQUE"
Number of terminal nodes:  3
Residual mean deviance:  1.239 = 24790 / 20000
Misclassification error rate:  0.3432 = 6864 / 20000
```

```
> summary(dadosTreeScoreCut3)

Classification tree:
tree(formula = paste("SCORE  ", stringAttCut3, -STATUS_1
-STATUS_2 -SCORE"),
data = dadosCut3)
Variables actually used in tree construction:
[1] "QTD_DIAS_ULTIMO_CHEQUEQTD_DIAS_PRIMEIRO_CHEQUE"
Number of terminal nodes:  3
Residual mean deviance:  1.82 = 36400 / 20000
Misclassification error rate:  0.4041 = 8081 / 20000
```

```
> summary(dadosTreeStatusCut5)

Classification tree:
tree(formula = paste("STATUS_1  ", stringAttCut5, -STATUS_1
-STATUS_2 -SCORE"),
data = dadosCut5)
Variables actually used in tree construction:
[1] "PZ_MAX_ULT_CHEQUE_PRE"
Number of terminal nodes:  3
Residual mean deviance:  1.236 = 24720 / 20000
Misclassification error rate:  0.3442 = 6883 / 20000

> summary(dadosTreeScoreCut5)

Classification tree:
tree(formula = paste("SCORE  ", stringAttCut5, -STATUS_1
-STATUS_2 -SCORE"),
data = dadosCut5)
Variables actually used in tree construction:
[1] "PZ_MAX_ULT_CHEQUE_PRE"
Number of terminal nodes:  3
Residual mean deviance:  2.658 = 53140 / 20000
Misclassification error rate:  0.5078 = 10155 / 20000
```

```
> summary(dadosTreeStatusCluster3)

Classification tree:
tree(formula = paste("STATUS_1  ", stringAttCluster3,
-STATUS_1 -STATUS_2 -SCORE"),
data = dadosCluster3)
Variables actually used in tree construction:
[1] "QTD_DIAS_PRIMEIRO_CHEQUEQTD_PARCELAS"
[3] "VL_LIMITE_EMITENTE_FLEXIBILIZADO"
Number of terminal nodes:  5
Residual mean deviance:  1.224 = 24470 / 20000
Misclassification error rate:  0.3215 = 6429 / 20000

> summary(dadosTreeScoreCluster3)

Classification tree:
tree(formula = paste("SCORE  ", stringAttCluster3, -STATUS_1
-STATUS_2 -SCORE"),
data = dadosCluster3)
Variables actually used in tree construction:
[1] "QTD_DIAS_PRIMEIRO_CHEQUEQTD_PARCELAS"
[3] "VL_LIMITE_EMITENTE_FLEXIBILIZADO"
Number of terminal nodes:  5
Residual mean deviance:  1.833 = 36660 / 20000
Misclassification error rate:  0.4044 = 8089 / 20000
```

```
> summary(dadosTreeStatusCluster5)

Classification tree:
tree(formula = paste("STATUS_1  ", stringAttCluster5,
-STATUS_1 -STATUS_2 -SCORE"),
data = dadosCluster5)
Variables actually used in tree construction:
[1] "QTD_DIAS_ULTIMO_CHEQUEQTD_DIAS_PRIMEIRO_CHEQUE"
Number of terminal nodes:  3
Residual mean deviance:  1.246 = 24920 / 20000
Misclassification error rate:  0.3454 = 6908 / 20000

> summary(dadosTreeScoreCluster5)

Classification tree:
tree(formula = paste("SCORE  ", stringAttCluster5, -STATUS_1
-STATUS_2 -SCORE"),
data = dadosCluster5)
Variables actually used in tree construction:
[1] "QTD_DIAS_ULTIMO_CHEQUEQTD_DIAS_PRIMEIRO_CHEQUE"
Number of terminal nodes:  3
Residual mean deviance:  2.796 = 55910 / 20000
Misclassification error rate:  0.5562 = 11123 / 20000

[1] "################################################"
[1] "SCORE CONSULTA = 8SCORE CONSULTA = 71"
[1] "################################################"
```

```
> print(paste("SCORE = ", consultasCut3$SCORE));
[1] "SCORE = (-0.08,33.3]SCORE = (66.7,100]"
> print(dadosTreeStatusCut3)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 27730 1 ( 0.4985 0.5015 )
2) QTD_DIAS_ULTIMO_CHEQUE: (0.333,0.667],(0.667,1] 8230 10210 0
( 0.6887 0.3113 ) *
3) QTD_DIAS_ULTIMO_CHEQUE: (-0.001,0.333] 11770 15450 1 (
0.3655 0.6345 )
6) QTD_DIAS_PRIMEIRO_CHEQUE: (0.333,0.667] 6544 9063 1 ( 0.4812
0.5188 ) *
7) QTD_DIAS_PRIMEIRO_CHEQUE: (-0.001,0.333],(0.667,1] 5226 5516
1 ( 0.2206 0.7794 ) *
> print(dadosTreeScoreCut3)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 41330 (66.7,100] ( 0.39710 0.17315 0.42975 )
2) QTD_DIAS_ULTIMO_CHEQUE: (-0.001,0.333] 11770 22280
(66.7,100] ( 0.25421 0.15370 0.59210 )
4) QTD_DIAS_PRIMEIRO_CHEQUE: (-0.001,0.333] 5225 7034
(66.7,100] ( 0.12019 0.09722 0.78258 ) *
5) QTD_DIAS_PRIMEIRO_CHEQUE: (0.333,0.667],(0.667,1] 6545 13750
(66.7,100] ( 0.36119 0.19878 0.44003 ) *
3) QTD_DIAS_ULTIMO_CHEQUE: (0.333,0.667],(0.667,1] 8230 15610
(-0.08,33.3] ( 0.60146 0.20097 0.19757 ) *
```

```
> print(paste("SCORE = ", consultasCut5$SCORE));
[1] "SCORE = (-0.08,20]SCORE = (60,80.1]"
> print(dadosTreeStatusCut5)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 27730 1 ( 0.4985 0.5015 )
2) PZ_MAX_ULT_CHEQUE_PRE: (0.199,0.4],(0.4,0.6],(0.6,0.801],(0.801,1]
15468 20970 0 ( 0.5877 0.4123 )
4) PZ_MAX_ULT_CHEQUE_PRE: (0.4,0.6],(0.6,0.801],(0.801,1] 8401
10480 0 ( 0.6837 0.3163 ) *
5) PZ_MAX_ULT_CHEQUE_PRE: (0.199,0.4] 7067 9777 1 ( 0.4735
0.5265 ) *
3) PZ_MAX_ULT_CHEQUE_PRE: (-0.001,0.199] 4532 4461 1 ( 0.1942
0.8058 ) *
> print(dadosTreeScoreCut5)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 58940 (80.1,100] ( 0.31595 0.11790 0.09955
0.12580 0.34080 )
2) PZ_MAX_ULT_CHEQUE_PRE: (-0.001,0.199] 4532 8275 (80.1,100] (
0.07149 0.03839 0.04678 0.10260 0.74073 ) *
3) PZ_MAX_ULT_CHEQUE_PRE: (0.199,0.4],(0.4,0.6],(0.6,0.801],(0.801,1]
15468 46260 (-0.08,20] ( 0.38757 0.14119 0.11501 0.13260
0.22362 )
6) PZ_MAX_ULT_CHEQUE_PRE: (0.199,0.4] 7067 21320 (80.1,100] (
0.27239 0.11688 0.11349 0.15509 0.34215 ) *
7) PZ_MAX_ULT_CHEQUE_PRE: (0.4,0.6],(0.6,0.801],(0.801,1] 8401
23550 (-0.08,20] ( 0.48447 0.16165 0.11630 0.11368 0.12391 ) *
```

```
> print(paste("SCORE = ", consultasCluster3$SCORE));
[1] "SCORE = 3SCORE = 1"
> print(dadosTreeStatusCluster3)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 27730 1 ( 0.4985 0.5015 )
2) QTD_DIAS_PRIMEIRO_CHEQUE: 2,3 14214 19300 0 ( 0.5845 0.4155
)
4) QTD_PARCELAS: 1,3 3787 4320 0 ( 0.7425 0.2575 ) *
5) QTD_PARCELAS: 2 10427 14420 0 ( 0.5271 0.4729 )
10) VL_LIMITE_EMITENTE_FLEXIBILIZADO: 3 6418 8611 0 ( 0.6052
0.3948 ) *
11) VL_LIMITE_EMITENTE_FLEXIBILIZADO: 1,2 4009 5403 1 ( 0.4021
0.5979 ) *
3) QTD_DIAS_PRIMEIRO_CHEQUE: 1 5786 6939 1 ( 0.2872 0.7128 )
6) QTD_PARCELAS: 1,3 966 1206 0 ( 0.6832 0.3168 ) *
7) QTD_PARCELAS: 2 4820 4927 1 ( 0.2079 0.7921 ) *
> print(dadosTreeScoreCluster3)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 42570 2 ( 0.2147 0.4006 0.3847 )
2) QTD_DIAS_PRIMEIRO_CHEQUE: 1 5786 9820 2 ( 0.1400 0.6782
0.1818 )
4) QTD_PARCELAS: 1,3 966 1886 3 ( 0.2371 0.1905 0.5725 ) *
5) QTD_PARCELAS: 2 4820 6620 2 ( 0.1205 0.7759 0.1035 ) *
3) QTD_DIAS_PRIMEIRO_CHEQUE: 2,3 14214 30090 3 ( 0.2450 0.2876
0.4674 )
6) QTD_PARCELAS: 1,3 3787 6452 3 ( 0.2255 0.1109 0.6636 ) *
7) QTD_PARCELAS: 2 10427 22560 3 ( 0.2521 0.3518 0.3961 )
14) VL_LIMITE_EMITENTE_FLEXIBILIZADO: 1,2 4009 8117 2 ( 0.1926
0.5253 0.2821 ) *
15) VL_LIMITE_EMITENTE_FLEXIBILIZADO: 3 6418 13580 3 ( 0.2893
0.2434 0.4673 ) *
```

```
> print(paste("SCORE = ", consultasCluster5$SCORE));
[1] "SCORE = 3SCORE = 5"
> print(dadosTreeStatusCluster5)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 27730 1 ( 0.4985 0.5015 )
2) QTD_DIAS_ULTIMO_CHEQUE: 1,2,4,5 9352 11940 0 ( 0.6637
0.3363 ) *
3) QTD_DIAS_ULTIMO_CHEQUE: 3 10648 13830 1 ( 0.3534 0.6466 )
6) QTD_DIAS_PRIMEIRO_CHEQUE: 1,3,4 6513 8985 1 ( 0.4589 0.5411
) *
7) QTD_DIAS_PRIMEIRO_CHEQUE: 2,5 4135 3987 1 ( 0.1872 0.8128 )
*
> print(dadosTreeScoreCluster5)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 61500 1 ( 0.30445 0.14240 0.28135 0.12165
0.15015 )
2) QTD_DIAS_ULTIMO_CHEQUE: 1,2,4,5 9352 27660 3 ( 0.11784
0.19087 0.41595 0.14115 0.13420 ) *
3) QTD_DIAS_ULTIMO_CHEQUE: 3 10648 30110 1 ( 0.46835 0.09983
0.16313 0.10453 0.16416 )
6) QTD_DIAS_PRIMEIRO_CHEQUE: 1,2,3,4 6514 20300 1 ( 0.31056
0.13770 0.22797 0.13417 0.18959 ) *
7) QTD_DIAS_PRIMEIRO_CHEQUE: 5 4134 7953 1 ( 0.71698 0.04015
0.06096 0.05781 0.12409 ) *
```

APÊNDICE B

Consulta Dupla — Dificuldade Alta

```
> summary(dadosTreeStatusCut3)

Classification tree:
tree(formula = paste("STATUS_1  ", stringAttCut3, -STATUS_1
-STATUS_2 -SCORE"),
data = dadosCut3)
Variables actually used in tree construction:
character(0)
Number of terminal nodes:  1
Residual mean deviance:  1.386 = 27730 / 20000
Misclassification error rate:  0.4985 = 9970 / 20000
```

```
> summary(dadosTreeScoreCut3)

Classification tree:
tree(formula = paste("SCORE  ", stringAttCut3, -STATUS_1
-STATUS_2 -SCORE"),
data = dadosCut3)
Variables actually used in tree construction:
[1] "VL_COMPRA"
Number of terminal nodes:  2
Residual mean deviance:  2.045 = 40900 / 20000
Misclassification error rate:  0.5282 = 10563 / 20000
```

```
> summary(dadosTreeStatusCut5)

Classification tree:
tree(formula = paste("STATUS_1  ", stringAttCut5, -STATUS_1
-STATUS_2 -SCORE"),
data = dadosCut5)
Variables actually used in tree construction:
[1] "QTD_DIAS_ULTIMO_CHEQUE"
Number of terminal nodes:  3
Residual mean deviance:  1.268 = 25360 / 20000
Misclassification error rate:  0.3527 = 7053 / 20000

> summary(dadosTreeScoreCut5)

Classification tree:
tree(formula = paste("SCORE  ", stringAttCut5, -STATUS_1
-STATUS_2 -SCORE"),
data = dadosCut5)
Variables actually used in tree construction:
[1] "QTD_DIAS_ULTIMO_CHEQUE"
Number of terminal nodes:  3
Residual mean deviance:  2.717 = 54340 / 20000
Misclassification error rate:  0.5198 = 10396 / 20000
```

```
> summary(dadosTreeStatusCluster3)

Classification tree:
tree(formula = paste("STATUS_1  ", stringAttCluster3,
-STATUS_1 -STATUS_2 -SCORE"),
data = dadosCluster3)
Variables actually used in tree construction:
character(0)
Number of terminal nodes:  1
Residual mean deviance:  1.386 = 27730 / 20000
Misclassification error rate:  0.4985 = 9970 / 20000

> summary(dadosTreeScoreCluster3)

Classification tree:
tree(formula = paste("SCORE  ", stringAttCluster3, -STATUS_1
-STATUS_2 -SCORE"),
data = dadosCluster3)
Variables actually used in tree construction:
character(0)
Number of terminal nodes:  1
Residual mean deviance:  2.129 = 42570 / 20000
Misclassification error rate:  0.5994 = 11988 / 20000
```

```
> summary(dadosTreeStatusCluster5)

Classification tree:
tree(formula = paste("STATUS_1  ", stringAttCluster5,
-STATUS_1 -STATUS_2 -SCORE"),
data = dadosCluster5)
Variables actually used in tree construction:
[1] "VL_COMPRAPZ_MIN_PARC"
Number of terminal nodes:  3
Residual mean deviance:  1.333 = 26650 / 20000
Misclassification error rate:  0.3962 = 7924 / 20000

> summary(dadosTreeScoreCluster5)

Classification tree:
tree(formula = paste("SCORE  ", stringAttCluster5, -STATUS_1
-STATUS_2 -SCORE"),
data = dadosCluster5)
Variables actually used in tree construction:
[1] "VL_COMPRA"
Number of terminal nodes:  2
Residual mean deviance:  3.008 = 60150 / 20000
Misclassification error rate:  0.6171 = 12342 / 20000

[1] "################################################"
[1] "SCORE CONSULTA = 26SCORE CONSULTA = 73"
[1] "################################################"
```

```
> print(paste("SCORE = ", consultasCut3$SCORE));
[1] "SCORE = (-0.08,33.3]SCORE = (66.7,100]"
> print(dadosTreeStatusCut3)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 27730 1 ( 0.4985 0.5015 ) *
> print(dadosTreeScoreCut3)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 41330 (66.7,100] ( 0.3971 0.1731 0.4298 )
2) VL_COMPRA: (-0.001,0.333] 17063 35170 (66.7,100] ( 0.3689
0.1745 0.4565 ) *
3) VL_COMPRA: (0.333,0.667],(0.667,1] 2937 5736 (-0.08,33.3] (
0.5608 0.1651 0.2741 ) *
```

```
> print(paste("SCORE = ", consultasCut5$SCORE));
[1] "SCORE = (20,40]SCORE = (60,80.1]"
> print(dadosTreeStatusCut5)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 27730 1 ( 0.4985 0.5015 )
2) QTD_DIAS_ULTIMO_CHEQUE: (0.199,0.4],(0.4,0.6],(0.6,0.801],(0.801,1]
12653 16860 0 ( 0.6153 0.3847 )
4) QTD_DIAS_ULTIMO_CHEQUE: (0.4,0.6],(0.6,0.801],(0.801,1] 5567
6629 0 ( 0.7174 0.2826 ) *
5) QTD_DIAS_ULTIMO_CHEQUE: (0.199,0.4] 7086 9789 0 ( 0.5350
0.4650 ) *
3) QTD_DIAS_ULTIMO_CHEQUE: (-0.001,0.199] 7347 8943 1 ( 0.2974
0.7026 ) *
> print(dadosTreeScoreCut5)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 58940 (80.1,100] ( 0.31595 0.11790 0.09955
0.12580 0.34080 )
2) QTD_DIAS_ULTIMO_CHEQUE: (-0.001,0.199] 7347 17910 (80.1,100]
( 0.14591 0.06792 0.07214 0.12100 0.59303 ) *
3) QTD_DIAS_ULTIMO_CHEQUE: (0.199,0.4],(0.4,0.6],(0.6,0.801],(0.801,1]
12653 37410 (-0.08,20] ( 0.41468 0.14692 0.11547 0.12859
0.19434 )
6) QTD_DIAS_ULTIMO_CHEQUE: (0.199,0.4] 7086 21680 (-0.08,20] (
0.32106 0.13364 0.12475 0.14733 0.27321 ) *
7) QTD_DIAS_ULTIMO_CHEQUE: (0.4,0.6],(0.6,0.801],(0.801,1] 5567
14750 (-0.08,20] ( 0.53386 0.16382 0.10365 0.10472 0.09395 ) *
```

```
> print(paste("SCORE = ", consultasCluster3$SCORE));
[1] "SCORE = 2SCORE = 1"
> print(dadosTreeStatusCluster3)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 27730 1 ( 0.4985 0.5015 ) *
> print(dadosTreeScoreCluster3)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 42570 1 ( 0.4006 0.3847 0.2147 ) *


> print(paste("SCORE = ", consultasCluster5$SCORE));
[1] "SCORE = 3SCORE = 4"
> print(dadosTreeStatusCluster5)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 27730 1 ( 0.4985 0.5015 )
2) VL_COMPRA: 1,2,3,4 10421 14080 0 ( 0.5936 0.4064 ) *
3) VL_COMPRA: 5 9579 12850 1 ( 0.3950 0.6050 )
6) PZ_MIN_PARC: 1,2,3,5 3159 4376 0 ( 0.5150 0.4850 ) *
7) PZ_MIN_PARC: 4 6420 8196 1 ( 0.3360 0.6640 ) *
> print(dadosTreeScoreCluster5)
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 20000 61500 1 ( 0.3044 0.1216 0.1424 0.1502 0.2813 )
2) VL_COMPRA: 1,2,3,4 10421 32110 5 ( 0.1990 0.1309 0.1746
0.1459 0.3496 ) *
3) VL_COMPRA: 5 9579 28040 1 ( 0.4191 0.1116 0.1073 0.1548
0.2071 ) *
```

Referências Bibliográficas

[Cra96] Mark William Craven. *Extracting Comprehensible Models from Trained Neural Networks*. PhD thesis, University of Wisconsin - Madison, 1996. Supervisor-Jude W. Shavlik.

[Cyb89] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989.

[DA01] Peter Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press, December 2001.

[Hay98] Simon S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2nd edition, July 1998.

[HH99] Robert J. Hilderman and Howard J. Hamilton. Knowledge discovery and interestingness measures: A survey. Technical report, Department of Computer Science, University of Regina, Regina, Saskatchewan, Canada S4S 0A2, 1999.

[HK05] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 2nd edition, November 2005.

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, March 1997.

[MP43] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysic*, 5:115–133, 1943.

[MP69] Marvin L. Minsky and Seymour A. Papert. *Perceptrons*. MIT Press, Cambridge, MA, USA, 1969.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Ros62] Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, DC, USA, 1962.

[RWL94] David E. Rumelhart, Bernard Widrow, and Michael A. Lehr. The basic ideas in neural networks. *Communications of the ACM*, 37(3):87–92, March 1994.

[SN88] K. Saito and R. Nakano. Medical diagnostic expert system based on pdp model. In *IEEE International Conference on Neural Networks*, volume 1, pages 255–262, San Diego, CA, USA, July 1988. IEEE.

[Thr95] Sebastian Thrun. Extracting rules from networks with distributed representations. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 505–512, Cambridge, MA, USA, 1995. MIT Press.

[Wer74] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA, USA, 1974.