Sistemas de Aprendizagem

Bruno Pereira[1] and Maria Ana de Brito[2]

[1] Universidade do Minho, Braga, Portugal
a75135@alunos.uminho.pt
[2] Universidade do Minho, Braga, Portugal,
a73580@alunos.uminho.pt

Abstract. A ideia de implementar sistemas inteligentes a desempenhar tarefas que são realizadas por seres humanos parte do pressuposto que nos podemos basear na inteligência humana e adaptá-la com o maior rigor a um contexto computacional. Não passa do simples sonho de criar um cérebro artificial equiparado (ou talvez mesmo melhor) ao nosso próprio cérebro. Assim, um ser (artificial ou não) inteligente deve ser capaz de aprender e corrigir os erros, de modo a poder evoluir e adaptar-se a novas situações. Sendo a aprendizagem, então, a capacidade de adaptação, modificação e melhoramento do comportamento, esta está relacionada com a otimização da prestação do sistema, interação com o meio em que se encontra inserido, aquisição de novos conhecimentos e representação destes, pois deve ser capaz de os armazenar e explorar. Os sistemas englobam vários tipos de aprendizagem, entre os quais a aprendizagem por reforço (reinforcement learning ou RL), algoritmos genéticos (genetic algorithms) e as árvores de decisão (decision trees), que iremos abordar ao longo deste trabalho. Cada um deles aborda esta questão de aprendizagem sob um prisma diferente, porém, mesmo com as suas vantagens e limitações, são capazes de criar agentes autónomos e inteligentes.

Keywords: aprendizagem por reforço, sistemas de aprendizagem, algoritmos genéticos, árvores de decisão

1 Introdução

A capacidade de um sistema aprender sem ser explicitamente programado para tal faz parte da área de Machine Learning. O sistema está habilitado de adquirir conhecimento sobre a sua própria experiência, assim como do ambiente que o rodeia. Este aspeto torna estes tipos de sistemas muito úteis para desenvolver várias tarefas com bastante precisão e auxiliar os seres humanos em diversas indústrias.

Assim, através da interação, os sistemas são expostos a novos dados e, a partir da sua análise, adaptam-se a novos objetivos de uma forma independente. Esta vertente da Inteligência Artificial está cada vez mais a ser explorada, uma vez que permite aos agentes tomarem decisões inteligentes, sem a intervenção humana.

Os sistemas de aprendizagem possuem três paradigmas: aprendizagem supervisionada, na qual apresentamos o comportamento que o agente deve tomar; aprendizagem não supervisionada, o agente deve aprender autonomamente quais as ações a tomar e aprendizagem por reforço, que se baseia na ideia de atribuir recompensas ou penalizações, conforme a decisão do agente.

Este trabalho está dividido em três temas distintos de Machine Learning, sendo eles aprendizagem por reforço, algoritmos genéticos e árvores de decisão. Será feita a descrição de cada um deles, contextualizando-os na área de Inteligência Artificial. A sua metodologia e representação de conhecimento também serão explicadas, assim como serão apresentados alguns softwares de desenvolvimento usados e as suas aplicações no mercado. Finalmente, são expostas as conclusões finais acerca dos temas desenvolvidos.

2 Aprendizagem por Reforço

2.1 História

A aprendizagem por reforço surgiu da contribuição entre dois processos de aprendizagem: controlo ótimo e aprendizagem dos animais através de tentativa e erro. O controlo ótimo encaixa-se na programação dinâmica, enquanto que a aprendizagem animal através de tentativa e erro tem raízes na área da psicologia.

Da junção dos estudos dos dois campos referidos anteriormente, resultou um método chamado temporal difference, desenvolvido por Ian H. Witten (1957) e Richard Sutton juntamente com Andrew Barto (1981) que, através dos algoritmos que o implementam, é muito importante na aprendizagem por reforço.

2.2 Descrição geral

A aprendizagem por reforço é caracterizada pela forma que um agente deve interagir com o ambiente envolvente de modo a maximizar as recompensas recebidas. Através destas recompensas, o agente consegue determinar o comportamento ótimo capaz de maximizar o seu desempenho, podendo obter o menor custo, o menor caminho, ou algo de diferente, dependendo do contexto do problema.

De seguida, apresenta-se uma figura representativa do processo de aprendizagem por reforço. Um agente efetua uma ação sobre o ambiente. O ambiente irá comunicar com o agente e transmitir-lhe o novo estado em que o agente se situa, bem como a recompensa que a sua ação merece (podendo esta ser positiva ou negativa, mediante a ação realizada).

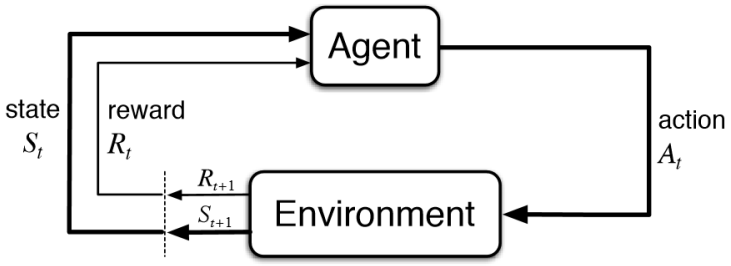


Fig. 1. Exemplo de um processo de aprendizagem por reforço

O grande desafio da aprendizagem por reforço recai sobre como o agente irá escolher as ações que irão maximizar as recompensas recebidas. Exemplificando com um exemplo real, quando tentamos ensinar um novo truque a um cão, não conseguimos explicar-lhe o que ele deve fazer, no entanto, podemos recompensá-lo quando ele efetua a ação desejada e castiga-lo quando escolhe a opção errada. O cão tem de entender quais o tipo de ações que o levaram a conseguir a recompensa

desejada. Analogamente, o agente tem de compreender que conjunto de ações o levaram a obter o maior número de recompensas, evitando as ações prejudiciais.

2.3 Processo de Decisão de Markov

A ideia que está na base da aprendizagem por reforço é de um agente que possui um conjunto de estados, ações sobre esses estados, uma probabilidade de transição e uma função de recompensa/*reward*. O objetivo desse agente é maximizar o valor das recompensas adquiridas, isto é, aprender uma boa estratégia de modo a ter o maior número de recompensas possível.

Trata-se, portanto, de um processo Markov, uma vez que satisfaz as suas propriedades, isto é, é um processo estocástico (representa a evolução de um sistema ao longo do tempo, ou seja, é uma sequência de estados aleatórios) e o próximo estado apenas depende do estado atual e não da sequência de estados que ocorreram até então. Tal como Markov afirmou: *"The future is independent of the past given the present"*, o estado futuro não necessita da informação do passado, apenas do estado atual em que o agente se encontra. Assim, o agente está num determinado estado e escolhe fazer uma certa ação que o leva a um novo estado. Ao agente é atribuído um valor de recompensa consoante o sucesso da ação escolhida. Desta forma, iterando este passo o número de vezes que forem necessárias, o agente acabará por aprender uma estratégia de obter boas recompensas.

2.4 Política de Ações

Existem estados que são melhores do que outros, pois quando o agente realiza uma ação que o leva a um desses estados recebe uma recompensa maior. Por outro lado, existem outros estados que não são bons para o progresso do agente, logo é-lhe atribuído um valor baixo de recompensa, ou seja, recebe uma penalização pelo comportamento tomado. Como o objetivo do agente é maximizar o valor das recompensas adquiridas, evitando, assim, estados menos desejáveis, é necessário encontrar uma política de ações que vise a escolha da ação com a recompensa mais elevada.

Dentro de um conjunto de políticas possíveis, uma delas será a ótima, porém o agente não sabe qual ela é, logo terá de a descobrir. Para isso é preciso saber comparar políticas. Assim, é preciso que o agente interaja com o ambiente em que se encontra, de modo a formular uma estratégia de ações.

Deste modo, uma política de ações tem como função indicar qual a ação que o agente deve tomar num determinado estado, isto é, modela o seu comportamento, mapeando estados em ações.

2.5 Função valor

As abordagens de função valor têm como objetivo encontrar uma política que maximize a recompensa, considerando um conjunto de recompensas expectáveis

para uma determinada política. Assim, a função valor **estima** o quão benéfico (que recompensas serão expectáveis no futuro) será para um agente realizar um determinado tipo de ação num determinado estado.

As funções valor dividem-se em dois ramos: as funções valor de estado e as funções valor de ações, sendo elas representados por $V^{\pi}(s)$ e $Q^{\pi}(s,a)$ respetivamente. O símbolo π está presente, pois as funções valor são sempre condicionadas por uma determinada política de ações.

Função valor de estado A função valor de estado, $V^{\pi}(s)$, representa a recompensa a receber num determinado estado inicial s e as recompensas a receber seguindo uma política de ações . O valor da função valor de estado é dado pela seguinte fórmula:

$$V^{\pi}(s) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots \quad (1)$$

em que γ representa o fator de desconto e r_t a recompensa r num determinado instante de tempo t.

Função valor de ações A função valor de ações, $Q^{\pi}(s,a)$, representa a recompensa a receber num determinado estado inicial s, através de uma ação a e as recompensas a receber seguindo uma política de ações π. O valor da função valor de ações é dado pela seguinte fórmula:

$$Q^{\pi}(s,a) = r(s,a) + \gamma V^{\pi}(s') \quad (2)$$

onde s' representa o próximo estado do agente resultante da ação realizada.

Resumindo, o objetivo continua a ser aprender uma política de ações ótima que irá maximizar tanto a função valor de estado, como a função valor de ações, podendo ser ambas igualmente utilizadas.

2.6 *Temporal Difference Learning*

Os métodos de aprendizagem *temporal difference* são métodos utilizados para estimar as funções valor, ocorrendo estimativas da recompensa final para cada estado e ação do mesmo. Caso os valores da função valor fossem calculados sem estimativas, o agente seria obrigado a esperar até á última recompensa ter sido recebida antes de atualizar os valores relativos a cada ação e estado. Esta opção obrigaria a, quando recebida a última recompensa, atualizar todos os valores considerando o caminho seguido. Estes métodos podem ser divididos em aprendizagem *on-policy* e aprendizagem *off-policy*.

Os métodos *on-policy* aprendem o valor da política que será usada para efetuar decisões. As funções valor são atualizadas utilizando resultados das ações executadas por uma determinada política. Baseiam-se em experiência, sendo SARSA um algoritmo que implementa este tipo de métodos.

Os métodos *off-policy* aprendem diferentes políticas através de estimativas. Algoritmos que implementem este tipo de métodos, como é o caso do algoritmo Q-learning, atualizam o valor da função valor utilizando ações hipotéticas, que ainda não foram experimentadas, contrastando com os métodos *on-policy* que se baseiam fortemente na experiência. Esta capacidade significa que estes algoritmos conseguem fazer a distinção entre exploração e controlo, enquanto que os restantes são obrigados a explorar para obter o resultado desejado.

Antes de avançar para a explicação dos algoritmos será necessário definir um episódio. Um episódio consiste numa sequência de estados, ações e recompensas que terminam num estado terminal.

Algoritmo Q-learning Através do algoritmo Q-learning é possível determinar a política de ações ótima conhecendo unicamente o valor ótimo da função valor de ações (não será necessário conhecer a função de transição de estados nem o valor das recompensas). Assim sendo, temos de aprender a função valor de ações ótima sem considerar o valor da função valor de estado. Deste modo, a fórmula será a seguinte:

$$Q^{\pi}\ (s_t, a_t\ ) = r(s_t, a_t\ ) + \gamma max_{a'}[Q(s_{t+1}, a')]. \tag{3}$$

Consequentemente, podemos definir o algoritmo da seguinte forma:

1. Para cada estado s e ação a, inicializar Q^{π}(s,a)
2. Repetir para cada episódio:
   (a) Observar o estado atual s;
   (b) Escolher uma ação a e executá-la;
   (c) Repetir para cada passo do episódio:
      i. Executar a ação a e observar o próximo estado s' e a recompensa r;
      ii. Atualizar Q(s,a)=Q(s,a) + α[r+γmax$_{a'}$ (Q(s',a'))-Q(s,a)]
      iii. s torna-se s'
   (d) Até s ser estado terminal.

Em que α é a taxa de aprendizagem (varia de 0 a 1, em que 0 não aprende e 1 aprende muito rápido) e γ é o fator de desconto (este fator faz com que as recompensas futuras valham menos que as imediatas).

Algoritmo SARSA O algoritmo SARSA difere do Q-learning pelo facto da recompensa máxima para o próximo estado não ser instantaneamente utilizada para atualizar o valor da função de valor de ação. O algoritmo pode ser definido da seguinte forma:

1. Para cada estado s e ação a, inicializar Q^{π}(s,a)
2. Repetir para cada episódio:
   (a) Observar o estado atual s;
   (b) Escolher uma ação a e executá-la;
   (c) Repetir para cada passo do episódio:

i. Executar a ação a e observar o próximo estado s' e a recompensa r;
ii. Escolher uma ação a' do próximo estado;
iii. Atualizar Q(s,a)=Q(s,a) + α[r+γQ(s', a')-Q(s,a)]
iv. s torna-se s' e a torna-se a';

(d) Até s ser estado terminal.

A escolha de uma ação em ambos os algoritmos obriga a uma questão. Qual deve ser a próxima ação a ser escolhida? Deve-se usufruir dos valores conhecidos ou explorar valores desconhecidos? Este problema é conhecido como *exploration vs exploitation*.

2.7 *Exploration vs exploitation*

Imagine que um indivíduo deseja ficar rico através da compra de raspadinhas. Que ação é que ele deve tomar? Comprar sempre a mesma raspadinha ou alternar entre os diferentes tipos de raspadinhas (considerando que o custo das raspadinhas é igual)? Este problema pode ser visto como um problema *exploration vs exploitation*. O indivíduo pode escolher a ação que possui o maior valor Q(s,a), *exploitation*, sendo neste caso continuar com a mesma raspadinha, ou pode escolher uma ação aleatória, de modo a que o seu valor Q(s, a) seja atualizado (*exploration*), sendo este caso alternar entre as diversas raspadinhas.

A escolha de qualquer uma das estratégias será feita considerando o problema de que uma ação pode ser mais benéfica agora, mas prejudicial a longo plano (ou vice-versa). Assim sendo, a escolha irá depender do ambiente, das ações já tomadas e das ações que ainda faltam tomar. Exemplificando, podemos concluir que se faltar unicamente uma ação, não será utilizada a *exploration*, mas sim a *exploitation*, visto que escolhe a ação com maior valor de Q(s,a).

Apesar de parecer um problema, existem três maneiras de resolver este dilema, utilizando os métodos ε-*greedy*, ε -*soft* e *softmax*.

2.8 Limitações

Com o algoritmo Q-Learning, a procura por uma política ótima é mais flexível, tendo em conta que existe a capacidade de aprender enquanto se muda de políticas. No entanto, este algoritmo pode ser mais lento na sua execução. Também temos de ter em conta que o conhecimento é pouco profundo, pois existe uma restrição na capacidade de aprender, uma vez que não se possui conhecimento prévio sobre o ambiente.

Em relação ao algoritmo SARSA, que se baseia principalmente na exploração, se se começar com uma boa política, então este tipo de algoritmo pode ser adequado, porém pode não explorar tão bem como outras políticas.

2.9 Ferramentas de Desenvolvimento

Existem várias ferramentas de desenvolvimento que permitem trabalhar com aprendizagem por reforço, entre elas, destacam-se as seguintes:

- TensorFlow, que permite treinar agentes para realizar certas tarefas;
- OpenAI Gym, que é um conjunto de ferramentas que visam o desenvolvimento e comparação de vários algoritmos de RL;
- Rllab, compatível com OpenAI Gym;
- Keras-RL, que implementa em Python vários algoritmos, como Deep Q-Learning;
- BURLAP, desenvolvido na Universidade de Brown, implementa processos de decisão Markovianos, assim como vários algoritmos de RL;
- PyBrain, uma biblioteca de Python, que implementa algoritmos de RL básicos (Q-Learning e SARSA), como também outros mais avançados;
- RL4J (Reinforcement Learning for Java), uma biblioteca que oferece implementações de vários algoritmos de RL.
- Project Malmo, que é uma plataforma desenvolvida pela Microsoft para experimentação e investigação de algoritmos de Inteligência Artificial (como, por exemplo, a aprendizagem por reforço) desenvolvida tendo por base o jogo Minecraft.
- DeepMind Lab, que é uma plataforma, semelhante a um jogo 3D, construída para investigação na área de aprendizagem por reforço.

2.10 Soluções no Mercado

A aprendizagem por reforço é usada por várias indústrias como forma de resolver vários problemas. Por exemplo, a maior empresa do mundo que se dedica à criação e desenvolvimento de robots, a Fanuc, usa o RL (*reinforcement learning*) para ensinar aos robots várias tarefas. Estes são especialmente usados nas fábricas, onde as ocupações tendem a ser muito monótonas e repetitivas. Assim, através da tentativa-e-erro, o robot ganha conhecimento e treina-se a si próprio até executar o trabalho rapidamente e com muita precisão. Várias empresas de vendas online ou até mesmo hipermercados usam este tipo de robots para organizar e entregar corretamente os seus produtos. Na mesma área, o algoritmo Q-Learning é muitas vezes usado para definir o melhor caminho que alguém deve fazer de forma a entregar as encomendas da maneira mais eficiente possível.

Também no setor financeiro, a aprendizagem por reforço é muito utilizada, uma vez que se revela bastante robusta para otimizar os objetivos financeiros de alguma instituição. Com essa ideia, nasceu a Pit.AI, que usa a Inteligência Artificial, mais especificamente a aprendizagem por reforço, para desenvolver as melhores estratégias de investimento e capitalização.

Por fim, este tipo de aprendizagem também pode ser usado para personalizar e melhorar significativamente os sistemas de recomendação atuais presentes em vários websites tanto sob a forma de anúncios como de produtos sugeridos. Assim, graças ao *feedback* por parte dos utilizadores, o RL é capaz de fazer várias correções a estes sistemas.

3 Algoritmos genéticos

3.1 História

O interesse pela área de computação evolucionária começou a surgir nos meados do século XX, nas décadas de 50 e 60 em que diversos cientistas decidiram estudar os sistemas, os organismos neles inseridos e a sua evolução ao longo do tempo com a esperança de encontrar aí uma solução para vários problemas de engenharia.

Durante esse período foram apresentadas várias estratégias evolucionárias, no entanto só no fim da década de 60 é que o termo algoritmo genético surgiu. Este foi inventado por John Holland e desenvolvido principalmente na Universidade de Michigan, nos Estados Unidos. Ao contrário do que tinha sido desenvolvido até então, Holland não pretendia inventar algoritmos para solucionar problemas específicos, mas sim estudar o processo que ocorre na Natureza e adaptá-lo para a área computacional.

3.2 Contexto

Desde sempre que se olhou para os processos naturais como guias para resolver problemas computacionais, assim os algoritmos genéticos, pertencentes aos algoritmos evolucionários, são inspirados nos processos de evolução de seres vivos que ocorrem na Natureza. Com o objetivo de obter a melhor solução para um dado problema fazem uso de outros fenómenos biológicos, tais como, seleção, combinação e seleção.

Este tipo de algoritmos tira partido da informação passada para direcionar a sua procura por uma solução melhor, de forma a obter um melhor desempenho. Dado que os métodos tradicionais de otimização e procura não são capazes de encontrar uma solução ótima quando o problema em mãos tem um espaço de procura demasiado complexo, os algoritmos genéticos são robustos o suficiente para a partir do mesmo search space encontrar uma solução que garantirá um bom desempenho.

Deste modo, os algoritmos genéticos são usados nas mais diversas áreas, tais como o processo de imagens, trajetória do movimento de robots, criação de peças para aviões, análise de ADN e obtenção do caminho mais curto (problema do caixeiro-viajante).

3.3 Um Bocadinho de Biologia...

Tendo em conta que os algoritmos genéticos são inspirados na evolução dos seres vivos, faz sentido esclarecer algumas noções básicas fundamentais para entender a metodologia deste tipo de algoritmos, que será explicada mais adiante.

Cada organismo é distinto dos restantes e essa diferença vai até ao nível dos genes, que são responsáveis pelas suas características, como, por exemplo, a cor dos olhos. Um conjunto de genes designa-se por cromossoma.

A evolução dos organismos é garantida pela sua reprodução, isto é, pela combinação de genes de ambos os pais. O organismo resultante terá uma parte dos genes de cada um dos organismos que lhe deram origem. No entanto, por vezes podem ocorrer mutações de genes, que dão origem a novas características que não são herdadas. Este último fenómeno garante a diversidade de organismos a cada geração.

A evolução na qual os algoritmos genéticos se baseiam é, também, feita através da seleção dos indivíduos mais aptos a sobreviverem. Este princípio foi proposto por Charles Darwin em 1898 no livro A Origem das Espécies, no qual apresenta a Teoria da Evolução que defende o processo de seleção natural entre organismos. Assim, tal como na Natureza, nos algoritmos genéticos vai existir uma competição, que beneficiará aqueles que têm mais condições para continuar as gerações futuras.

3.4 Metodologia

Um algoritmo genético procura imitar os processos naturais de evolução de forma a encontrar a melhor solução possível para um certo problema proposto. Ao longo desta secção serão abordados cada um dos seguintes temas, que se podem analisar na figura abaixo, que fazem parte da estrutura básica de um algoritmo genético.

Fig. 2. Estrutura de um algoritmo genético

3.5 Inicialização

Os algoritmos genéticos simulam os processos de seleção natural de seres vivos que ocorrem de geração em geração de forma a poder encontrar a melhor solução

possível para um determinado problema. Cada geração consiste, portanto, numa população de indivíduos diferentes entre si, análogos aos cromossomas presentes no ADN dos organismos. Sendo que cada um destes indivíduos representa uma solução candidata ao problema presente é preciso, então, codificá-la corretamente. Normalmente, uma solução é uma *string* de dígitos binários, cada dígito correspondendo a um gene do cromossoma.

Deste modo, o espaço de procura ou search space contendo as possíveis soluções terá o seguinte aspeto:

Fig. 3. Estrutura do espaço de procura

Este é, então, o espaço de procura ou população inicial. Cada cromossoma ou solução codificada em código binário é gerado aleatoriamente. São estes os indivíduos que irão fazer parte do processo de evolução, ou seja, irão competir e reproduzir-se entre si. Alguns irão sobreviver entre gerações, enquanto outros irão ser postos de parte.

3.6 Criação da Nova Geração

De forma a perceber quais são as soluções mais aptas a resolverem o problema é preciso definir uma função objetivo para calcular os valores de fitness de cada uma delas. A definição de uma função objetivo adequada pode ser a parte mais complicada da implementação deste tipo de algoritmo, uma vez que depende do problema em questão e é preciso que os seus cálculos sejam computacionalmente rápidos, pois serão repetidos várias vezes ao longo do algoritmo. Ela considera cada uma das soluções candidatas e atribui-lhe um valor que representa a sua capacidade de resolver o problema, ou seja, indica quais são os melhores e os piores indivíduos daquela população.

As soluções que obtiverem melhores valores de fitness terão maior oportunidade de se reproduzirem, recombinando os seus genes, gerando, assim, novas soluções que poderão ser melhores do que os pais, dado que herdam algumas das suas características. Até podem existir certas soluções que, graças ao seu fitness score passarão automaticamente para a geração seguinte – estas dizem-se que

fazem parte do grupo de elite. Vejamos assim: dentro de uma população existe uma pequena fração de soluções ("o 1% da população") que tem a sorte de possuir um valor alto de fitness, logo são altamente competentes para resolverem o problema, sendo, então, candidatas fortes a serem a solução ótima. Por esta razão, terão maior oportunidade de se reproduzirem e produzirem mais soluções que potencialmente serão ainda melhores do que as que lhe deram origem. Além disso, poderão sobreviver para integrarem a nova geração acabada de se formar.

No entanto, existem outras soluções que não possuem valores de fitness equiparados aos restantes. Estas serão menos capazes de resolverem satisfatoriamente o problema em causa, logo terão menos oportunidade de se reproduzirem e não farão parte da geração seguinte. Desta forma, reduz-se o número de soluções menos aptas na nova população. Estas soluções serão inevitavelmente substituídas por outras mais capazes.

Com este processo, análogo ao da seleção natural e à evolução das espécies, geram-se sucessivamente gerações cada vez melhores e mais próximas da solução ótima do problema. Todavia, se não se introduzir diversidade dentro da população, esta tende a convergir para uma solução ótima local, não chegando ao seu potencial máximo. Graças a certas mutações em alguns genes, algumas das soluções produzidas terão características novas e diferentes das soluções existentes na população. Caso estas mudanças se revelem benéficas para o valor do fitness, a procura da solução ótima será redirecionada para outro domínio do espaço de procura.

Deste modo, existem três tipos de soluções que podem ser encontradas numa nova geração:

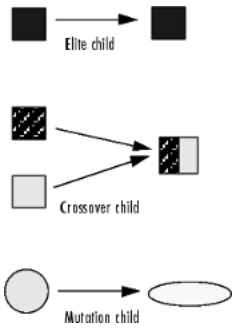


Fig. 4. Tipos de soluções numa dada população

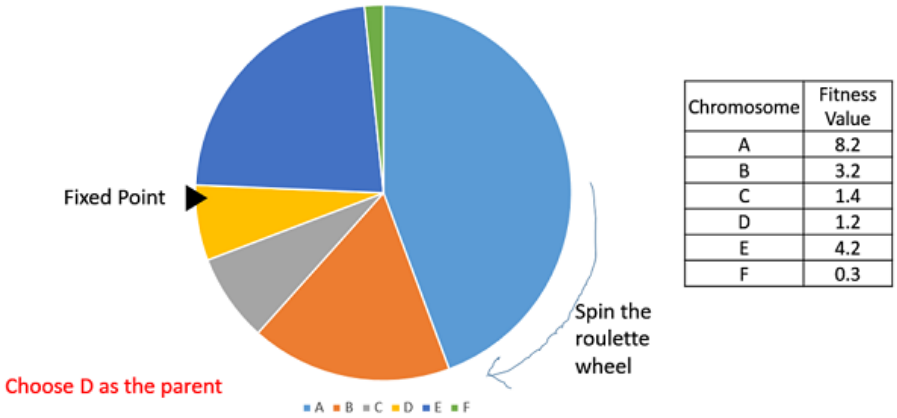
Tendo em conta que as soluções que fazem parte da elite já foram acima explicadas, proceder-se-á a uma maior clarificação da geração de soluções através de crossover ou recombinação de genes e ao fenómeno da mutação, assim como ao da seleção das soluções para reprodução.

3.7 Seleção Através da Roleta

Já foi referido que as soluções mais aptas a resolverem o problema, isto é, que possuem valores altos de fitness, têm mais oportunidades de se reproduzirem com outras soluções do que as que têm um valor de fitness mais baixo. Esta oportunidade é implementada recorrendo a um método probabilístico, análogo a uma roleta.

Considerando, então, a roleta, aos indivíduos com maior fitness será atribuída uma maior probabilidade. Seguindo esta lógica, os cromossomas menos aptos a resolverem o problema de uma forma ótima terão a si atribuída uma probabilidade menor.

Assim, a escolha de quais os cromossomas que se devem reproduzir é feita através da rotação da roleta. Desta maneira, ao mesmo tempo que favorecemos as melhores soluções, também damos uma oportunidade às soluções menos aptas.



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Fig. 5. Roleta da seleção das soluções-pai

Tal como podemos verificar na Figura acima representada, temos uma população de seis cromossomas (ou soluções). A cada uma delas foi atribuído um valor de fitness, portanto as soluções mais aptas possuem um valor mais alto de fitness, ao contrário das que não são tão eficientes na resolução do problema.

Deste modo, podemos analisar que a solução identificada pela letra A é a solução mais habilitada, ao passo que o cromossoma F é claramente o que tem menos capacidades. Por esta razão, a solução A (distinguida pela cor azul claro) tem a si atribuída uma porção maior da roleta (ou *pie chart*), ou seja, tem maior probabilidade de ser escolhida e à solução F (cor verde) corresponde a menor "fatia" da roleta, isto é, tem a menor probabilidade de todos os cromossomas da população.

Fixando um ponto e girando a roleta, podemos aferir que a solução escolhida para ser um dos pais de uma nova solução é a D (cor amarela), que não é de todo

a que tem a maior probabilidade. De facto, a solução D é a que tem o segundo valor de *fitness* mais baixo.

Como se pode ver, este método é importante para contribuir para a diversidade da população, uma vez que se apenas as soluções aptas fossem escolhidas para se reproduzirem entre si, a cada geração existiria cada vez menos variedade de soluções. O algoritmo corria, assim, o risco de localizar uma solução que corresponderia a um ótimo local e não a um ótimo global.

3.8 *Crossover*

Escolhidas as soluções que se irão reproduzir, passamos ao próximo passo: o crossover de genes, isto é, a sua recombinação numa nova solução. É aqui que se conjugam os genes de duas soluções e que o cromossoma resultante herda algumas das características dos pais.

Primeiramente é preciso escolher aleatoriamente o ponto em que se fará o crossover. De seguida, procede-se à recombinação dos genes numa nova solução. Tomemos em consideração este par de soluções que se irão reproduzir: "Olá, munpo!" e "Ulá, mundo!". Escolhendo o ponto de crossover como sendo a quarta posição da string obtemos as seguintes soluções-filho: "Olá, mundo!" e "Ulá, munpo!". Como se pode ver, um dos filhos herdou as características boas de ambos os pais, enquanto que outro herdou apenas as características menos favoráveis.

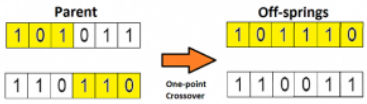


Fig. 6. *Crossover* de genes

Deste modo, as soluções resultantes da reprodução herdarão genes de ambos os pais. Se estes tiverem um valor de fitness alto é provável que os filhos sejam melhores que as soluções de origem. Assim, de geração em geração obtém-se populações cada vez mais aptas a resolverem o problema.

No entanto, mesmo que a seleção de soluções para reprodução engloba as mais e as menos habilitadas, a diversidade da população eventualmente irá tornar-se cada vez mais escassa, o que põe em causa a obtenção de bons resultados por parte do algoritmo.

3.9 Mutação

Existe uma pequena possibilidade de que a codificação de um bit do cromossoma seja mudada. A isto chama-se mutação de genes. Tomando o exemplo anterior,

uma das soluções geradas poderia sofrer uma mutação e, em vez de apresentar a forma "Olá, mundo!" seria "Olé, mundo!". Ocorreu, então, uma mutação na terceira posição da string, pois o gene "á" sofreu uma transformação e passou a ser "é".

Fig. 7. Mutação de um gene num cromossoma

A mutação, uma simples mudança aleatória num dos genes num determinado cromossoma da população, irá promover a diversidade das futuras gerações. Como se pode verificar a simples transformação de um bit com valor 1 para um bit com valor 0 irá trazer para a população soluções com características diferentes, impulsionando, então, a procura pela solução ótima numa direção diferente da atual.

3.10 *Exploitation vs Exploration*

Uma das grandes dificuldades que os algoritmos genéticos enfrentam é o equilíbrio necessário entre as duas vertentes de tratar o espaço de procura de soluções: *exploitation vs exploration*.

A primeira engloba a seleção e a recombinação, que, apesar de chegarem a uma solução, raramente é a solução ótima. Esta tende a ser um ótimo local, ou seja, é uma solução razoável para o problema, mas não é a ótima. Esta convergência premeditada deve-se à falta de diversidade que se sente com o passar das gerações. Não há nenhum fator externo que introduza novas características na população, logo, ao fim de algumas iterações do algoritmo, todas as soluções da população terão praticamente as mesmas características.

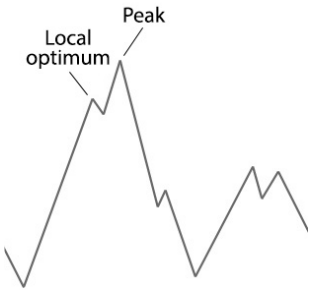


Fig. 8. Ótimo local e ótimo global

A segunda, por outro lado, prioriza a seleção e a mutação, que previne uma convergência prematura, porém tende a não favorecer as melhores soluções, desperdiçando-se tempo em soluções que não são tão aptas – andamos a explorar o espaço de procura aleatoriamente, sem nenhuma direção definida.

Numa implementação de um algoritmo genético, é preciso ter o cuidado de fazer uma combinação das duas vertentes, ou seja, explorar eficientemente o mais possível do espaço de procura, mas também tirar partido da informação das gerações passadas.

3.11 Condições de paragem

Tendo uma população inicial que já foi submetida aos processos evolutivos mencionados acima, a questão que permanece é: quando é que sabemos que o algoritmo chegou à solução ótima?

Existem várias maneiras de definir as condições de paragem, como, por exemplo, fixar um número máximo de gerações (o algoritmo termina quando atingir esse limite) ou a função objetivo que atribui os valores de fitness às soluções atinge também um valor pré-definido. No entanto, também sabemos que podemos terminar o algoritmo quando de geração para geração não se regista melhorias significativas, isto é, chegamos a ponto em que a geração seguinte de soluções não oferece nada de novo em relação à geração anterior.

3.12 Ferramentas de Desenvolvimento

Dado que existem várias ferramentas de desenvolvimento de algoritmos genéticos, vamos apresentar apenas um conjunto das que são mais usadas e aconselháveis. Deste modo, temos:

- MatLab oferece uma ferramenta de trabalho, a Global Optimization Toolbox;
- GPdotNet, uma ferramenta em C que permite modelar, prever e otimizar problemas;
- JGAP, um pacote de Java que permite programar algoritmos genéticos;
- ECJ (Evolutionary Computation Journal) desenvolvido pelo laboratório de computação evolucionária na Universidade George Mason, na Virgina, Estados Unidos. Está escrito em Java e oferece várias funcionalidades para lidar com algoritmos genéticos;
- JCLEC, também desenvolvido em Java, é um sistema de software que fornece mecanismos para programar algoritmos genéticos;
- Open BEAGLE em C++ é uma framework que permite programar algoritmos genéticos através de árvores;

3.13 Soluções de mercado

As aplicações dos algoritmos genéticos no mundo real são imensas: variam desde a conceção de peças para carros profissionais de corrida até prever o comportamento dos mercados financeiros.

Também podem ser usados para definir a trajetória de movimento de um robot, ou seja, ajudam a planear o caminho que o braço de um robot deve fazer de um ponto A para um ponto B. Ainda em relação aos robots, o design dos braços destes são otimizados recorrendo ao mesmo tipo de heurística. Também a área militar é abrangida, pois estes algoritmos já foram implementados na simulação do voo de um avião que se encontra a tentar fugir de um míssil. Como se pode verificar, os algoritmos genéticos são muito utilizados no planeamento. Mais um exemplo: eles são usados para resolver problemas de planeamento de tarefas nos laboratórios, em que cada atividade pode afetar outras, caso não seja concluída atempadamente.

A área da medicina também usa algoritmos genéticos, principalmente nos raios-X e nos diagnósticos de vários tipos de cancro, como o cancro da mama, devido à alta taxa de diagnósticos falhados. Além disso, em relação à segurança, estes algoritmos tanto podem ser usados para encriptar dados, assim como para os descodificar.

Por fim, são igualmente utilizados numa área mais atual - o gaming -, pois são criados jogos que usam os algoritmos genéticos para incorporarem as estratégias mais bem-sucedidas dos jogos anteriores no novo jogo em fase de desenvolvimento.

4 Árvores de decisão

4.1 Descrição

Uma árvore de decisões consiste em ramificações em árvore nas quais os nós intermédios são testes relativos aos valores de atributos, os ramos são os possíveis valores que esses atributos poderão tomar e as folhas consistem no valor da classe (decisões sobre a previsão). A classe é uma variável dependente dos atributos (variáveis independentes). Cada objeto a ser classificado deve poder ser expresso em termos de atributos e classes, sendo imperativo que devem existir mais objetos que classes, visto que, se tal não acontecesse, a árvore de decisão apresentaria uma raiz com tantos ramos quanto classes existentes.

Ignorando momentaneamente o processo de construção da árvore de decisão (será explicado numa fase posterior), consideremos a seguinte árvore de decisão, contruída a partir dos dados ilustrados na seguinte tabela. A árvore surgiu com a dúvida de um determinado indivíduo sobre se deveria ir a um determinado concerto, sendo os dados referentes a experiências anteriores de idas a concertos. As colunas pintadas a azul são referentes aos atributos, enquanto que a coluna verde refere-se à classe (P representa um valor positivo, ou seja, ir ao concerto, enquanto que N representa um valor negativo).

Banda favorita	Preço dos bilhetes	Acompanhado	Classe
Sim	12	Não	P
Sim	1000	Não	P
Não	8	Não	P
Não	11	Não	N
Não	20	Sim	P

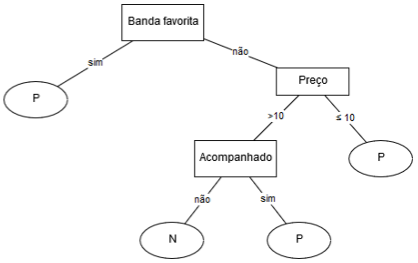


Fig. 9. Exemplo dos dados e respetiva árvore de decisão

Uma característica específica das árvores de decisão é que são representados por regras fáceis de entender. Analisando a árvore de decisão anterior, podemos deduzir a seguinte regra:

$$banda = nao, preco > 10, acompanhado = sim => classe = P \qquad (4)$$

O processo de construção de uma árvore de decisão (ou classificador) pode ser, muito sucintamente, descrito pelo seguinte diagrama.

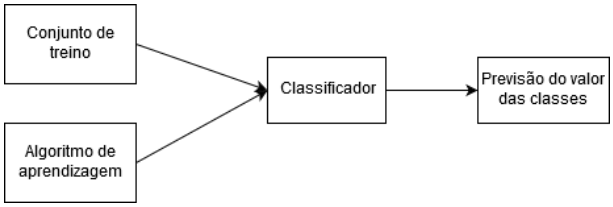


Fig. 10. Processo de construção de uma árvore de decisão

Através deste diagrama conseguimos identificar duas utilizações das árvores de decisão, sendo elas denominadas por modelação preditiva e modelação descritiva.

Na modelação descritiva, após ter sido construída a árvore através dos dados de treino, podemos usar o classificador para analisar que atributos melhor classificam uma classe. Assim sendo, temos uma classe queremos saber quais o tipo de atributos que a caracterizam e a sua frequência. Exemplificando com o exemplo acima, podemos ver que se a banda preferida do indivíduo tocar, ele irá sempre ao concerto.

Com a modelação preditiva, é possível utilizar o classificador para prever os valores das classes num conjunto de teste em que os valores de classe sejam desconhecidos. Exemplificando, se neste exato momento, o indivíduo se encontrar numa situação em que a banda favorita dele não vai estar no concerto, mas os bilhetes custam 5 euros e pode ir acompanhado de um grupo de amigos, ele pode percorrer a árvore e concluir que deve ir ao concerto.

Assim, as árvores de decisão podem ser facilmente utilizadas devido à sua simplicidade de interpretação. Garantindo a hierarquia definida pelo classificador, começa-se pela raiz efetuando perguntas sobre os atributos para depois avançar ao longo da árvore até atingir um valor de classe.

4.2 Aprendizagem

A aprendizagem de árvores de decisão, ou construção de árvores de decisão, caracteriza-se por ser uma aprendizagem indutiva, em que através de um conjunto de treino e de um algoritmo de aprendizagem constrói-se um classificador. A estratégia passa pela aprendizagem não incremental através de exemplos. Assim, a essência da indução é construir uma árvore que consiga identificar tanto objetos que estejam representados no conjunto de treino, tal como aqueles que não estão representados, sendo isto atingido se a árvore conseguir identificar relações significativas entre os valores dos atributos e das classes.

Apesar da facilidade de interpretação das árvores de decisão, a sua construção não é tão literal e apresenta dificuldades, tal como o mesmo conjunto de dados pode resultar em árvores diferentes (resultante da escolha de cada atributo para nó da árvore), ou os elevados gastos computacionais que impedem a obtenção da árvore ótima (quanto maior for a quantidade de atributos e classes, maiores são as possíveis árvores existentes). Apesar destes obstáculos, existem algoritmos capazes de lidar bem com eles e obter árvores de decisão próximas do ótimo e em tempo real.

Resumindo, o objetivo passará por construir árvores simples, visto que são expectáveis que consigam identificar mais objetos fora do conjunto de treino.

4.3 Algoritmo TDIT

O algoritmo TDIT (*Top Down Induction of Decision Trees*), usado como base nos algoritmos ID3, C4.5 e CART, constrói a árvore de decisão dividindo os casos sucessivamente de acordo com os valores dos atributos. Há três hipóteses que caracterizam este algoritmo, sendo elas:

1. O conjunto de treino apresenta objetos de uma única classe, sendo a árvore um único nó folha que identifica essa classe;
2. O conjunto de treino não apresenta objetos, sendo a árvore um nó folha. No entanto, a identificação da classe tem de ser determinada através do contexto do problema, visto que não existem objetos no conjunto de treino;
3. O conjunto de treino apresenta diversos objetos pertencentes classes diferentes. A construção da árvore será feita escolhendo um atributo que melhor divide o conjunto de treino e dividindo o conjunto mediante esse atributo. Este passo repete-se recursivamente até todos os atributos tenderem para classes únicas.

No entanto, parece que este algoritmo obriga à pergunta “Qual é o melhor atributo a escolher para cada nó da árvore?”.

4.4 Escolha do melhor atributo

A escolha do atributo para cada nó da árvore é crucial, pois pretendemos uma árvore balanceada com os casos corretamente divididos, ou seja, para cada nó folha todos os casos sejam relativos à mesma classe. Assim, pretende-se associar a cada nó o atributo que possui mais informação (entropia).

Há diversas formas de calcular a melhor escolha do atributo para um determinado nó, no entanto, serão apresentadas duas, sendo elas o ganho de informação e a razão do ganho. Como ambas utilizam o conceito de entropia, este será o primeiro a ser explicado.

Entropia Entropia é um indicador de incerteza. Quanto maior for a entropia, maior é a informação contida. A entropia é definida pela seguinte equação:

$$Entropia(S) = \sum_{classe} -p_{classe} \times \log_2 p_{classe} \tag{5}$$

em que p_{classe} é a proporção entre os elementos da classe e os elementos de S.

Ganho de informação O ganho de informação utiliza a entropia como grau de impureza. De modo a determinar a qualidade do teste (divisão dos atributos) comparamos o grau de entropia do nó-pai (antes da divisão) com o grau de entropia dos nós-filhos. O atributo que gerar a maior diferença entre estes dois valores será o escolhido. O ganho é calculado através da seguinte equação:

$$Ganho(S, A) = Entropia(S) - \sum_{v \in valores(A)} \frac{\#S_v}{\#S} \times Entropia(S_v) \tag{6}$$

em que S é o conjunto pai, A é o atributo a testar e Sv corresponde a um nó filho.

Apesar de ser uma boa ferramenta para escolher o melhor atributo, o ganho apresenta um problema quando são considerados atributos irrelevantes que identificam unicamente uma classe (o caso de identificadores únicos, tais como, por exemplo, o número de identificação fiscal). No caso da existência destes atributos, ou não os consideramos na construção da árvore, ou podemos ter uma árvore com tantos ramos quantos valores desses atributos existirem (isto é, um conjunto de dados que apresente 20 alunos identificados pelo número de aluno, terá 20 ramos) e para cada um desses ramos uma classe. Segundo a fórmula o ganho é máximo, no entanto a árvore é inútil.

Razão do ganho A razão do ganho de informação é uma razão entre o ganho de informação e a entropia do nó considerado. A fórmula da razão do ganho de informação é:

$$razaodoganho(no) = \frac{ganho(no)}{entropia(no)} \tag{7}$$

Esta medida apresenta resultados qualitativamente superiores ao ganho de informação.

4.5 Dados contínuos

Se os atributos de um conjunto de dados apresentarem valores contínuos, as árvores podem tornar-se demasiados grandes e complexas. Este problema pode ser resolvido discretizando os dados em intervalos de valores, por exemplo, criar intervalos onde é preservado o valor. Se considerarmos o exemplo inicial do indivíduo que considerava ir a um concerto, podemos ver que o valor do atributo preços dos bilhetes é contínuo, no entanto, o ramo apresenta um teste sobre um intervalo de valores, tornando a árvore muito mais simples, comparativamente a ter ramos para cada um dos valores contínuos.

4.6 Dados que apresentam valores nulos

Num conjunto de dados existem casos em que os valores de atributos ou classe estão em falta. Esta situação pode ser resolvida de duas maneiras:

- Substituir os valores nulos. Esta substituição pode ser feita analisando o contexto do problema ou substituindo pelos valores mais comuns nesse conjunto de dados;
- Proceder normalmente com a indução da árvore de decisão até surgirem valores nulos. Quando tal acontecer, é efetuado um cálculo de distribuição de probabilidades através de um formalismo Bayesiano. Assim, percorre-se cada ramo e calcula-se a classificação. A classe com maior probabilidade é a escolhida.

4.7 Dados que apresentam erros

A indução da árvore de decisão geralmente apresenta dificuldades, visto que no mundo real os dados apresentam demasiadas inconsistências, por exemplo, os atributos podem ser baseados em medições subjetivas, ou simplesmente apresentarem valores errados. Este tipo de erros são denominados de ruído (noise) e, de modo a serem suportados, a aprendizagem tem de ser capaz de suportar estes atributos inapropriados e decidir, através de testes, se estes novos atributos contribuirão para a precisão da árvore de decisão.

4.8 *Pruning*

Considerando que as árvores de decisão trabalham com grandes conjuntos de dados, é comum acontecer um sobreajustamento das hipóteses no processo de aprendizagem, ou seja, quanto mais profundos forem os nós folha menos exemplos os cobrem. Este fenómeno é denominado de *overfitting*. Para combater este problema, utiliza-se *pruning*. *Pruning* (ou poda) é uma simplificação do modelo de previsão em que resulta uma árvore de decisão mais simples. Os métodos de pruning dividem-se em:

Pré-*pruning* Parar de expandir um ramo quando a informação se torna pouco fiável. O critério de paragem utilizado pode ser o ganho de informação. Define-se um valor de ganho mínimo e caso todas os ramos apresentem valores de ganho inferiores a esse valor estabelecido, poda-se a árvore.

Pós-*pruning* Deixar crescer a árvore até ao fim. No final, podar as sub-árvores pouco fiáveis. Calcula-se o erro caso a sub-árvore do nó considerado seja podada e caso não haja poda. Se a diferença for considerável, transforma-se a subárvore numa folha, ou seja, poda-se a árvore. Esta estratégia apresenta melhores resultados que o pré-*pruning*.

4.9 Principais algoritmos de construção de uma árvore de decisão

ID3, desenvolvido por Ross Quinlan Algoritmo recursivo e *greedy*. Procura sobre um conjunto de atributos o que melhore divide os exemplos, utilizando como método de melhor seleção o ganho de informação. Como foi o primeiro algoritmo a ser desenvolvido, apresenta diversas limitações, tais como não aceitar valores contínuos nem nulos, nem apresenta nenhum método de pós-*pruning*.

C4.5, desenvolvido por Ross Quinlan Neste algoritmo, Quinlan tentou resolver as limitações do ID3, apresentando as seguintes vantagens sobre o anterior:

- Lida com atributos contínuos, através da binarização (define-se um valor de corte e divide-se os exemplos, mediante se são maiores ou menos que o corte);
- Possibilita a representação de valores nulos;
- Utiliza a razão do ganho em vez do ganho de informação;
- Apresenta métodos de pós-*pruning*.

CART(*Classification and Regression Trees*), desenvolvido por Leo Breiman O algoritmo CART é um algoritmo que gera árvores binárias, podendo ser árvores de decisão (caso o atributo seja nominal) ou árvores de regressão (o atributo é nominal). Similarmente ao C4.5, o CART utiliza pesquisa exaustiva para definir os limites dos nós para dividir atributos contínuos. O CART tem uma enorme capacidade de detetar relações entre os dados, produzindo árvores mais simples e legíveis. Este algoritmo também implementa métodos de pós-*pruning*.

5 Ferramentas de desenvolvimento existentes

Existem inúmeras ferramentas de desenvolvimento de árvores de decisão disponíveis, sendo necessário fazer menção às seguintes:

- Package Scikit do Python: a linguagem de programação Python apresenta este pacote com inúmeros mecanismos de mineração de dados, incluindo as árvores de decisão.

– Package rpart do R: a linguagem R apresenta este pacote que permite criar árvores de decisão através do algoritmo CART.
– WEKA: o software criado na universidade de Waikato, em Hamilton na Nova Zelândia possui inúmeros métodos de implementação de classificadores, tais como árvores de decisão.

No seguinte curto exemplo, utilizaremos um dos ficheiros que vem na instalação do WEKA, hypothyroid.arff , carregando-o para memória clicando em "Open file". Como podemos ver na seguinte figura, é possível examinar quais os atributos pertencentes neste conjunto de dados, quantos são, quantos objetos existem neste conjunto de dados, entre outros.

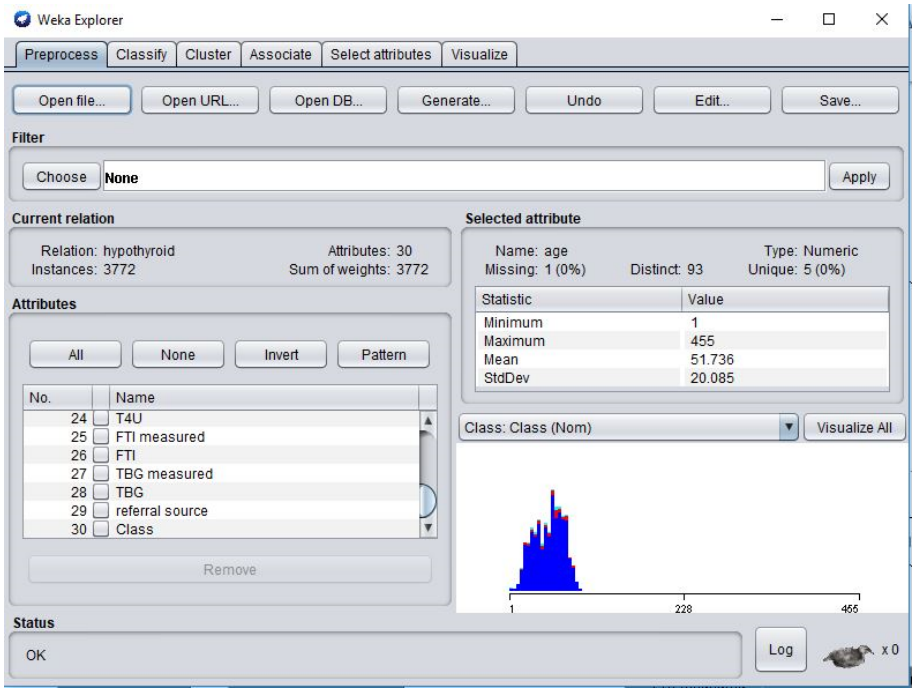


Fig. 11. Ecrã inicial no software Weka com a base de dados carregada

De seguida, podemos escolher o tipo de algoritmo que queremos utilizar para aprender o conjunto de dados. Na zona "Classifier", selecionamos "trees" e "J48". O algoritmo C4.5 apresenta o nome J48 no WEKA. Neste caso, vamos utilizar 66% do conjunto de dados para treino e o restante para testes. Depois de obtermos o classificador, é possível ver algumas estatísticas, como se pode verificar na Fig.12..

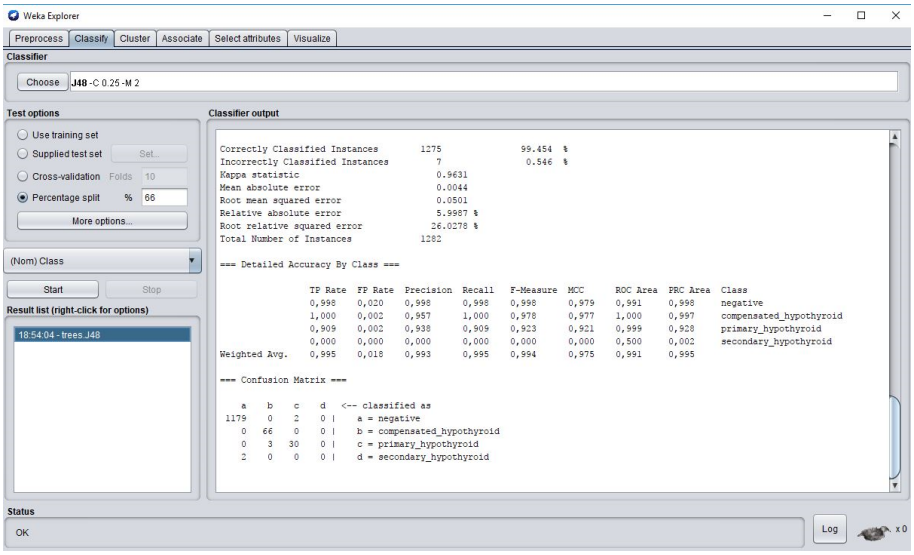


Fig. 12. Ecrã com os resultados após a árvore ter sido construída

Como podemos analisar, dos 1282 objetos do conjunto de dados, o classificador só classificou erradamente 7 objetos, com uma taxa de acerto de 99.454%. Logo, podemos concluir que este classificador conseguiu obter bons resultados.

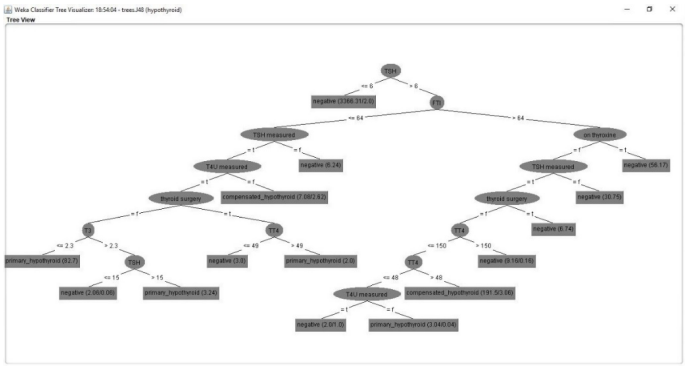


Fig. 13. Árvore de decisão criada pelo Weka

Concluindo, se se desejar ver a árvore de decisão desenhada, basta fazer clique com o botão direito em cima do resultado e clicar na opção “Visualize tree”, surgindo uma árvore do aspeto desta.

5.1 Soluções existentes no mercado

A GfK NORM possui a Shopper Decision Tree (lançada em 2016), que consiste numa árvore de decisão do consumidor. Tendo em conta as decisões de um consumidor, a Shopper Decision Tree, irá construir uma árvore de decisão, onde estarão quantificadas as diferentes categorias (podendo ser elas o sabor, a marca, ou a embalagem). A utilização desta árvore de decisão traz vários benefícios, tais como a otimização das prateleiras; capacidade de desenvolver novos produtos, caso as marcas rivais possuam o monopólio sobre os mesmos; distribuição dos produtos, sabendo que há produtos que têm de estar sempre disponíveis, devido à sua constante (e elevada) procura.

Segundo afirmações da própria empresa, a Westinghouse Electric's Water Reactor Division, conseguiu aumentar o lucro em mais de dez milhões de dólares por ano, através do uso de árvores de decisão.

6 Conclusão

Neste trabalho apresentamos uma visão sobre três tipos de aprendizagem diferentes que abordam a questão com pressupostos e metodologias distintas, mas que, no entanto, têm um objetivo comum: tornar um sistema inteligente.

Cada um dos temas desenvolvidos possui características próprias e existem situações em que uns podem ter melhor desempenho do que outros. Os algoritmos genéticos funcionam muito bem com espaços de procura em constante mudança e onde pouco é conhecido. Só necessita de saber o que precisa de fazer bem (através da função fitness) para o fazer, chegando a uma solução ótima. Deste modo, trabalham bem quando têm um número elevado de soluções candidatas e até mesmo quando estas soluções são díspares entre si. No entanto, não são tão eficientes para situações em que os espaços de procura são muito simples, pois poderão demorar muito tempo até encontrar uma solução adequada.

Dentro da aprendizagem por reforço, o algoritmo Q-Learning não assume uma performance muito eficiente quando os dados têm muito "barulho", isto é, quando há dados que não têm um significado útil para o problema ou não são facilmente compreendidos pelo sistema. Numa situação assim, o algoritmo tem dificuldade em convergir para um valor ótimo, encontrando, então, apenas um valor razoável. Por sua vez, o algoritmo SARSA encontra maiores adversidades quando trabalha com pouco conhecimento, uma vez que, ao contrário do Q-Learning que lida com informação futura, requer informação imediata para os cálculos da política ótima.

Por fim, as árvores de decisão não têm uma performance boa quando os dados possuem muitas perturbações, uma vez que com uma pequena mudança nos dados pode-se obter uma árvore completamente nova. Além disso, se não se tomar as precauções necessárias, corre-se sempre o risco de chegarmos a situação de overfitting, isto é, a árvore aperfeiçoou-se demasiado bem aos dados de treino, falhando na previsão de dados que não sejam semelhantes aos aprendidos.

Em suma, não se pode concluir que um dos métodos de aprendizagem é melhor do que todos os outros, dado que cada um tem os seus pontos fracos, pois funcionam melhor em determinados casos e noutros apresentam uma prestação mais fraca, podendo ser substituídos por métodos mais adequados.

7 References

"Genetic Algorithms in Plain English" Retirado de:http://www.ai-junkie.com/ga/intro/gat1.html
"How the Genetic Algorithm Works". Retirado de: https://www.mathworks.com/help/gads/how-the-genetic-algorithm-works.html
"Genetic Algorithm Terminology". Retirado de: https://www.mathworks.com/help/gads/some-genetic-algorithm-terminology.html
"Research Highlights". Retirado de: https://www.nrl.navy.mil/itd/aic/highlights
"Introduction to Genetic Algorithm their application in data science" Retirado de: https://www.analyticsvidhya.com/blog/2017/07/introduction-to-genetic-algorithm/
"The Evolution of Code" Retirado de: http://natureofcode.com/book/chapter-9-the-evolution-of-code/
"An Introduction to Genetic Algorithms" Retirado de: http://www.boente.eti.br/fuzzy/ebook-fuzzy-mitchell.pdf
"Index of Most Important Applications of the Genetic Algorithms" Retirado de: http://neo.lcc.uma.es/TutorialEA/semEC/cap03/cap_3.html
"GPdotNET - artificial intelligence tool" Retirado de:https://gpdotnet.codeplex.com/
"Reinforcement Learning" Retirado de: http://reinforcementlearning.ai-depot.com/
"Deep Reinforcement Learning" Retirado de: https://deepmind.com/blog/deep-reinforcement-learning/
"Taming the Noise in Reinforcement Learning via Soft Updates" Retirado de: https://arxiv.org/abs/1512.08562
“Reinforcement Learning”. Retirado de http://www.cse.unsw.edu.au/ cs9417ml/RL1/
Von Zuben, Fernando J. and Attux, Romis R.F., “Árvores de Decisão”, PowerPoint presentation, DCA/FEEC/Unicamp, disponível em: goo.gl/fH478F
“Aprendizagem por Reforço”. Retirado de goo.gl/orrMGK
“Reinforcement learning”. Retirado de goo.gl/9uT2Ci
Quinlan, J. R., “Induction of Decision Trees”, Machine Learning 1: 81-106, Kluwer Academic Publishers, 1986