Sistemas de aprendizagem: introdução aos algoritmos genéticos, às árvores de decisão e ao raciocínio baseado em casos

João Coelho e Luís Fernandes,

Universidade do Minho
{a74859, a74748}@alunos.uminho.pt

Resumo. Atualmente, o diálogo em torno da tecnologia deriva frequentemente em Inteligência Artificial. Este é um tema cada vez mais presente, mais estudado e mais trabalhado, pelo que é importante ter uma noção de tudo o que engloba. Aqui são abordados 3 algoritmos de aprendizagem para sistemas de apoio à decisão: raciocínio baseado em casos (*case-based reasoning)*, algoritmos genéticos (g*enetic algorithms)* e árvores de decisão (d*ecision trees)*. O funcionamento do algoritmo e a forma como se manifesta a aprendizagem são os principais assuntos explorados sobre cada um dos três paradigmas, com destaque também para a exposição de ferramentas e soluções no mercado.

Palavras-chave: algoritmo, decisão, caso, paradigma, aprendizagem, sistema, dados, exemplo.

1 Introdução

Em meados dos anos 50, no século XX, John McCarthy criou a expressão "Inteligência Artificial", que o próprio definiu como sendo "a ciência e engenharia de tornar as máquinas inteligentes."

Hoje, segundo a enciclopédia online *Wikipedia*, Inteligência Artificial é a inteligência exibida através de máquinas. Na ciência da computação, o campo da pesquisa da Inteligência Artificial define-se como o estudo de "agentes inteligentes": qualquer dispositivo que percebe o seu ambiente e toma ações que maximizam as suas hipóteses de sucesso num dado objetivo. Coloquialmente, o termo "inteligência artificial" é aplicado quando uma máquina imita funções "cognitivas" que os seres humanos associam a outras mentes humanas, como a aprendizagem e a resolução de problemas.

À medida que as máquinas se tornam cada vez mais capazes, certas capacidades mentais que antes se pensava exigirem inteligência acabam por desmenti-lo. Por exemplo, o reconhecimento ótico de caracteres deixa de ser entendido como um exemplo de "inteligência artificial", tornando-se uma tecnologia rotineira.

O objetivo geral da pesquisa de inteligência artificial é criar uma tecnologia que permita que computadores e máquinas funcionem de forma inteligente. O problema de simular (ou criar) inteligência foi dividido em subproblemas. Abaixo, destacam-se dois deles:

- ***Machine Learning***

  É o estudo de algoritmos de computador que são melhorados, com base na experiência adquirida através de interações. Este é um campo muito explorado desde o início dos estudos sobre inteligência artificial.

- **Linguagem Natural**

  O processamento da linguagem natural concede às máquinas a capacidade de ler e entender as línguas que os humanos falam. Ou seja, a interação com um agente inteligente é fluída e próxima de uma conversa com outro ser humano.

  Este trabalho recairá sobre o primeiro dos subproblemas apresentados, distinguindo três algoritmos de aprendizagem para sistemas de apoio à decisão.

2 Árvores de decisão

Pela documentação da Oracle, "O algoritmo de árvore de decisão (…) é baseado em probabilidades condicionais. (…) árvores de decisão geram regras. Uma regra é uma declaração condicional que pode ser facilmente compreendida pelos humanos e facilmente executada numa base de dados para identificar um conjunto de registos."

Abstraindo o conceito, árvores de decisão são estruturas compostas por uma sucessão de nós, desde aquele que constitui a raiz até às extremidades, onde os nós são designados por folhas, estando nós sucessivos unidos por ramos. Desta alusão às árvores resultou o nome do sistema de aprendizagem, mas agora importa explicitar o significado de cada uma destas terminologias. Cada nó interno da árvore representa um teste de verdade, cada ramo explicita um desfecho do teste anterior e as folhas são o valor de saída, isto é, a classificação após os sucessivos testes ao *input*. Existem essencialmente dois tipos de árvores de decisão - classificação e regressão – que diferem no facto de, no caso das árvores de regressão, o resultado da previsão poder ser um valor contínuo, como um número real (o custo de algo, por exemplo), ao contrário das árvores de classificação, que só preveem a classe dos dados.

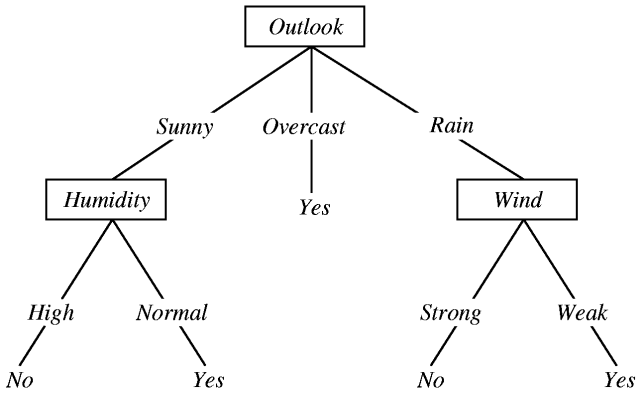

Fig. 1. Exemplo da representação gráfica de uma árvore de decisão de classificação. (https://i1.wp.com/cloudmark.github.io/images/kotlin/ID3.png)

O foco deste documento é a aprendizagem e, como tal, este capítulo centrar-se-á na aplicação deste tipo de algoritmo em *machine learning*, porém, as árvores de decisão são também usadas como modelo preditivo, para inferência indutiva[1], em áreas como a estatística e o *data mining*. Nestas situações, o objetivo é criar um modelo que preveja o valor de uma variável de entrada, baseando-se nos resultados de variáveis anteriores, qualquer que seja a classificação em causa, isto é, as árvores são treinadas de acordo com um conjunto de treino (exemplos previamente classificados) e, posteriormente, outros exemplos são classificados de acordo com essa mesma árvore. Separam-se, assim, os valores de treino dos valores de teste.

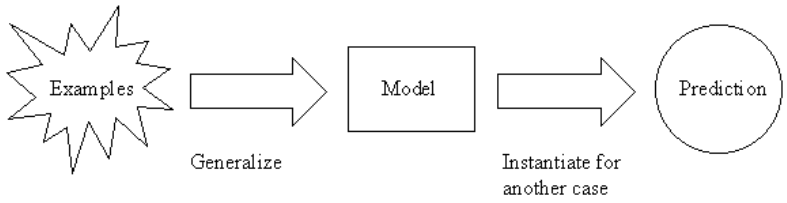

Fig. 2. As árvores de decisão aprendem a prever ao serem treinadas a partir de exemplos fornecidos.
(https://ramandeep2017.files.wordpress.com/2017/08/inductivenference.gif?w=640)

[1] Inferência indutiva é o processo de chegar a uma conclusão geral a partir de exemplos específicos.

A fase de treino consiste em usar variáveis de entrada, com um conjunto de atributos e um valor de saída conhecidos, e submetê-las aos testes ao longo da árvore de decisão, os quais se baseiam nos atributos conhecidos. Atingindo o nível das folhas, compara-se o valor de saída da variável de teste com a classificação presente na folha da árvore. Esta fase pode coincidir com a construção da árvore de decisão, caso os dados fornecidos sejam insuficientes para gerar e testar a árvore. Havendo dados suficientes, uma parte usa-se na construção e os restantes testam a árvore construída. Também importa mencionar que, quanto maior a amostra, melhor será a aprendizagem da árvore, aumentando a probabilidade de correção na fase de teste.

A fase de teste aplica o conhecimento adquirido na fase de treino para prever o valor de novos *inputs*. Para estas novas variáveis de entrada, não se conhece o valor da classificação final, mas são conhecidos os atributos necessários para responder aos testes da árvore, o que permite determinar a folha em que termina essa variável e, consequentemente, prever o valor de saída. Esta previsão é tão mais correta quanto mais intensivo foi o treino da árvore de decisão.

Antes das duas fases abordadas anteriormente – treino e teste – há o processo de construção da árvore de decisão, que ocorre em função do conjunto de dados que caracteriza as variáveis do problema – os atributos conhecidos. São, assim, usados diferentes algoritmos, como CART, ID3 e C4.5, os dois últimos criados por J. Ross Quinlan. O C4.5 é uma versão posterior e melhorada do ID3, com vantagens como a aceitação de atributos contínuos e discretos, todavia, o processo de ambos pode, concisamente, ser dividido em três passos:

1. Seleção das variáveis a partir do conjunto de atributos conhecidos, isto é, das variáveis que compõem o conjunto de dados, seleciona-se a que será a dependente e todas as independentes.

2. Análise a cada variável que afeta o resultado, através de um processo iterativo de agrupamento sobre os valores contidos em cada uma destas variáveis. Para tal, utilizam-se os conceitos de entropia e ganho.

    - A entropia é a medida de incerteza num determinado tipo de dados, ou seja, o grau de desordem. Matematicamente, dado um conjunto *S*, com instâncias pertencentes à classe *i*, com probabilidade p_i, temos:

$$Entropia(S) = \sum p_i \log_2 p_i$$

    - O ganho mede a redução de entropia esperada, decidindo que atributos são usados nos nós de decisão. A fim de reduzir a profundidade da árvore, o atributo com menor redução de entropia é a melhor escolha. *Ganho(S,A)* significa a redução esperada na entropia de S, ordenando pelo atributo A:

$$Ganho(S, A) = Entropia(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} \cdot Entropia(S_v)$$

3. Seleção da variável considerada a mais influente para a variável dependente, após os agrupamentos terem sido criados e analisados, que é utilizada para criar os nós das folhas da árvore.

Para analisar um exemplo prático, considere-se o seguinte problema (https://dzone.com/articles/machine-learning-with-decision-trees): são recolhidos dados climatéricos para a construção de uma árvore de decisão, cuja classificação alvo pretende esclarecer sobre jogar ou não basebol.

Day	Outlook	Temperature	Humidity	Wind	Should I play baseball?
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Fig. 3. Conjunto de dados exemplo que suportarão a construção e o treino da árvore. (https://dzone.com/articles/machine-learning-with-decision-trees)

Tem-se, portanto, os seguintes atributos: aspeto do clima (*outlook*), temperatura (*temperature*), humidade (*humidity*) e vento (*wind*), cada um com diferentes valores. É necessário descobrir qual estará no nó raiz, já que a construção da árvore é feita no sentido descendente, em direção às folhas (*top-down*). Neste exemplo, como só existem dois valores para a classificação final, é possível agrupar os exemplos positivos e os exemplos negativos em duas porções. A equação da entropia desdobra-se, assim, em:

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

Entropia(S) = – (9/14) Log2 (9/14) – (5/14) Log2 (5/14) = 0.940

Tendo a entropia, calcula-se o ganho para os diferentes atributos:

Ganho(S,**Vento**) = Entropia(S) – (8/14)*Entropia(S**fraco**) – (6/14)*Entropia(S**forte**)

= 0.940 – (8/14) * 0.811 – (6/14) * 1.00

= 0.048

Entropia(S**fraco**/*weak*) = – (6/8) Log2 (6/8) – (2/8) Log2 (2/8) = 0.811

Entropia(S**forte**/*strong*) = – (3/6) Log2 (3/6) – (3/6) Log2 (3/6) = 1.00

Repetindo o processo para os restantes atributos:

Ganho(S,**Aspeto**) = 0.246

Ganho(S,**Temperatura**) = 0.029

Ganho(S,**Humidade**) = 0.151

Com estes resultados, percebe-se que o aspeto do clima é o atributo com maior ganho e, como tal, deve ser usado como fator de decisão no nó raiz.

O processo de construção da árvore continua. Como para o aspeto do clima há três valores possíveis – solarengo (*sunny*), nublado (*overcast*) e chuva (*rain*) – o nó raiz terá três ramos, sendo necessário avaliar que atributo será usado na decisão no nó que estará na outra extremidade do ramo. Sabendo que um atributo já foi usado, a decisão passará agora por temperatura, humidade ou vento. Começando pelo nó resultante do ramo correspondente a um aspeto solarengo, efetuam-se os seguintes cálculos:

Ssolarengo = {D1, D2, D8, D9, D11} = 5 (filtragem da tabela)

Ganho(Ssolarengo,**Humidade**) = 0.970

Ganho(Ssolarengo,**Temperatura**) = 0.570

Ganho(Ssolarengo,**Vento**) = 0.019

O maior ganho coincide com o atributo humidade, logo será usado na decisão deste nodo. Estes cálculos repetem-se até não restarem atributos ou a árvore conseguir classificar perfeitamente os dados fornecidos. Neste exemplo, a árvore de decisão resultante é a da Fig. 1.

As vantagens de se construir uma árvore de decisão como sistema de aprendizagem, reveladas pelo exemplo anterior, podem ser sumariadas em:

- Não ser necessário conhecimento sobre o tema;
- Facilidade de compreensão;
- Processo de aprendizagem e classificação fácil e rápido.

Contudo, naturalmente também apresentam desvantagens, relativamente a outros sistemas de aprendizagem. Algumas dessas desvantagens são:

- A árvore gerada pode ser demasiado complexa e não generalizar bem em casos futuros (*overfitting)*;
- Pequenas alterações nos dados podem levar a grandes reformulações na árvore;
- A construção de uma árvore de decisão ótima é um problema NP-completo no que toca a vários aspetos de otimização, pelo que são usados algoritmos heurísticos, como aqueles que procuram a decisão ótima em cada nodo (*greedy algorithms*).

2.1 Expressão da aprendizagem

Partindo de um conhecimento nulo sobre o problema, a questão que se coloca é: como é que uma árvore de decisão aprende?

A resposta está nos exemplos de casos prévios que são fornecidos e que permitem construir a árvore. A partir destes dados, a árvore constrói um conjunto de regras *if-then-else*, que lhe permitirão comportar-se como um sistema de suporte à decisão em casos vindouros. Trata-se de uma aprendizagem por indução.

A quantidade de casos de treino torna o conjunto de regras mais ou menos sólido, porém, não é a adição de um caso de treino que aumentará o conhecimento do sistema. Pelo contrário, um novo caso ou a alteração de um dos casos previamente conhecidos pode levar, no limite, à completa reformulação da árvore de decisão. Esta reformulação traduz-se na alteração das regras induzidas anteriormente, ou seja, a árvore reaprende.

2.2 Ferramentas de desenvolvimento

O conjunto de dados usados na construção de uma árvore de decisão pode ser bastante extenso, uma vez que, quantos mais exemplos de treino, maior será a precisão da árvore no teste de novos casos. Por outro lado, a sua construção pode ser orientada segundo diferentes algoritmos. Perante isto, é habitual usar-se *software* na construção destes sistemas de aprendizagem.

Os programas de *data mining* comportam, geralmente, ferramentas para a construção de árvores de decisão. Alguns exemplos desse *software* são:

- Weka (University of Waikato, 1993)
- SAS Enterprise Miner (SAS Institue Inc, 2011)
- IBM SPSS Modeler (IBM Corporation, 2015)
- AC2 (ISoft, 1996)

Uma quantidade considerável de outros *softwares* existentes no mercado permite a construção de árvores de decisão. Estes são mais alguns exemplos:

- BigML (BigML, 2011), que oferece árvores de decisão e *machine learning* como um serviço.
- SMILES (The MIP Group, 2003), que estende os clássicos sistemas de aprendizagem com árvores de decisão de várias formas (novos critérios de divisão, pesquisa "não-gananciosa", novas partições e extração de várias e diferentes soluções).
- YaDT (Salvatore Ruggieri, Dipartimento di Informatica, Universit`a di Pisa, 2004), uma nova implementação do algoritmo C4.5, desenhada e implementada em C++ desde a raiz.
- PrecisionTree (Palisade, 2004), árvores de decisão e diagramas de influência para o Excel.

2.3 Soluções no mercado

As árvores possuem, desde há alguns anos para cá, várias aplicações no mercado, nas mais diversas áreas. São usadas em:

- **Astronomia** – filtragem de barulho das imagens provenientes do satélite espacial *Hubble Space Telescope*, usando o *software* OC1.
- **Engenharia biomédica** – identificação de aptidões a serem usadas em implantes.
- **Análise Financeira** – o algoritmo CART é usado para asserto da atratividade de *buy-writes*.
- **Medicina** – áreas do diagnóstico, cardiologia, psiquiatria e gastroenterologia, bem como na deteção de microcalcificações nas mamografias.

- **Reconhecimento de objetos** – reconhecimento de objetos tridimensionais e para visão de alto nível.
- **Processamento de texto** – uso do algoritmo ID3 para classificação de textos médicos, recorrendo a um sistema FIGLEAF (*FIne-Grained LExical Analysis Facility*).
- **Biologia molecular** – análise de sequências de aminoácidos, também recorrendo ao pacote de *software* OC1.

Agricultura, sistemas de controlo, produção, farmacologia, física e sensores remotos são outras áreas de aplicação onde, de alguma forma, as árvores de decisão têm ou tiveram impacto, num passado recente.

3 Algoritmos genéticos

Citando Jonathan Shapiro (*Department of Computer Science, University of Manchester*), no seu livro *Genetic Algorithms in Machine Learning*, "Algoritmos genéticos são algoritmos de procura estocástica que atuam sobre uma população de possíveis soluções. São basicamente baseados em mecanismos de genética populacional e seleção. As potenciais soluções são apelidadas de "genes"- *strings* de caracteres de um dado alfabeto. Novas soluções podem ser produzidas através da "mutação" de membros da população atual e "acasalando" duas soluções para formar uma nova. As melhores soluções são selecionadas para reprodução e mutações e as piores são descartadas. São métodos de procura probabilística; isto significa que os estados que exploram não são determinados somente pelas propriedades dos problemas. Processos aleatórios ajudam a conduzir a procura. Algoritmos genéticos são usados em inteligência artificial como outros algoritmos de procura – para procurar o espaço de potenciais soluções, a fim de encontrar aquela que resolve o problema."

Estes algoritmos são inspirados no processo de seleção natural e nas suas operações de mutação, cruzamento e seleção. O ponto de partida é uma população de soluções ("fenótipos") candidatas à resolução do problema em causa, com tamanho dependente da natureza do problema. Estas soluções podem ser geradas aleatoriamente ou concentradas em determinada área, onde é maior a probabilidade de encontrar boas soluções. O intervalo de soluções é apelidado de espaço de procura (*the search space*). Computacionalmente, a representação mais comum de uma solução é um *array* de *bits.* Duas soluções, tendo o mesmo comprimento, facilitam as operações de cruzamento entre elas.

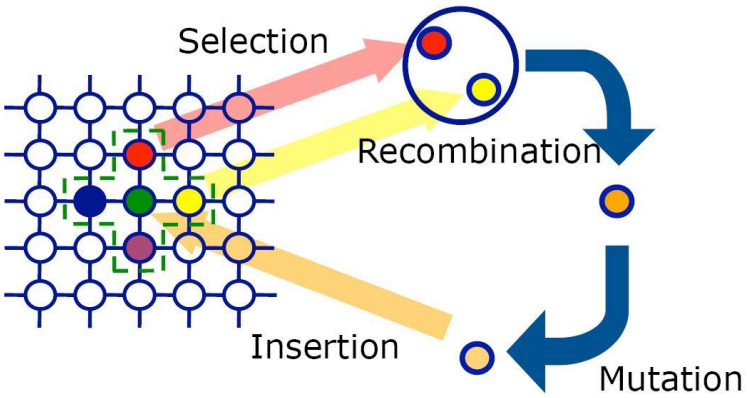

Fig. 4. Fases do processo de iterativo de um algoritmo genético. (http://autoaprendi.webcindario.com/images/jcell2.jpg).

A evolução do algoritmo ocorre, iteração a iteração, no sentido da otimização das soluções. Cada solução candidata possui um conjunto de propriedades, apelidadas de cromossomas, e o conjunto de soluções numa dada iteração constitui uma geração. Em cada geração, é aplicada uma função objetivo (*fitness function*) às soluções, relacionada com o problema que se pretende resolver e adaptada às propriedades genéticas das soluções, sendo o valor resultante o termo de comparação entre elas. As melhores soluções são destacadas da restante população – seleção – e o seu genoma é recombinado e/ou mutado – cruzamento e mutação – surgindo assim uma nova geração de soluções, que participará na próxima iteração do algoritmo, constituída por soluções que geralmente partilham características daquelas que lhes deram origem. Nalguns casos, o elevado número de soluções leva a que a função objetivo seja aplicada apenas a uma fração aleatória e não a todas as soluções, para reduzir o tempo da operação. Por outro lado, há casos em que é muito difícil ou mesmo impossível definir uma função objetivo, pelo que é realizada uma simulação para determinar o valor da solução.

O término de um algoritmo genético ocorre quando se atinge um determinado número máximo de iterações (*threshold*), quando se chega ao limite de tempo/orçamento disponível ou quando se alcança uma solução que cumpre determinados critérios de satisfação.

Comparativamente com outros sistemas de aprendizagem, os algoritmos genéticos apresentam algumas limitações, tais como:

- Não escalam bem com a complexidade. Encontrar a solução ótima para um problema do mundo real pode requerer várias horas, ou mesmo dias, de simulação;
- A melhor solução obtém-se por comparação com outras soluções, pelo que o critério de paragem nem sempre é óbvio em todos os problemas;
- Há a tendência de convergirem para soluções ótimas locais – ou mesmo pontos arbitrários - e não globais.

3.1 Expressão da aprendizagem

Como já foi mencionado, um algoritmo genético gera uma coleção de potenciais soluções, descobre o *fitness* de cada uma, escolhe as mais *fit*, altera-as de alguma forma e aplica os mesmos passos à geração resultante. Cada iteração aumenta o *fitness* geral da população, até se encontrar uma solução ótima. Isto resume a aprendizagem do algoritmo. Ao selecionar as soluções com valor objetivo mais favorável e descartar as outras, o sistema está a aprender o caminho para a solução ótima, utilizando para tal o único conhecimento que possui à partida – a função objetivo –, considerando que a seleção não é aleatória. Fatores como a complexidade do problema vão condicionar a velocidade de aprendizagem do algoritmo genético.

3.2 Ferramentas de desenvolvimento

É possível desenvolver *Genetic Algorithms* a partir de praticamente todas as linguagens de programação, uma vez que existem inúmeras bibliotecas que permitem o uso dos mesmos, porém, as mais comuns são Python e Java.

No final da década de 1980, a *General Electric* começou a vender o primeiro produto baseado em *Genetic Algorithms*: um kit de ferramentas *mainframe-based,* projetado para processos industriais.

Em 1989, a *Axcelis, Inc.* lançou o *Evolver*, o primeiro produto comercial baseado em *GA* para computadores de mesa. O escritor de tecnologia do *New York Times* John Markoff desvendou o Evolver ao mundo em 1990, tendo este *software* permanecido o único algoritmo genético comercial interativo até 1995. O *Evolver* foi vendido ao grupo *Palisade* em 1997, traduzido para vários idiomas e está atualmente na sua 6ª versão.

Atualmente encontram-se disponíveis várias bibliotecas e *frameworks* para implementação e desenvolvimento de algoritmos genéticos, entre elas:

- EvolveDotNet (*Framework open-source* para Algoritmos Genéticos - C#);
- GAlib (*Framework open-source* para Algoritmos Genéticos - C++);
- GAUL (Biblioteca *open-source* para Algoritmos Genéticos e meta heurísticas - C);
- GeneticSharp (Biblioteca *open-source* e multiplataforma para Algoritmos Genéticos - C#);
- JAGA (Pacote *open-source* para Algoritmos Genéticos e Programação Genética - Java);
- JGAP (Pacote *open-source* para Algoritmos Genéticos - Java);
- jMetal (*Framework open-source* para otimização multiobjetivo que contém Algoritmos Genéticos - Java);
- Pyevolve (*Framework open-source* para Algoritmos Genéticos e Programação Genética - Python).

3.3 Soluções no mercado

Além dos *softwares* de desenvolvimento, existem outros que permitem a utilização de algoritmos genéticos. Entre eles, o mais conhecido será provavelmente o MATLAB, mas não é o único. *GeneHunter* é uma poderosa solução de *software* para problemas de otimização, que utiliza uma metodologia de algoritmo genético de última geração. O *GeneHunter* inclui um complemento do Microsoft Excel que permite ao utilizador executar um problema de otimização nesta plataforma, bem como uma biblioteca de vínculo dinâmico de funções de algoritmo genético, que podem ser chamadas a partir de linguagens de programação, como Microsoft Visual Basic ou C.

4 Raciocínio baseado em casos

Raciocínio baseado em casos – em inglês, *Case-Based Reasoning* - é um paradigma da inteligência artificial para raciocínio e aprendizagem. Este paradigma resolve novos problemas através da recuperação de dados armazenados de problemas resolvidos anteriormente (casos) e da adaptação das suas soluções às novas circunstâncias. Cada episódio de processamento fornece um novo caso que será armazenado para futura reutilização, tornando a aprendizagem um efeito secundário natural do processo de raciocínio. Este paradigma é também estudado na ciência cognitiva como um modelo de raciocínio humano: estudos mostram que os humanos usam pedaços de informação que guardaram de problemas anteriores para orientar o seu raciocínio numa ampla gama de tarefas, entre elas programação, resolução de problemas matemáticos, diagnósticos médicos, tomadas de decisão.

O ciclo de funcionamento de um sistema RBC apoia-se na regra dos 4 R's:

- Recuperação: a partir da apresentação ao sistema de um novo problema, é feita a recuperação na base de casos daquele mais parecido com o problema em questão. Isto é feito a partir da identificação das características mais significativas em comum entre os casos.
- Reutilização: a partir do caso recuperado, é feita a reutilização da solução associada àquele caso. Geralmente, a solução do caso recuperado é transferida ao novo problema diretamente como sua solução.
- Revisão: é feita quando a solução não pode ser aplicada diretamente ao novo problema. O sistema avalia as diferenças entre os problemas (o novo e o recuperado) e identifica que partes do caso recuperado são semelhantes ao novo caso, a ponto de poderem ser transferidas, adaptando, assim, a solução do caso recuperado à solução do novo caso.

- Retenção: é o processo de armazenar o novo caso, a sua respetiva solução e anotações de como foi obtida tal solução para futuras recuperações. O sistema decide que informação armazena e de que forma.

Importa, portanto, referir que um sistema de raciocínio baseado em casos é tanto melhor e mais capaz, quanto maior for a sua base de dados de casos passados.

Na origem destes encontram-se os *RBES*. Durante os anos 70 e 80, um dos desenvolvimentos mais acentuados no âmbito da Inteligência Artificial foi o emergir de *Rule-Based Expert Systems* (*RBES*). Estes sistemas foram aplicados nas mais variadas tarefas de diferentes domínios, entre elas resolução de problemas de *hardware*, exploração geológica ou diagnóstico médico. Em geral, os *RBES* devem ser baseados num modelo profundo, explícito e causal do conhecimento do domínio do problema, o que lhes permite raciocinar usando certos princípios.

Mas, o que acontece se o conhecimento sobre um certo domínio é superficial? Começaram assim, apesar do sucesso em muitos setores, a serem encontrados vários problemas pelos desenvolvedores do *RBES*. Estes problemas podem ser resumidos, como foi proposto por Schank, em:

- Construção difícil e demorada da base de conhecimento pretendida, devido a elicitação de conhecimentos especializados e demorados.
- Incapacidade de lidar com problemas que não são explicitamente cobertos pela regra base utilizada. Em geral, os *RBES* são úteis se o conhecimento incorporado estiver bem formalizado, circunscrito, estabelecido e estável.
- Se nenhuma instalação de aprendizagem for incorporada a um *RBES*, qualquer adição ao programa existente requer a intervenção do programador.

A resolução destes problemas através de melhores e mais avançadas técnicas e ferramentas de elicitação, melhores paradigmas de desenvolvimento, linguagens e ontologias de modelagem do conhecimento, técnicas e ferramentas avançadas para manutenção de sistemas trouxeram nas últimas décadas os sistemas RBC.

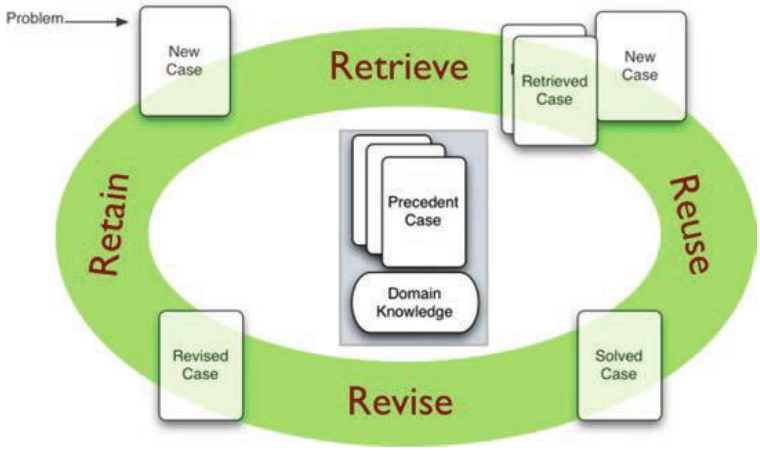

Fig. 5. Ilustração do ciclo de funcionamento de um sistema de aprendizagem com base no Raciocínio baseado em casos.
(https://www.researchgate.net/profile/Lara_Quijano-Sanchez/publication/262253429/figure/fig2/AS:392513895583744@1470594013453/Fig-2-Case-Based-Reasoning-Cycle.ppm)

4.1 Expressão da aprendizagem

Segundo Wangenhein (2003) a principal forma de expressão da aprendizagem são os casos, sendo que, outras formas de representação podem ser utilizadas, como casos abstratos, generalizados, tipos diversos de dados, modelos de objetos usados como informação, entre outros. Os casos englobam as demais formas de expressão da aprendizagem, pois todas as demais deverão ter as mesmas características básicas para o funcionamento dos sistemas RBC. Um caso é a maneira de demonstrar o conhecimento de forma contextualizada, registando o evento problemático ou uma situação cujo problema foi solucionado de forma parcial ou total.

Como já foi mencionado anteriormente, um caso é, basicamente, a descrição de um problema e a sua solução, a solução sendo caracterizada como a experiência adquirida neste caso. Esta representação do conhecimento é feita através dos casos, porém não se restringe a estes. Para um sistema RBC funcionar e ser eficaz, deve ser criado um Repositório de Conhecimento.

O Repositório de Conhecimento é uma base de dados que engloba todos os casos, o vocabulário usado para gerar estes casos (um exemplo de vocabulário são consultas médicas nas quais se preenche um formulário antes da consulta, com os sintomas. Este formulário é igual para todos os pacientes e gera um caso, facilitando o diagnóstico do médico. Este formulário seria o nosso vocabulário), uma base com medidas de similaridade referentes a cada caso para poder identificar e reutilizar casos anteriores e uma última base apara armazenar os dados de como adaptar os casos similares para resolver o novo problema.

Apresentando este Repositório de Conhecimento com uma base de dados de casos, é necessária uma medida de similaridade, que será ponto fulcral nestes sistemas e é a partir desta que todo o processo de raciocínio se torna viável. Pode-se definir similaridade como sendo a formalização de uma determinada filosofia de julgamento de semelhança, através de um modelo matemático concreto. Este processo é um dos mais importantes do paradigma RBC e tem como propósito encontrar, numa memória de casos resolvidos anteriormente, um caso mais adequado, mais próximo ao problema que precisa de resolver.

A similaridade pode ser vista como local (determina como será realizado o cálculo da similaridade entre cada atributo do caso) ou global (determina como serão computados os valores de similaridade de todos os atributos do caso atual com os casos da base, além de definir os limiares de apresentação ao utilizador). Quando um caso é resolvido, este carrega informações importantes que serão gravadas na base de dados. É nessas informações que a similaridade atua, comparando-as com o novo problema, a fim de encontrar algum caso similar que tenha sido resolvido anteriormente.

4.2 Ferramentas de desenvolvimento

Embora os sistemas RBC sejam uma metodologia da Inteligência Artificial relativamente nova, existem inúmeras aplicações bem-sucedidas, tanto no meio académico como no meio comercial. Em 1994, Watson e Marir relataram mais de 100 sistemas RBC comercialmente disponíveis.

Aparecendo com sucesso como uma alternativa a sistemas *RBES* em vários domínios, os primeiros a surgir foram:

- JUDGE (1986) – Sentença penal
- CHEF (1986) – Criação de receitas
- CASEY (1989) – Diagnóstico de insuficiência cardíaca
- JULIA (1992) – Planeamento de refeições
- CADET (1992) – Design de peças mecânicas

Foram também utilizados sistemas RBC para melhorar a tomada de decisões em jogos de vídeo.

4.3 Soluções no Mercado

Como já foi anteriormente referido, apesar de este ser um algoritmo recentemente estudado e trabalhado, já são inúmeros os *softwares* que implementam este paradigma de aprendizagem. Entre eles foram destacados:

- **Case-Based Reasoning Shell** (AIAI, Stuart Aitken) – Esta RBC Shell é uma ferramenta de *software* eficaz para orientar o diagnóstico e a descoberta de falhas em instâncias atuais, identificando padrões e conhecimentos implícitos em bases de dados de informações históricas.

- **FreeCBR** (Lars Johanson) – FreeCBR é uma implementação *open source* gratuita Java de uma ferramenta de RBC. Os casos são armazenados como texto, em que cada caso é um conjunto predefinido de recursos e onde a ideia é encontrar a melhor combinação entre os casos guardados na base de casos.

- **jCOLIBRI** (University Complutense Madrid, GAIA group) – jCOLIBRI é um *software* para desenvolver várias aplicações RBC. É baseado em Java e usa a tecnologia JavaBeans para representação de casos e geração automática de interfaces de utilizador.

- **myCBR** (German Research Center for Artificial Intelligence) – myCBR é uma ferramenta de recuperação e de desenvolvimento de *software* (SDK) baseada em similaridade. Com o MyCBR Workbench, é possível modelar e testar medidas de similaridade altamente sofisticadas e intensivas em

conhecimento numa GUI poderosa e integrá-las facilmente nas aplicações próprias através do uso do myCBR SDK. Os sistemas de recomendação de produtos baseados em casos são apenas um exemplo de *softwares* de recuperação baseados em similaridades.

- **eXiTCBR** (University of Girona) – eXiTCBR é uma ferramenta de raciocínio baseada em casos, desenvolvida no grupo de pesquisa eXiT da Universidade de Girona. Este *software* vai além da prototipagem pura de RBC e visa apoiar a experimentação. A estrutura eXiTCBR foi projetada para reunir métodos de RBC atualmente utilizados em *softwares* médicos, *softwares* de *data mining* e técnicas de visualização que podem ser utilizadas em conjunto.

5 Conclusão

Apresentados os três sistemas de aprendizagem, é agora possível concluir sobre as suas diferenças. Se a finalidade é comum aos três, a forma como conseguem levar a que uma máquina aprenda é distinta. Por outro lado, há situações mais ou menos apropriadas para um tipo de sistema.

A complexidade do problema leva frequentemente a que se abdique do uso de algoritmos genéticos, porque a função de *fitness*, indispensável à procura da solução ótima, pode também tornar-se bastante complexa.

Ao contrário dos algoritmos genéticos, que requerem conhecimento prévio do problema para que se possa construir a função, as árvores de decisão e os sistemas RBC partem com uma base de conhecimento sobre o problema nula, apoiando-se em casos passados para adquirir conhecimento. Porém, uma vez mais as diferenças vêm ao de cima. Se, nos sistemas RBC, a chegada de informação relativa a um caso exemplo adicional apenas contribui para estender a aprendizagem da máquina, no caso das árvores de decisão isto pode levar a grandes alterações na sua estrutura e, consequentemente, à reformulação do conjunto de regras *if-then-else* induzidas pelo sistema. Assim, torna-se recomendável recorrer a árvores de decisão quando há certeza nos dados exemplos fornecidos e quando a chegada de mais informação é improvável. Caso alguma destas condições falhe, pode ser melhor opção um sistema RBC, por exemplo, em detrimento de uma árvore de decisão.

Referências

1. Quinlan, J. R.: Induction of Decision Trees. (1986)
2. Shapiro, J.: Genetic Algorithms in Machine Learning. (2001)
3. Wikipedia, https://en.wikipedia.org/wiki/Decision_tree_learning
4. KDnuggets, https://www.kdnuggets.com/software/classification-decision-tree.html
5. Oracle, https://docs.oracle.com/cd/B28359_01/datamine.111/b28129/algo_decisiontree.htm
6. Wikipedia, https://en.wikipedia.org/wiki/Genetic_algorithm
7. Pantic, Maja: Introduction to Machine Learning & Case-Based Reasoning.
8. Mascarenhas, Samuel: Case Based Reasoning & Game AI. (2010)
9. Lobo, Vitor: Sistemas de Apoio à Decisão, Técnicas e Algoritmos. (2009)
10. Wikipedia: https://pt.wikipedia.org/wiki/Racioc%C3%ADnio_baseado_em_casos
11. Koza, John R.: Genetic Programming (1998)
12. Cbcb, http://www.cbcb.umd.edu/~salzberg/docs/murthy_thesis/survey/node32.html (1995)
13. Sourceforge,https://sourceforge.net/directory/development/algorithms/genetic-algorithms/os:windows/
14. GeneHunter, http://www.wardsystems.com/genehunter.asp
15. Edinburgh Innovations, http://www.research-innovation.ed.ac.uk/Opportunities/Case-Based-Reasoning
16. Passent, ElKafrawy; Rania, A. Mohamed: Comparative Study of Case Based Reasoning Software
17. myCBR, http://www.mycbr-project.net/