3
Aprendizado de Máquina em Jogos Eletrônicos

Jogos populares como xadrez e dama foram um dos pioneiros a utilizarem técnicas de Aprendizado de Máquina (AM), também denominada na literatura como *Machine Learning* (*ML*). Mas estes jogos apresentam diferenças para o estilo de jogo adotado neste trabalho, os quais possuem gráficos realistas e um gerenciamento mais cuidadoso dos recursos disponíveis para execução dos mesmos. Indicando assim, que não são aplicações dedicadas para realizarem somente cálculos de Inteligência Artificial (IA). Neste capítulo são discutidas técnicas de aprendizado que estão sendo utilizadas em jogos de entretenimento atualmente.

3.1.
Avaliação da Necessidade

O campo de AM em jogos de entretenimento em tempo real, até o momento, foi pouco explorado, mas já manifestou bons exemplos de sucesso. Há vários motivos para a falta do uso de técnicas de aprendizado, segundo Manslon (2002):

- a utilização dessas técnicas é vistas como de alto risco;
- há sempre uma associação com técnicas como redes neurais e algoritmos genéticos, que normalmente têm uma "fama" de serem consideradas de difícil utilização em jogos.

Apesar da falta do uso dessas técnicas de AM, ou algoritmos baseados no princípio de aprendizado, há vários motivos e benefícios em uma maior utilização das mesmas. Por exemplo: adaptar o jogo a estilos de jogadores que não foram previstos na fase de desenvolvimento; ou determinar soluções para problemas que seriam difíceis de resolver manualmente. No jogo *Colin McRae Rally 2.0,* é usada uma rede neural para aprender como dirigir um carro, evitando a necessidade de escrever um conjunto complexo de regras para esta tarefa.

Mas também é preciso verificar a real necessidade da utilização de aprendizado em um jogo, pois poderá ocorrer uma situação onde poderá haver um

gasto elevado de tempo de desenvolvimento em algo que poderá não ser percebido pelo jogador. É fundamental analisar os benefícios para os desenvolvedores e jogadores, e a quantidade adicional de complexidade exigida para implementar esses algoritmos, tanto nas fases de desenvolvimento quanto nas de testes. Às vezes, a escolha por uma "impressão" de aprendizado pode ser mais vantajosa. Por exemplo: induzir erros para uma IA que resolve muito bem um problema e, com o passar do tempo, fazer que esta quantidade de erros seja diminuída para passar a impressão de um aprendizado progressivo.

3.2. Adaptação

Em jogos, segundo Manslon (2002), o aprendizado pode ser feito através de duas maneiras de comportamento adaptativo:

- **Adaptação indireta:** extrai dados do mundo que são usados pelo sistema de IA, para modificar o comportamento do agente. Quais os dados, assim como sua interpretação para determinar alguma mudança no comportamento do agente são determinadas pelo *designer* da IA;
- **Adaptação direta:** são aplicados algoritmos de aprendizado no próprio agente.

Na adaptação indireta o papel do mecanismo de aprendizado reside basicamente na extração dos dados para o sistema de IA, não há um uma intervenção direta na modificação do comportamento do agente. Essa extração de dados é feita facilmente, resultando em uma efetiva adaptação. Por exemplo, considere em um jogo do estilo *First Person Shooter* (*FPS*), como *Quake* e *Unreal*, onde existe um agente representado por um *bot*, *roBotic computer controlled player*, que modifica seu comportamento (*pathfinding* – cálculo de caminho) para visitar mais vezes os locais do cenário onde ele obteve maior sucesso durante o jogo. Outro exemplo está no jogo *B&W*, onde este tipo de adaptação é feita para determinar quais objetos são apropriados para alimentar uma criatura, desta forma, induzindo-a a escolher somente os que são realmente bons.

O uso de algoritmos de aprendizado para a modificação direta de comportamento consiste no aprendizado com adaptação direta. Nesta forma de adaptação são feitas modificações no comportamento do agente verificando o desempenho dessas mudanças. Nesta abordagem é necessária alguma forma de parametrização do comportamento do agente, para que seja usado algum algoritmo de aprendizado por recompensa, *reinforcement learning*, para descobrir os parâmetros que oferecem melhor desempenho. Em (Neller, 2002) há a discussão de algumas idéias para este problema.

Manslow (2002) afirma que de forma geral, a técnica mais eficiente de adaptação é a indireta, pois a grande dificuldade nesta alternativa é resolvida pelos desenvolvedores da IA, que implica em decidir como o comportamento de um agente deve mudar em resposta a uma informação aprendida do mundo. O restante, que inclui essencialmente as coletas de dados sobre mundo, é feito de forma trivial.

A adaptação direta é mais eficiente em problemas mais específicos e bem limitados.

3.3. Técnicas de Aprendizado

Entre as técnicas de AM utilizadas até o momento, existem as sem embasamento científico, como por exemplo, a que foi implementada no jogo *Magic & Mayhem*, desenvolvido pela empresa *Mythos Games*, em que o jogador assume o papel de um mago que deve destruir monstros, feiticeiros entre outros obstáculos. Neste jogo, periodicamente, são gravadas informações sobre as disputas com o jogador e, antes de iniciar uma nova batalha, a IA do jogo compara a técnica de ataque a ser utilizada com estas informações armazenadas para determinar o grau de efetividade deste tipo de ataque. Desta forma o jogo tem uma característica de se adaptar ao estilo do jogador (Sewald e Giraffa, 2000).

Entre as técnicas com embasamento científico adotadas até o momento, destacam-se a utilização das Redes Neurais, Algoritmos Genéticos e, mais recentemente, as Árvores de Decisão. Nas subseções a seguir são dados os conceitos das técnicas, Redes Neurais e Árvores de Decisão, utilizadas no

presente trabalho assim como são mencionados os jogos que fazem uso das mesmas.

3.3.1. Redes Neurais

As Redes Neurais (RN) estão baseadas nos comportamentos dos neurônios no cérebro. As RNs são caracterizadas por um modelo de processamento paralelo e distribuído. Os neurônios transmitem sinais através de impulsos elétricos e esses sinais chegam até os neurônios através dos dendritos e saem através dos axônios (Buckland, 2002) (ver **Figura 10**).

Um cérebro humano é composto por 10^{11} neurônios e cada neurônio tem por volta de 10.000 conexões através dos dendritos.

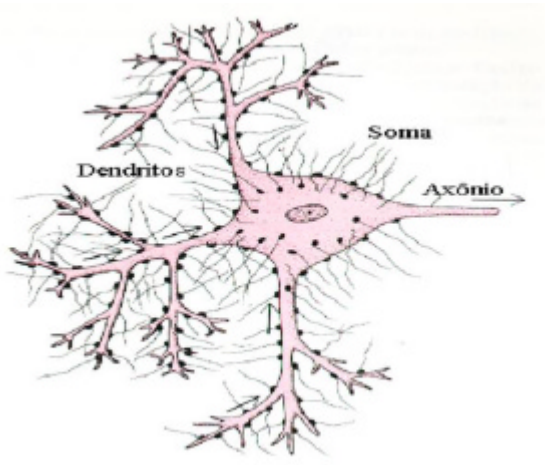

Figura 10. Neurônio biológico.

A região onde um axônio encontra um dendrito é denominada sinapse (ver **Figura 11**).

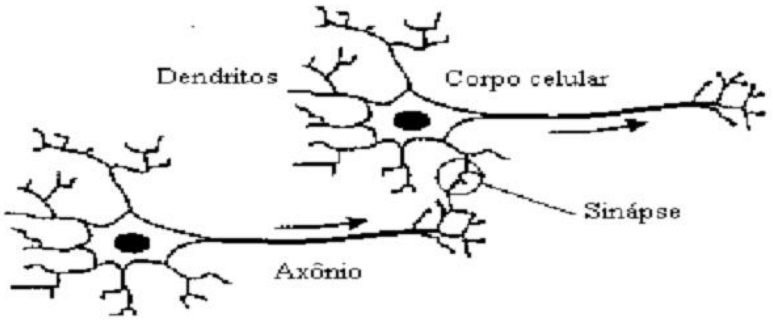

Figura 11. Sinapse.

Como substituto eletrônico do neurônio biológico, o neurônio artificial possui o seguinte formato ilustrado na **Figura 12**.

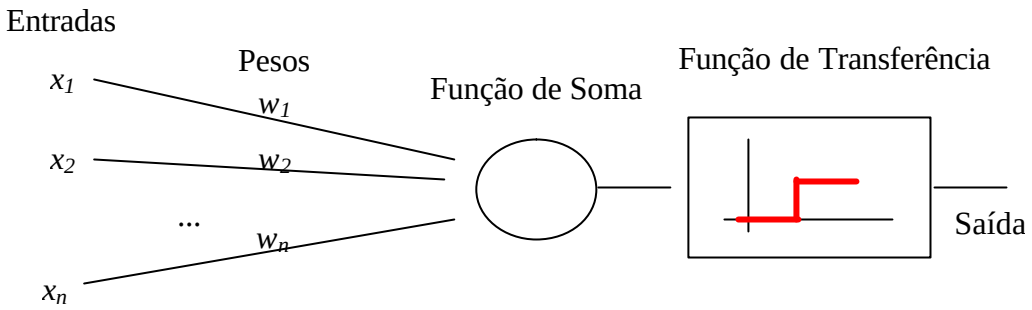

Figura 12. Neurônio Artificial.

As conexões entre os neurônios artificiais procuram simular as conexões sinápticas biológicas fazendo uso de uma variável chamada peso. A função de soma acumula os dados recebidos (estímulos) de outros neurônios e a função de transferência, ou também denominada de função de ativação, processa a função soma transformando-a. Em outras palavras, um neurônio corresponde a uma soma ponderada de entradas, soma esta aplicada a uma função de transferência que vai determinar a ativação do neurônio (Coelho, 2003).

O número de camadas de neurônios, o tipo de conexão e o tipo de treinamento são aspectos que diferenciam os tipos de RNs digitais existentes.

Quanto ao número de camadas, as redes podem ser classificadas como redes de:

- **camada única:** há somente um único neurônio entre qualquer entrada e qualquer saída da rede;
- **múltiplas camadas:** há mais de um neurônio entre alguma entrada e alguma saída da rede. Podem surgir camadas escondidas (**Figura 13**);

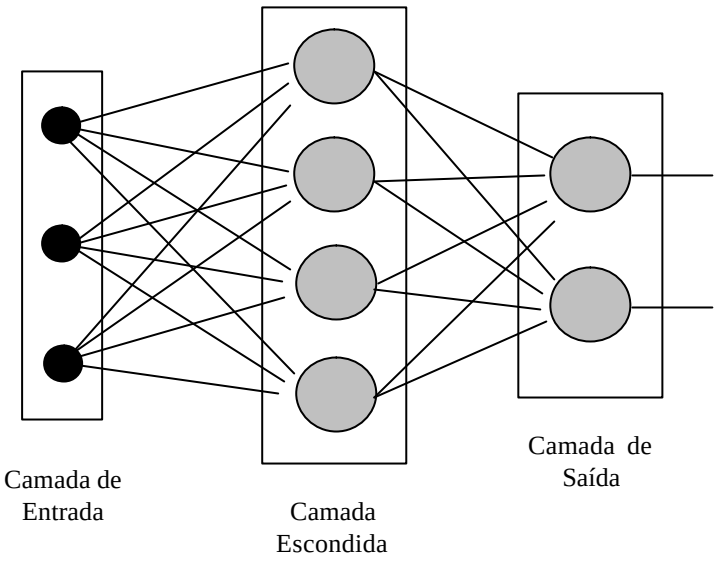

Figura 13. Rede Neural com múltiplas camadas.

Os tipos de conexões entre os neurônios podem ser:

- ***feedfoward* (acíclica):** a saída do neurônio na *i*-ésima camada da rede não pode ser usada como entrada para neurônios em camadas de índice menor ou igual a *i*;
- ***feedback* (cíclica):** a saída de algum neurônio na *i*-ésima camada da rede é usada como entrada para neurônios em camadas de índice menor ou igual a *i*;

Aprender é uma capacidade que uma RN possui graças aos seus neurônios. Se uma rede aprende, ela retém conhecimento e o conhecimento está distribuído por toda a rede. Os algoritmos de aprendizagem para RNs têm como objetivo treinar a RN modificando os pesos sinápticos fazendo com que esta se aperfeiçoe. Há duas formas de treinamento de RNs para que ocorra aprendizado:

- **supervisionado:** quando existem exemplos para treinar a rede. Ou seja, é fornecido para RN um conjunto de entradas e a saída resultante é comparada com exemplos conhecidos. Se a saída for diferente dos exemplos, então os pesos são alterados sutilmente para que na próxima vez que este mesmo conjunto de entradas for utilizado, a saída esteja um pouco mais próxima dos exemplos conhecidos;

- **não-supervisionado:** ocorre quando não há exemplos rotulados da função a ser aprendida. Neste caso, pode-se utilizar técnicas que atualizam esses pesos, como algoritmos genéticos.

Existem diversos algoritmos de aprendizado, entre os mais comuns estão a propagação para frente (*forwardpropagation*) e o de retropropagação de erro ou simplesmente retropropagação (*backpropagation*). O algoritmo de retropropagação não será detalhado neste trabalho por não fazer parte do escopo do mesmo.

O algoritmo de propagação para frente tem a característica de ser um algoritmo de aprendizagem supervisionada, e tem os seguintes passos definidos a seguir (**Quadro 1**).

1. Inicializa todos os pesos com valores randômicos entre –1.0 e 1.0;
2. Inicializa um conjunto de entradas para a camada de entrada da rede neural;
3. Ativa cada neurônio da camada corrente:
   Multiplica os valores dos pesos das conexões pelos valores das saídas dos neurônios antecedentes;
   Faz um somatório destes valores;
   Envia o resultado para uma função de transferência, que computa a saída deste neurônio;
4. Repetir até atingir a camada de saída da rede;
5. Compara as saídas calculadas com as saídas desejadas e computa um erro;
6. Modifica todos os pesos, adicionando o valor do erro aos valores dos pesos;

Quadro 1. Algoritmo de Propagação para Frente.

A primeira onda de entusiasmo com as RNs surgiu com o *perceptron* de *Frank Rosenblatt*, publicado no estudo *The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain*, em 1958. Atualmente este nome está associado a uma RN simples composta por apenas uma camada, mas existem estudos sobre *perceptrons* com múltiplas camadas. Um *perceptron* consegue aprender a classificação de padrões a partir de exemplos, fazendo com que os pesos da rede fiquem convergidos de tal forma que representem corretamente estes exemplos, mas estes exemplos precisam ser representados por funções linearmente separáveis. O algoritmo de treinamento para os *perceptrons* utiliza um algoritmo de aprendizagem supervisionada do tipo propagação para frente, explicado anteriormente (Russel e Norvig, 2003; Coelho, 2003).

Não é simples adaptar uma IA de um jogo para utilizar RN, mas uma vez dominada a utilização desta técnica pode-se utilizá-la em conjunto com lógica

fuzzy, algoritmos genéticos, entre outros, para criar uma IA robusta. A utilização de uma RN substitui a necessidade da definição de complexos blocos de código compostos por declarações do tipo "se..então" ou o uso de *scripts.* Entre as suas utilidades estão (Lamothe, 2000):

- **classificação:** Uma RN pode ser alimentada com informações do ambiente do jogo. E estas informações podem ser utilizadas para selecionar uma resposta, ou para treinar a rede. Essas respostas podem ser aprendidas durante o jogo e atualizadas para otimizar a resposta;
- **memória:** Pode ser utilizada pelas criaturas de um jogo como uma forma de "memória". A RN pode aprender, através de experiências, um conjunto de respostas. Desta forma, quando algo de novo acontecer ela pode determinar o que pode ser mais bem feito na situação;
- **controle de comportamento:** A saída de uma RN pode ser usada para controlar as ações de uma criatura em um jogo. As entradas podem ser várias variáveis do jogo.

Um dos jogos pioneiros na utilização de RN é o *BattleCruiser: 3000 AD* (*BC3K*), http://www.3000ad.com/home.shtml, um simulador espacial desenvolvido pela empresa *Derek Smart*, que utiliza uma RN para controlar os elementos do jogo. São utilizadas variações do algoritmo de retropropagação e algoritmos não supervisionados para o processo de aprendizagem. Neste jogo, também são utilizados *scripts* com redes neurais supervisionadas pré-treinadas para serem executados em momentos específicos do jogo, como por exemplo, no início de guerras, ou no planejamento de invasões (Sweetser, 2002; Sewald e Giraffa, 2001).

Jogos de corridas de carros como *Dirt Tracking Race*, desenvolvido pela *ValueSoft*, e o *Colin McRae Rally 2.0*, desenvolvido pela empresa *Codemasters*, têm os benefícios da utilização de RNs para o controle da movimentação dos carros (Woodcock, 2003; Manslon, 2002).

Outro jogo que utiliza RN é o *B&W*, sendo que na forma de *perceptrons*. Os *perceptrons* são usados para a representação dos desejos de uma criatura do jogo. Cada desejo é composto por fontes que influenciam a intensidade do desejo. Os valores destas fontes são usados como as entradas para a RN. E o aprendizado

supervisionado é aplicado quando o jogador quer moldar o comportamento da criatura da forma que lhe é apropriada, ou seja, há uma modificação dos valores dos pesos que influenciam o desejo (Evans, 2002). Mais informações sobre o uso dos *perceptrons* no *B&W* estão no Capítulo 5 sobre a arquitetura de IA deste jogo.

Está fora do escopo deste trabalho detalhar todas as características e algoritmos relativos as RNs, mas há referências como o livro *AI Thechniques for Game Programming* (Buckland, 2002) que explica o assunto de forma bem didática e agradável, tentando diminuir o formalismo matemático e dando ênfase para a aplicação em jogos eletrônicos. Neste livro há vários exemplos, incluindo um que utiliza uma rede neural em conjunto com algoritmo genético para controlar o comportamento de entidades.

3.3.2. Árvores de Decisão

As Árvores de Decisão (AD) são um dos métodos mais simples e de grande sucesso no campo do aprendizado indutivo (Russel e Norvig, 2003). O aprendizado indutivo está baseado na indução de regras a partir de exemplos. As AD são bastante utilizadas como um estimador que pode auxiliar em tomadas de decisões ou como um classificador que classifica um exemplo em uma categoria. Usada para aproximar funções alvo, é geralmente utilizada no lugar de outras técnicas não lineares devido a legibilidade das suas regras de aprendizado e a sua eficiência com relação ao seu treinamento e avaliação.

Uma AD representa uma função *h*, chamada hipótese, que se aplica a um conjunto de valores de atributo *A* retornando uma conclusão: $C = h(A)$. Essa conclusão *C* pode ter *n* valores: $C = \{ c_1, c_2, \ldots , c_n \}$, esses valores são também chamados de categorias ou classes. Induzir *h* que aproxime uma função *f* alvo (ver **Figura 14**) é encontrar a mais simples árvore que é consistente com o conjunto de treinamento (exemplos, situações, observações..).

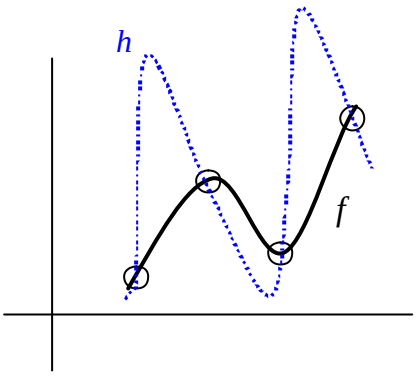

Figura 14. A função hipótese *h* tenta aproximar a função alvo *f*.

A consistência de uma hipótese *h* é muito importante segundo o princípio do **Ockham's razor:** "A hipótese mais provável é a mais simples que é consistente com todas as observações" (Russel e Norvig, 2003).

Há dois algoritmos bastante utilizados na construção de ADs: o ID3 e o C4.5. O último corresponde a uma extensão do primeiro. A maioria dos algoritmos de indução de árvores de decisão se baseiam na abordagem "dividir para conquistar" que consiste na sucessiva divisão do problema em vários subproblemas menores até que uma solução para cada um dos problemas seja encontrada (Halmenschlager e Alvares, 2001)

O ID3 constrói uma árvore a partir de um conjunto fixo de exemplos. A futura árvore pode ser utilizada para classificar futuros exemplos. Um exemplo é composto por vários atributos e pertence a uma categoria. Esta categoria nada mais é do que um valor de um atributo pertencente ao conjunto de atributos do exemplo. Ou seja, dado conjunto de atributos *A* que um exemplo *Si* contém, é escolhido um atributo *Ai* para ser o atributo classificador e os valores de *Ai* são considerados as categorias. Freqüentemente são escolhidos como classificadores os atributos com poucos valores, por exemplo, um atributo que tenha os valores {*yes* , *no*}. Os nós folhas de uma AD contém valores de categorias enquanto os demais nós contém nós de decisão. Um nó de decisão corresponde a um teste de atributo onde cada ramo saindo para outro nó de decisão corresponde a um possível valor de atributo. O algoritmo ID3 usa um critério chamado ganho de informação (*information gain*) para decidir qual atributo deve ser associado a um nó de decisão (Ingargiola, 2003; Ross, 2000).

As ADs podem ser vistas também como representações lógicas, o que tornar o entendimento desta estrutura mais fácil. Uma AD é uma conjunção de implicações (se... então...), e implicações são cláusulas de *Horn* (a concussão de literais implica em um literal). E cada atributo é considerado como uma proposição que é totalmente expressiva dentro da classe de linguagens proposicionais.

Apesar das ADs corresponderem a um conjunto de implicações, elas não podem representar qualquer conjunto e estão implicitamente limitadas a um único objeto. Não é possível utilizá-las para representar atributos que se referem a dois ou mais objetos (Russel e Norvig, 2003).

O primeiro jogo de entretenimento a utilizar árvores de decisão foi o *Black&White.* Neste jogo as ADs são utilizadas para representar as informações sobre as experiências que a criatura tem ao longo do jogo, por exemplo, as experiências sobre que tipos de objetos que foram comidos. Desta forma, em ocasiões futuras, a criatura é capaz de tomar decisões sobre que tipo de objetos que são mais apropriados para comer (Evans, 2001).

Neste trabalho é priorizada a explicação referente ao algoritmo ID3, o qual foi utilizado na implementação da aplicação final do trabalho. Nas subseções a seguir são detalhados os procedimentos necessários para construir uma AD.

Seleção de Atributo

A seleção de atributos é feita usando uma propriedade estatística chamada ganho de informação. O ganho mede o quanto um atributo é capaz de separar um conjunto de exemplos em categorias. Aquele que possuir o maior ganho é selecionado. Mas antes de continuar a definição de ganho, é necessário definir uma idéia oriunda da teoria da informação: a entropia. A entropia mede a quantidade de informação de um atributo, caracterizando a impureza de um conjunto de exemplos (Ingargiola, 2003).

Dado um conjunto de exemplos S e uma categorização C em categorias $c_1, c_2, \ldots, c_n$, a entropia *E(S)* é definida como:

$$E(S) \equiv \sum_{i=1}^{n} -p_i \log_2(p_i)$$

Um exemplo de cálculo de entropia está nos problemas de classificação binária, onde há uma categorização definida pelos valores "-" e "+", que é dada por:

$$E(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

Quando um conjunto de exemplos *S* está distribuído uniformemente nas categorias definidas para este conjunto a entropia é alta. Um exemplo para isto seria considerar que cada categoria possui o mesmo número de exemplos. A entropia é baixa quando há uma diferença grande na distribuição dos exemplos nas categorias, por exemplo, considere um conjunto de 10 exemplos que estão categorizados em apenas uma categoria, sendo que existe um conjunto de 3 categorias.

O valor da medida ganho para um atributo *A* com respeito a um conjunto de exemplos *S* é dado por:

$$Ganho(S, A) \equiv Entropia(S) - \sum_{v \in Valores(A)} \frac{|S_v|}{|S|} Entropia(S_v)$$

O ganho de informação pode ser considerado como uma redução no valor da entropia causada pela escolha do atributo A.

Algoritmo ID3

Como mencionado anteriormente, este algoritmo utiliza o critério do ganho de informação para criar uma árvore. Este ganho é usado para determinar os atributos de cada nó da árvore. Dado um conjunto de exemplos *S* categorizados em categorias pertencentes a um atributo A_i, o algoritmo segue os seguintes passos especificados no **Quadro 2**.

1. Escolha o atributo A_i com o maior ganho de informação para se tornar o nó raiz. Este atributo não pode ser o que contém as categorias;
2. Para cada valor vi que A_i possa ter, crie um ramo;
3. Para cada ramo de A_i que corresponde a um valor v_i de A_i , calcule S_v. S_v corresponde a um subconjunto de S que contém somente os valores de v_i ;
4. Se S_v for vazio, crie um nó folha e associe à ele a categoria cpadrão . Esta categoria é a que possui a maior frequência no conjunto de exemplos S. Considerando S o conjunto inicial de exemplos.
5. Se S_v contém exemplos somente de uma categoria ci , então crie um nó folha e considere ci associado a ele;
6. Caso contrário, remova A_i do conjunto de atributos disponíveis para se tornarem nós, e crie um novo nó e associe a ele o atributo com o maior valor de ganho relativo a S_v. A parti deste novo nó, reinicie o ciclo a parti do passo 2 até que não haja mais atributos, ou a árvore tenha classificado todos os exemplos.

Quadro 2. Algoritmo ID3.

O diagrama na **Figura 15** explica o algoritmo visualmente.

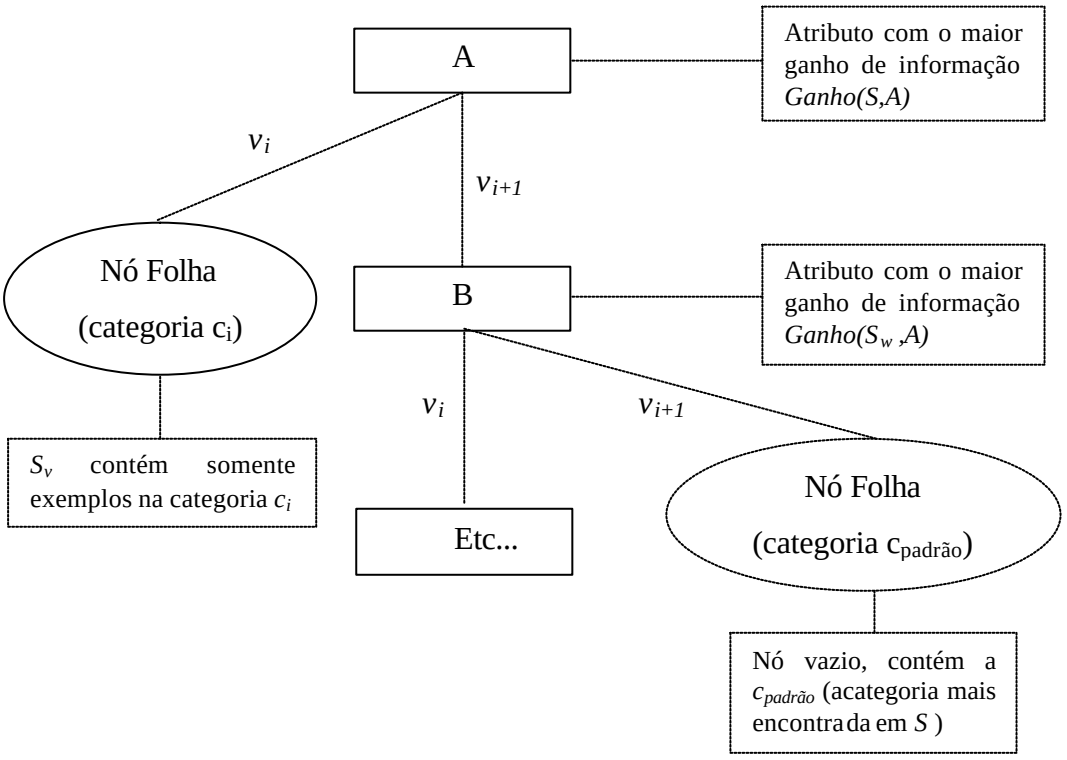

Figura 15. Esquema representativo do algoritmo ID3.

A seguir há um exemplo ilustrando situações ocorridas durante finais de semana (ver **Tabela 1**). Nas situações são identificados quatro atributos: Tempo, Parentes, Dinheiro e Decisão. O atributo "Decisão" é escolhido como o atributo categórico.

Fim de Semana	Tempo	Parentes	Dinheiro	Decisão
F1	Sol	Sim	Muito	Cinema
F2	Sol	Não	Muito	Kart
F3	Vento	Sim	Muito	Cinema
F4	Chuva	Sim	Pouco	Cinema
F5	Chuva	Não	Muito	Ficar em Casa
F6	Chuva	Sim	Pouco	Cinema
F7	Vento	Não	Pouco	Cinema
F8	Vento	Não	Muito	Shopping
F9	Vento	Sim	Muito	Cinema
F10	Sol	Não	Muito	Kart

Tabela 1. Conjunto de situações utilizadas para criar uma Árvore de Decisão.

O primeiro passo para iniciar a construção da árvore para estas situações é determinar qual o atributo que deve ser o nó raiz da árvore. Desta forma calcula-se a entropia do conjunto de situações:

$Entropia(S) : -p_{cinema} \log_2(p_{cinema}) - p_{kart} \log_2(p_{kart}) - p_{shopping} \log_2(p_{shopping}) - p_{ficar\ em\ casa} \log_2(p_{ficar\ em\ casa})$

$Entropia(S) : - (6/10) * \log_2(6/10) - (2/10) * \log_2(2/10) - (1/10) * \log_2(1/10) - (1/10) * \log_2(1/10)$

$Entropia(S) : 0.4422 + 0.4644 + 0.3322 + 0.3322$

$Entropia(S) : 1.571$

A seguir é preciso determinar qual o melhor atributo:

$Ganho(S,tempo) : 1.571 - (|S_{Sol}|/10) * Entropia(S_{Sol}) - (|S_{Vento}|/10) * Entropia(S_{Vento}) - (|S_{Chuva}|/10) * Entropia(S_{Chuva})$

$Ganho(S,tempo) : 1.571 – (0.3)*(0.918) – (0.4)*(0.81125) – (0.3)*(0.918)$

$Ganho(S,tempo) : 0.70$

$Ganho(S,parentes) : 1.571 - (|S_{Sim}|/10) * Entropia(S_{Sim}) - (|S_{Não}|/10) * Entropia(S_{Não})$

$Ganho(S,parentes) : 1.571 – (0.5) * 0 – (0.5) * 1.922$

$Ganho(S,parentes) : 0.61$

$Ganho(S,dinheiro) : 1.571 - (|S_{Muito}|/10) * Entropia(S_{Muito}) - (|S_{Pouco}|/10) * Entropia(S_{Pouco})$

$Ganho(S,dinheiro) : 1.571 – (0.7) * (1.842) – (0.3) * 0$

$Ganho(S,dinheiro) : 0.2816$

Como o atributo “Tempo” (ver
Figura 16) teve o maior ganho ele será o primeiro nó da árvore.

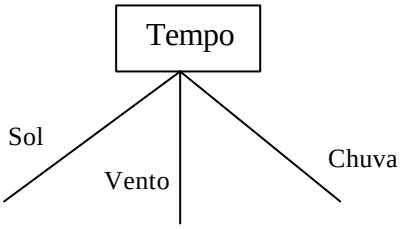

Figura 16. O atributo com maior ganho de informação é o nó raiz da árvore de decisão.

Analisando o conjunto de exemplos S_{Sol} (F1, F2, F10) formados pelo ramo “Sol” é verificado que as categorias pertencentes a estes exemplos não são do mesmo tipo, ou seja, a categoria de F1 é “Cinema” e a categoria de F2 e F10 é “Kart”. Desta forma cria-se um novo nó neste ramo. A mesma situação ocorre para o conjunto S_{Vento} (F3, F7, F8, F9) e S_{Chuva} (F4, F5, F6) (ver
Figura 17).

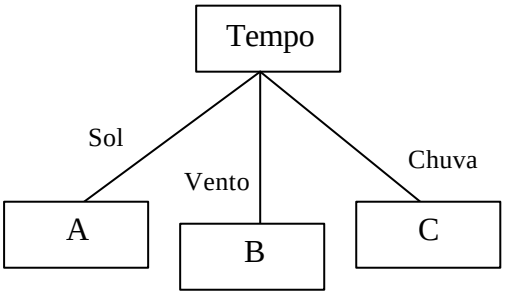

Figura 17. Os ramos do atributo Tempo apontam para outros nós que precisam de atributos.

A seguir inicia-se o calculo para escolher o atributo que vai pertencer ao nó “A”. Lembrando que este não pode ser o atributo “Tempo” pois o mesmo já foi escolhido para outro nó. Desta forma é necessário calcular Ganho(S_{Sol},Parentes) e Ganho(S_{Sol}, Dinheiro). Inicia-se calculando Entropia(S_{Sol}) , lembrando que S_{Sol} compreende somente as situações F1,F2 e F10.

Ganho(S_{Sol},parentes) : 0.918 – (|S_{Sim}|/3) * Entropia(S_{Sim}) - (|$S_{Não}$|/3) * Entropia($S_{Não}$)
Ganho(S_{Sol},parentes) : 0.918 – (1/3)*0 – (2/3)*0
Ganho(S_{Sol},parentes) : 0.918

Ganho(S_{Sol},dinheiro) : 0.918 - (|S_{Muito}|/3) * Entropia(S_{Muito}) - (|S_{Pouco}|/3) * Entropia(S_{Pouco})
Ganho(S_{Sol},dinheiro) : 0.918 – (3/3)*0.918 – (0/3)*0
Ganho(S_{Sol},dinheiro) : 0

Note que a entropia de S_{Sim} e $S_{Não}$ tiveram o valor 0 pois nos dois conjuntos haviam categorias iguais, ou seja, em S_{Sim} somente haviam categorias do tipo “Cinema” e em $S_{Não}$ só haviam categorias do tipo “Kart”. Uma vez calculados os ganhos coloca-se o atributo “Parentes” no nó A.

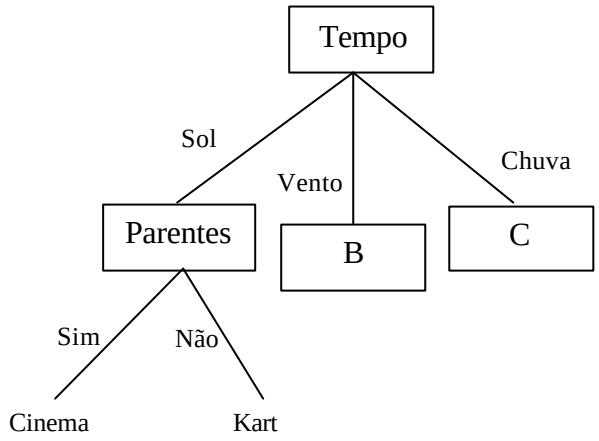

Figura 18. O atributo Parentes é escolhido como o primeiro nó filho do atributo Tempo.

Continuando os passos do algoritmo ID3 para o restante dos atributos é obtida a árvore ilustrada na **Figura 19**:

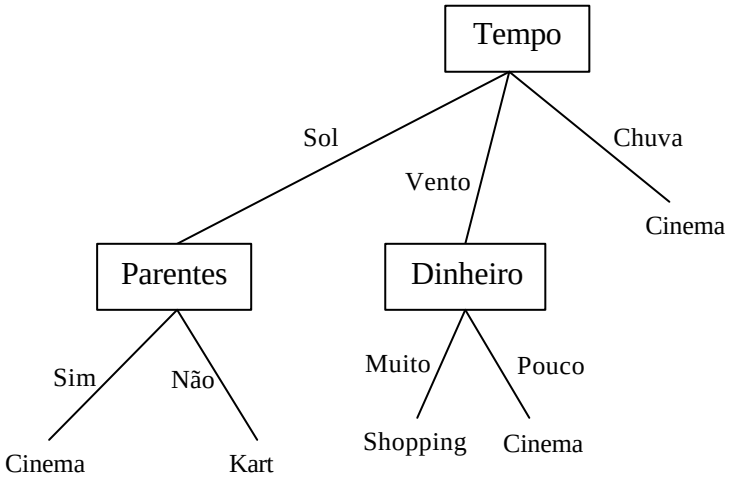

Figura 19. Árvore de Decisão resultante do conjunto de situações da Tabela 1.