Sistemas de Aprendizagem

Tiago André Alves Bouças (A75054)[1] e Tiago Nuno Fernandes Duarte (A71004)[2]

Grupo Nº8
Universidade do Minho, Braga, Portugal
[1] a75054@alunos.uminho.pt
[2] a71004@alunos.uminho.pt

Resumo. Este artigo tem como principal objetivo dar a conhecer 3 tipos de aprendizagem máquina ligados à inteligência artificial, que neste caso são a aprendizagem por reforço, as redes neuronais artificiais e as árvores de decisão. Neste sentido, o artigo está dividido nos 3 tipos de aprendizagem onde cada um tem uma descrição geral do tema, os algoritmos de aprendizagem mais utilizados, algumas ferramentas de desenvolvimento e aplicações na vida real. Para além de descrever os tipos de aprendizagem este artigo procura efetuar comparações, de modo a que se perceba em que situações podemos tirar partido de um em detrimento dos outros.

Palavras Chave: Aprendizagem máquina, Inteligência artificial, aprendizagem por reforço, redes neuronais artificiais, árvores de decisão.

1. Introdução

O aspeto interativo da aprendizagem máquina é importante porque, à medida que os modelos são expostos a novos dados, eles são capazes de se adaptar de forma independente. É uma ciência que não é nova, mas que está a ganhar um novo impulso. Para isso contribuiu o crescente volume e variedade de dados, o processamento computacional mais barato e mais poderoso e o armazenamento de dados de forma fácil que ajudou a despertar o interesse pela aprendizagem máquina.

Hoje em dia a aprendizagem máquina está por todo o lado, seja na deteção de fraudes, nas pesquisas web, no reconhecimento de padrões e imagem, a filtrar spams no email, até mesmo nas redes sociais, etc.

Existem vários tipos de aprendizagem, a supervisionada, a não supervisionada e a aprendizagem por reforço. A aprendizagem supervisionada lidera com cerca de 70% da aprendizagem máquina existente hoje em dia, enquanto que a aprendizagem não supervisionada varia de 10 a 20%, sendo o resto para a aprendizagem por reforço.

Na aprendizagem supervisionada existe a necessidade de um especialista de domínio para rotular tipos de exemplos, que forneça casos para o modelo se basear. Quando se fala na aprendizagem não supervisionada, o aprendizado identifica padrões em dados e agrupa-os de acordo com a semelhança dos dados, sem a necessidade de

um especialista. Já a aprendizagem por reforço encontra-se ao meio, pois não precisa de um especialista mas utiliza entradas do ambiente como reforço.

2. Aprendizagem por Reforço

2.1 Descrição

É uma área da aprendizagem máquina que pertence ao ramo da inteligência artificial, inspirada na psicologia ligada a comportamentos, que se preocupa especialmente como deve agir um agente num ambiente, de modo a maximizar a recompensa cumulativa. Esse ambiente pode ser muitas vezes dinâmico.

Difere da aprendizagem supervisionada na medida em que a aprendizagem supervisionada aprende com um conjunto de treinos fornecido por um supervisor externo. Os exemplos fornecidos contém tanto a entrada como a saída.

Este tipo de aprendizagem também difere da aprendizagem não-supervisionada onde não existe qualquer tipo de "professor" ou supervisor, e por conseguinte, a rede tem de descobrir sozinha qualquer tipo de padrão, relação ou categoria nos dados que lhe vão sendo apresentados e codificá-los nas saídas. O fornecimento dessa informação como ocorre nos supervisionados pode custar muito tempo, dinheiro e esforço.

Um dos desafios que este tipo de aprendizagem tem de ultrapassar é o *trade-off* entre *exploration* e *exploitation*, ou seja, entre o território não explorado e o conhecimento já adquirido, respetivamente. A *exploration* está ligada à aquisição de novos conhecimentos enquanto que o *exploitation* consiste na otimização de conhecimentos já adquiridos.

A aprendizagem por reforço permite que máquinas e agentes de Software determinem automaticamente o comportamento ideal pelo método de tentativa e erro, de modo a maximizar as recompensas. É recomendado um sistema de feedback de recompensa simples de modo a que o agente aprenda o seu comportamento (conhecido como o sinal de reforço).

Neste tipo de aprendizagem é possível aprender tudo de uma vez ou ir aprendendo ao longo do tempo, ou seja, melhorar o seu desempenho obtendo recompensas dependendo também do problema a resolver. É muito utilizado na robótica, jogos e navegação.

Esta tarefa de aprendizagem complica quando passamos de um sistema com apenas um agente para um sistema multiagente. Se o problema possui múltiplos agentes a aprender e a interagir no mesmo ambiente a complexidade aumenta consideravelmente, pois nestas condições o comportamento do agente tem de ser influenciado também pelas ações de outros agentes, além das caraterísticas do ambiente (estático ou dinâmico, discreto ou continuo, etc.). A aprendizagem com um único agente tem as suas raízes na aprendizagem animal. É muito comum punir ou recompensar um animal mediante uma ação realizada, a qual ele vai assimilar e reagir em ações futuras. Para garantir que a solução é encontrada, a aprendizagem por reforço recompensa os comportamentos desejados e pune todos os indesejados.

Quando falamos em aprendizagem por reforço multiagente, o agente precisa de aprender a comportar-se na presença de outros agentes que também são capazes de aprender. O aumento do número de agentes no mesmo ambiente provoca um aumento considerável no número de variáveis a observar. Esse aumento pode aumentar consideravelmente a complexidade do problema, mas também traz imensos benefícios.
Entre esses benefícios está o aumento na rapidez do processo de aprendizagem, a partilha de experiência e robustez, já que caso falhem alguns agentes outros podem assumir as suas tarefas.

Este modelo apresenta algumas limitações, como em certos casos é muito caro armazenar valores de cada estado, devido à existência de problemas de elevada complexidade.

2.2 Aprendizagem

Em Aprendizagem por Reforço, para se definir uma solução, são utilizados os processos de decisão de Markov. Estes definem a seguinte estrutura:

- S, conjunto de estados possíveis em que o ambiente se encontra;
- A, conjunto de ações que o agente pode tomar;
- R, recompensa;
- π, política.

Para por em contexto, consideremos um agente que interage com o ambiente em vários momentos, t, até atingir o objetivo que lhe foi proposto. Em cada um desses momentos o ambiente encontra-se num estado St ∈ S e, com base nisso, o agente toma uma ação At ∈ A(St), como consequência da ação tomada o agente recebe uma recompensa, Rt, que pode ser positiva ou negativa. A cada momento, a probabilidade de para um dado estado tomar uma determinada ação é dada pela política, π, que vai sendo alterada com a experiência. O objetivo do agente passa por maximizar o total de recompensas recebidas.

Quase todos os algoritmos de aprendizagem por reforço requerem o cálculo de estimativas sobre o quão "bom" é para o agente estar num determinado estado. Esta estimativa é calculada com recurso à política e pode ser obtida através da seguinte fórmula:

$$v_\pi(s) = \mathrm{E}_\pi[\,\mathrm{G}_t | S_t = s] = \mathrm{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \quad (1)$$

onde γ representa uma taxa de desconto, entre 0 e 1, e R as recompensas.

De forma similar, também se pode estimar o valor de tomar uma determinada ação, segundo a política π, partindo de um dado estado inicial:

$$q_\pi(s,a) = \mathrm{E}_\pi[\,\mathrm{G}_t | S_t = s, A_t = a] = \mathrm{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right] \quad (2)$$

As equações de Bellman permitem-nos, ainda, obter o valor, para cada um das equações anteriores, utilizando uma política ótima:

$$v_*(s) = max_{a \in A(s)} \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \quad (3)$$

$$q_*(s,a) = \sum_{s',r} \mathrm{p}(\mathrm{s}', \mathrm{r}|\mathrm{s}, \mathrm{a})[\mathrm{r} + \gamma \mathrm{max}_{\mathrm{a}'}\, \mathrm{q}_*(\mathrm{s}', \mathrm{a}')] \quad (4)$$

Esta é uma estrutura abstrata e bastante flexível e que, como tal, pode ser utilizada em várias situações e de formas diferentes.

Para por isto em prática, podem ser utilizados três métodos: programação dinâmica, métodos de Monte Carlo e os métodos de diferença temporal.

2.3 Programação Dinâmica

O método de programação dinâmica tem uma aplicação um pouco limitada, devido à assunção de que estão perante um modelo perfeito do ambiente e devido ao seu elevado consumo computacional. Em programação dinâmica, é realizada uma procura da melhor política através de cálculos do q_π e do v_π. Sabendo que se para um dado estado e uma dada ação, se q_π for maior do que v_π, então é melhor tomar essa dada ação e só depois seguir a política. Daqui obtemos que se

$$q_\pi(s, \pi') \geq v_\pi(s), para\ todo\ s \in S \quad (5)$$

então, a política π' produz um melhor desempenho que a política π. Contudo, a avaliação de uma política é um processo custoso, que requer múltiplas travessias do conjunto de estados, e este algoritmo de programação envolve esse cálculo a cada iteração. Alternativamente, é possível truncar a avaliação da política, sem perder a garantia de convergência.

$$V(s) = max_a \sum_{s',r} p(s', r!\, s, a)[r + \gamma V(s')] \quad (6)$$

Consideremos uma variável delta, que será inicializada a 0 e um vetor, de tamanho igual ao número de estados, inicializado arbitrariamente. Para cada estado, delta tomará o valor do maior número entre o seu valor atual e o módulo da diferença entre o valor do vetor, corresponde ao estado em questão, e o valor obtido pela fórmula 6, que tem a conta a avaliação da política truncada e o melhoramento da política. Este processo é repetido até delta tomar um valor menor que o critério de paragem, um

número pequeno e positivo definido previamente. A política determinística é obtida através de:

$$\pi(s) = argmax_a \sum_{s',r} p(s', r|sla)[r + \gamma V(s')] \quad (7)$$

2.4 Métodos de Monte Carlo

Ao contrário da programação dinâmica, os métodos de Monte Carlo não necessitam de informação completa sobre o ambiente, estes utilizam apenas a experiência. Esta metodologia tem por base a média dos retornos obtidos em sequência e está definida para tarefas episódicas, em vez de tarefas por passos, ou momentos. Dado que os conjuntos de estados e de ações sejam finitos, para uma dada política, o objetivo passa por calcular por o valor de retorno para cada estado e calcular a sua média. Após visitados, todos os estados a política é atualizada e um novo episódio é iniciado. Para um estado, s, v_π (s) converge, quando o número de visitas a s tende para infinito. Apesar de a capacidade de aprendizagem sem conhecimento sobre o ambiente ser vantajoso, este método também tem as suas desvantagens: apenas pode ser utilizado em problemas que possam ser decompostos em episódios e que possuam conjuntos finitos de estados e ações e o caráter episódico pode levar a que muito tempo seja despendido a avaliar política que estejam muito aquém da ótima.

2.5 Diferença Temporal

O algoritmo da diferença temporal resulta de uma combinação de ideias do algoritmo de Monte Carlo e de programação dinâmica. Como no algoritmo de Monte Carlo, os métodos de diferença temporal podem aprender diretamente da experiência sem um modelo de programação dinâmica, no sentido em que não precisam do ambiente. Como programação dinâmica os métodos de diferença temporal atualizam as estimativas com base em outras estimativas aprendidas. As principais diferenças entre os métodos é na previsão.

Predição

Ambos os métodos, diferença temporal e Monte Carlo utilizam a experiência para resolver o problema de previsão.

Nos processos de decisão de Markov, a diferença temporal visa estimar a função do valor do estado seguindo uma politica pi.

A regra de atualização baseia-se na seguinte formula.

$$\boldsymbol{V(s) = V(s) + \alpha(G(t) - V(s)), onde\ G(t) = r + \gamma V(s')} \quad (8)$$

S representa um estado antigo e s' o novo estado, α é a taxa de aprendizagem e r a recompensa na transição, e finalmente λ a taxa de desconto.

Como este método baseia a sua atualização numa estimativa existente diz-se que é um método de *bootstrapping*.

Após a junção das formulas de métodos anteriores obteve-se a formula da diferença temporal.

$$v_\pi(t) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \quad (9)$$

O resultado será apenas uma estimativa por duas razões, pelo facto de usar o valor da estimativa de V em vez do valor verdadeiro de V_π e por resultar de valores esperados como acontece no caso da equação (9). De notar que o erro em cada momento é o erro feito na estimativa para um dado momento.

O algoritmo TD(0) (simples método da diferença temporal) executa da seguinte forma:

- Inicializa V(s) arbitrariamente, para um valor de pi;
- Repete até s ser terminal:
    - Calcula valores intermédios, por exemplo o valor de R
    - Calcula a equação 9
    - S passa a S'

A diferença temporal é dada por V(S') – V(S).

Vantagens em relação aos modelos anteriores

Uma vantagem clara deste modelo em relação à programação dinâmica é o facto de não requerer um modelo ambiente. Em relação ao modelo de Monte Carlo também existe uma vantagem clara, pois no modelo MC é preciso esperar pelo final para obter valores, enquanto que neste modelo temos de esperar apenas uma iteração.

O método de Monte Carlo deve ignorar ou descontar ações realizadas em experiências, o que dificulta a aprendizagem. Os métodos de diferença temporal são muito menos suscetíveis a esses problemas porque aprendem em cada transição.

Otimalidade de TD(0)

Deve-se treinar com uma quantidade infinita de dados até existir convergência. A atualização deve ser feita de acordo com o algoritmo TD(0), mas apenas atualizar após uma passagem completa pelos dados, um exemplo é treinar repetidamente em episódios de 10 até à convergência.

Para qualquer tarefa finita de previsão de Markov, em atualizações de lote, o algoritmo TD(0) converge para um valor de α pequeno.

Sarsa

O algoritmo Sarsa é um algoritmo *On-Policy* (de acordo com uma politica). A principal diferença entre ele e o Q-Learning, é que a recompensa máxima para o próximo estado não é necessariamente usada para atualizar os Q-values. Em vez disso, uma nova ação e, consequentemente, recompensa, é selecionada usando a mesma política que determinou a ação original. O nome do algoritmo (Sarsa) vem do fato de que as atualizações são feitas usando um quíntuplo Q (s, a, r, s', a'). Onde: s é o estado original, a é a ação, r a recompensa observada no seguinte estado e s', a' são o novo par ação de estado.

$$\mathrm{Q(S,A)} \leftarrow \mathrm{Q(S,A)} + \alpha\,[\mathrm{R} + \gamma \mathrm{Q(S',A')} - \mathrm{Q(S,A)}] \qquad (10)$$

Se S_{t+1} é terminal, então Q(S_{t+1}, a_{t+1})=0.

As propriedades de convergência dependem da natureza da politica em Q. Pode-se usar politicas gananciosas ou *soft*.

Q-Learning

O Q-Learning é um algoritmo *Off-Policy* (fora de politica) para aprendizagem de Diferença Temporal. Assim sendo, o algoritmo pode ser usado para encontrar uma politica de seleção de ação ideal para cada processo de decisão de Markov. Mediante treino suficiente sob qualquer politica, o algoritmo converge com a probabilidade 1 para uma aproximação próxima da função de valor de ação. O Q-Learning aprende a política ótima mesmo quando as ações são selecionadas de acordo com uma política mais exploratória ou mesmo aleatória. O algoritmo é muito parecido ao TD(0), embora com uma formula diferente, que neste caso é a seguinte.

$$\mathrm{Q}(S_t, \mathrm{A}_t) \leftarrow \mathrm{Q}(S_t, \mathrm{A}_t) + \alpha\,[\,\mathrm{R}_{t+1} + \gamma \max_a \mathrm{Q}(\mathrm{S}_{t+1}, \mathrm{a}) - \mathrm{Q}(S_t, \mathrm{A}_t)\,] \qquad (11)$$

2.6 Ferramentas de Desenvolvimento

Entre as principais ferramentas onde é possível desenvolver software baseado na aprendizagem por reforço estão algumas linguagens de programação bem conhecidas, como *python* e c++, devido à existência de *packages* com diversas funcionalidades.

- ***Python*** – Existem vários *packages* e *toolkits* que permitem trabalhar facilmente com aprendizagem por reforço, como por exemplo o Reinforcement-Learning-Toolkit.

- **C++** - Nesta linguagem de programação existe uma biblioteca (Dlib) que disponibiliza algoritmos de aprendizagem máquina, com capacidade de criar software de elevada complexidade. Tem a grande vantagem de ser um software de código aberto.

2.7 Soluções Reais

A aderência a este tipo de aprendizagem tem aumentado nos últimos anos com o avanço da tecnologia. Encontram-se alguns casos de empresas bem conhecidas, como a google e a Microsoft, por exemplo.

A google desenvolveu um sistema de inteligência artificial, designado por AlphaGo, criado para lidar com a infinidade de jogadas possíveis do Go. Enquanto que por exemplo nos jogos de xadrez são determinadas todas as combinações possíveis, a equipa da DeepMind carregou no sistema cerca de 30 milhões de jogadas de jogadores profissionais. O principal destaque neste sistema é a capacidade de criar as suas próprias estratégias recorrendo para isso a um processo de tentativa-erro. Este sistema tornou-se assim no primeiro sistema de inteligência artificial a bater um humano. [9]

Também a Microsoft utilizou jogos para testar os seus sistemas de inteligência artificial, neste caso para obter a pontuação máxima do jogo PacMan, muito popular nos anos 80. [10]

3. Redes Neuronais Artificiais

3.1 Descrição

O cérebro humano é a grande inspiração por detrás das redes neuronais artificias, visto que é o que, para muitos, define um ser inteligente. Este interage com o meio ambiente processando informação e delegando a resposta apropriada, informação que é recolhida a partir dos vários sensores à disposição do homem, os cinco sentidos.

O cérebro é um órgão altamente complexo, capaz de realizar várias tarefas ao mesmo tempo, ou seja paralelo, e não linear, isto é, o pensamento humano é capaz de expandir para várias direções, não segue um caminho definido. Ainda hoje, é alvo de numerosos estudos e ainda não é compreendido na sua totalidade, no entanto, nem tudo é desconhecido. Os neurónios são os constituintes base do cérebro, estes são células que recebem, processam e transmitem informação através de sinais elétricos e químicos, e os milhares de milhões de neurónios que constituem o cérebro estão ligados entre si através de sinapses. É também sabido que o cérebro pode ser moldado, o cérebro de um recém-nascido não está totalmente formado, apesar de já apresentar capacidade de aprendizagem, é com a experiência que o cérebro vai alterando a sua estrutura, vai adquirindo conhecimento e evoluindo. É isto que as redes neuronais artificiais procuram emular.

Assim, as redes neuronais artificiais baseiam-se em duas principais características de modo a tornar possível a resolução de problemas complexos: uma topologia distribuída, que privilegia o paralelismo e a capacidade de aprendizagem e, como tal, generalização. Contudo, na prática, não conseguem dar resposta a qualquer problema sozinhos. Por norma, são integradas em sistemas onde um problema complexo é dividido em pequenas tarefas e onde cada rede realiza uma dessas tarefas.

As redes neuronais apresentam as seguintes características, para além das duas já referidas:

- Não linearidade, a rede deve poder solucionar problemas lineares e não lineares, sendo que estes últimos representam a maioria;
- Adaptabilidade, a rede é capaz de se adaptar ao ambiente que a rodeia;
- Tolerância a falhas, a rede deve ser capaz de agir eficientemente, ou, por outros termos, degradar suavemente, em condições adversas, como a existência de informação incompleta ou eventuais falhas de nodos ou ligações.;
- Flexibilidade, as redes neuronais podem ser aplicadas a um vasto leque de problemas .

3.2 Arquitetura

Estruturalmente, as redes neuronais podem ser vistas como grafos direcionais, onde os nodos representam os neurónios e as suas ligações as sinapses. Cada sinapse tem ainda um peso associado que varia ao longo do tempo e que determina a importância do sinal que entra no neurónio. Cada neurónio possui um integrador, que reduz o número de argumentos de entrada para um, e uma função de ativação, que transforma o valor de entrada no valor de transferência ou saída.

A forma como os nodos estão organizados na rede é denominada de topologia, que normalmente podem ser classificadas em um destes três tipos:

- Rede *Feed Forward* de uma camada. Caracterizada por não conter ciclos, ligações unidirecionais, e por ser composta por apenas uma camada de entrada e uma de saída.
- Rede *Feed Forward* Multi-Camada. Semelhante à anterior, no sentido de ser acíclica. Porém, contém camadas intermédias, ou seja que estão entre as de entrada e de saída. Isto permite aumentar a capacidade da rede em modelar funções com nível de complexidade mais elevado, no entanto, o tempo de aprendizagem também cresce exponencialmente.
- Rede Recorrente. Utilizada, sobretudo, em sistemas dinâmicas, estas redes são diferem das outras devido à existência de conexões cíclicas. Isto leva também a que certas conexões passem a depender de uma dimensão temporal, a função que as caracteriza será do tipo recursivo e obedecerá a uma determina condição de paragem imposta.

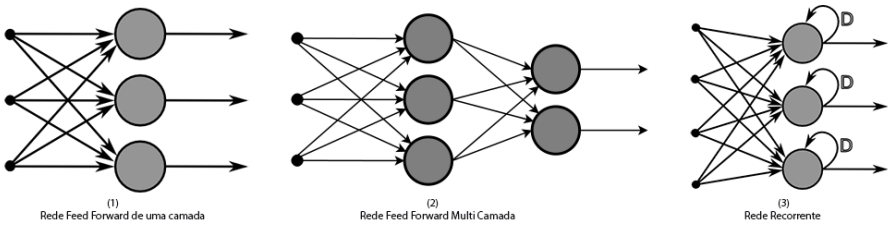

Figura 1 - Ilustração das diferentes arquiteturas de rede

3.3 Aprendizagem

As redes neuronais artificiais utilizam informação proveniente do ambiente onde se encontram para aprender. A forma como estas se relacionam com o ambiente é definida através de paradigmas.

O paradigma mais comum é o de aprendizagem supervisionada. Este paradigma requere a participação de um "professor", que confere respostas corretas à rede, a partir das quais esta vai aprender. Cada um destes casos de treino deverá ter parâmetros de entrada e os parâmetros de saída correspondentes. Durante a aprendizagem, é feita uma comparação entre o valor esperado e o valor obtido, a partir da qual é calculado um erro, que é utilizado para ajustar os pesos das ligações. Este processo é repetido iterativamente até se obter um valor para o erro considerado aceitável.

Outro paradigma utilizado é o de reforço. Neste não se pode dizer que haja o envolvimento de um "professor", visto que as respostas não são fornecidas à rede. Em vez disso, são dadas indicações sobre a correção da resposta da rede. Quando a rede obtém uma resposta correta, é lhe dado um estímulo positivo como reforço.

Por fim, há ainda o paradigma de aprendizagem não supervisionada. Este já não necessita de um "professor". Neste caso, não é fornecida nenhuma informação externa à rede quanto à correção das respostas. A aprendizagem é realizada através da adaptação às regularidades estatísticas dos dados de entrada, permitindo criar novas formas de representação interna desses, agrupando-os por padrões encontrados ou determinadas características que lhes são inerentes.

Para além da forma como se relaciona com o meio, a aprendizagem é, ainda, realizada a partir de um conjunto de regras bem definidas, que podem ser dividas em cinco tipos:

- Hebbian. Trata-se da expansão em duas partes a partir do postulado mais antigo de aprendizagem, proposto por Donald O. Hebb. A primeira parte diz que se dois nodos, um em cada lado de uma dada conexão, forem ativados simultaneamente, a força dessa conexão é aumentada. Por outro lado, se para um dada conexão, os dois nodos que a compõem forem ativados ao mesmo tempo, a força dessa ligação é diminuída, podendo, inclusive, levar à sua eliminação. Este algoritmo é sobretudo utilizada em redes que sigam paradigmas de aprendizagem não supervisionadas.

- Estocástico. Neste caso, os pesos são ajustados tendo em conta uma distribuição probabilística.
- Competitivo. Neste tipo de aprendizagem, apenas um nodo está ativo, a um dado momento, em cada, levando a que os vários nodos de cada camada compitam entre si para se tornarem ativos. Inicialmente, as ligações têm todas pesos pequenos, mas diferentes. Sempre que um padrão é fornecido, os pesos das ligações do nodo que responder melhor serão reforçados, sendo que, me certas situações, este reforço também pode ser aplicado aos nodos vizinhos.
- Baseado na memória. Neste método, são armazenadas, em conjuntos de pares entrada - saída, todas as experiências passadas. Quando emergir uma nova experiência, são procuradas experiências passadas na região de vizinhança da nova, através de um critério de definição de vizinhança local.
- Gradiente Descendente. Este tipo de aprendizagem que está normalmente associado ao algoritmo de *Back - Propagation*, baseia-se na diminuição do erro entre o valor pretendido e o valor obtido pela rede. Tem como objetivo minimizar uma função de custo, definida em termos do sinal de erro. Esta regra de aprendizagem baseia-se na seguinte equação:

$$\Delta\omega = \eta\Delta\xi \tag{12}$$

onde η é a taxa de aprendizagem, representada por uma constante positiva e $\Delta\xi$ o gradiente da função de custo.

3.4 Algoritmo de *Back-Propagation*

Um dos algoritmos mais populares é o de *Back-Propagation*, que é do tipo gradiente descendente e pertence ao paradigma de aprendizagem supervisionada e que se caracteriza pela sua eficiência, quando utilizado como método de treino em redes *feed forward* multi-camada. Antes de se iniciar o treino, são atribuídos pesos, valores pequenos e aleatórios, às ligações. Iniciado o treino, os valores fornecidos aos nodos de entrada vão-se propagando para a frente até chegarem aos nodos de saída, onde é feita a comparação entre o valor obtido e o esperado. Calculado, o erro este é propagado para trás, como o nome do algoritmo indica, desde os nodos de saída até aos de entrada, levando ao ajustamento dos pesos segundo a regra de *Widrow-Hoff*:
onde t é a iteração e n a taxa de aprendizagem. Este processo é repetido para cada iteração. O treino termina quando forem cumpridas os critérios de paragem definidos.

3.5 Ferramentas de desenvolvimento

Existem algumas ferramentas disponíveis que permitem a implementação de redes neuronais artificiais.

A linguagem de programação R tem à sua disposição o pacote neuralnet, que permite a implementação de redes neuronais supervisionadas que utilizem o algoritmo de *Back - Propagation.*

De forma semelhante, existem também pacotes ou bibliotecas para Matlab (Neural NetWork Toolbox), Haskell (HNN) e Python (TensorFlow).

Existem ainda aplicações que oferecem uma interface gráfica e mais intuitiva que permitem, também elas, a implementação de redes neuronais como o JustNN, o Neuroph ou o Encog.

3.6 Soluções existentes no mercado

Este é um mercado em constante crescimento e que já apresenta, hoje em dia, um variado número de opções. De seguida, alguns exemplos, em diferentes mercados.

O GMDH Shell é um programa que efetua previsões e ajuda a otimizar os inventários das empresas, através do alinhamento do inventário com a procura do consumidor, ajudando a precaver a falta ou o excesso de produto em *stock*. Estas previsões são feitas recorrendo a redes neuronais artificiais. [7]

A Prisma é uma aplicação para dispositivos móveis que aplica filtros, inspirados, sobretudo, por famosos pintores, a imagens. A aplicação utiliza três redes neuronais para analisar a imagem original (identificar pessoas e objetos), extrair o estilo artístico da pintura famosa e, por fim, aplicá-la. [8]

As redes neuronais artificiais também desempenham um papel importante no emergente mercado dos carros autónomos. Estas são um dos grandes intervenientes que têm permitido a estes carros emular o comportamento dos condutores, ajudando na delimitação das vias de trânsito, na identificação dos demais veículos e consequentes decisões a tomar. [9]

4. Árvores de Decisão

4.1 Descrição

Uma árvore de decisão é um exemplo muito simples com grande sucesso de um algoritmo de aprendizagem perfeitamente enquadrado na área de aprendizagem indutiva. São ferramentas de suporte à decisão, que utilizam grafos semelhantes a árvores ou um modelo de decisões, onde inclui as suas consequências (por exemplo, resultados de eventos ocasionais ou outras utilidades). São ainda comumente usadas em pesquisas, mais especificamente na análise de decisões e na identificação de uma estratégia com maior probabilidade de atingir um objetivo. Tornaram-se numa ferramenta popular na aprendizagem por máquina. São semelhantes a fluxogramas em que cada nó representa um "teste" aos atributos. Ao longo dos ramos tomam-se várias

decisões, em que cada ramo está etiquetado com valores possíveis do atributo, obtendo o resultado final quando é atingida a folha da árvore, após o cálculo de todos os atributos.

Designa-se por classe o conceito alvo, como por exemplo, numa árvore de decisão em que se pretende saber se hoje é um bom dia para jogar ténis perante as condições atmosféricas, o conceito alvo varia entre sim e não, ou seja, a classe acaba por ser a resposta que pretendemos.

Existem dois tipos de árvores, as árvores de classificação e as de regressão.

Nas árvores de classificação a variável alvo pode ter um conjunto discreto de valores, onde os ramos normalmente representam conjunções de características. Os caminhos da raiz até às folhas representam regras de classificação.

As árvores de decisão onde a variável alvo pode ter valores contíguos são chamadas de árvores de regressão.

Normalmente as árvores de decisão podem ser linearizadas em regras de decisão onde o resultado é o conteúdo da folha e o caminho percorrido formam uma conjugação de clausulas *If*. Por exemplo, Se condição 1 e condição 2, então resultado.

Figura 2 - Exemplo de uma árvore de decisão

Fonte: http://web.tecnico.ulisboa.pt/ana.freitas/bioinformatics.ath.cx/bioinformatics.ath.cx/uploads/RTEmagicC_arv_dec4_01.gif .gif

Neste exemplo, a cor vermelha representa o atributo (Aspeto, Humidade e Vento), enquanto que a cor azul representa os valores possíveis dos atributos, por exemplo no caso do Aspeto que se divide em Sol, Nuvens e Chuva. A verde está representada a classe, que varia apenas entre Sim e Não. Como a variável alvo tem valores discretos estamos perante uma árvore de decisão.

Supondo que hoje está um lindo dia de sol com elevada humidade, chega-se à conclusão que não é um bom dia para jogar ténis. Isto pode ser representado através da conjugação *If*, Se Aspeto = Sol e Humidade = Elevada então Não é um bom dia para jogar ténis.

4.2 Tarefa de Indução

O processo de indução de árvores de decisão exige grande poder computacional, mas depois de construída, o seu uso é imediato e muito rápido, tornando-se uma das vantagens da sua utilização.

A indução de uma árvore de decisão é a construção de árvores de decisão a partir de treinos com informação da classe. Consiste no desenvolvimento de uma regra de classificação que pode determinar a classe de qualquer objeto a partir dos valores dos seus atributos. Se dois objetos possuem valores semelhantes para cada atributo e pertencem a classes diferentes é claramente impossível diferenciar esses 2 objetos apenas com a referência somente aos atributos dados. Nesse caso os atributos são considerados inatos para a tarefa de indução.

A aprendizagem da árvore pode ser feita dividindo o conjunto de fontes em subconjuntos, com base em testes de valor feitos aos atributos. Repete-se o processo para todos os subconjuntos de forma recursiva. A recursão termina quando o subconjunto em um nó tem o mesmo valor da variável alvo ou quando a divisão não aumenta o valor das previsões. Este processo de indução que se realiza de cima para baixo em árvores de decisão (designado por TDIDT) é um exemplo de um algoritmo ganancioso e, de longe, é a estratégia mais comum na aprendizagem das árvores de decisão.

Este algoritmo do tipo *top-down* divide o problema em subproblemas, os quais são resolvidos posteriormente.

Exemplos de Treino

Dia	Aspecto	Temp.	Humidade	Vento	Jogar Ténis
D1	Sol	Quente	Elevada	Fraco	Não
D2	Sol	Quente	Elevada	Forte	Não
D3	Nuvens	Quente	Elevada	Fraco	Sim
D4	Chuva	Ameno	Elevada	Fraco	Sim
D5	Chuva	Fresco	Normal	Fraco	Sim
D6	Chuva	Fresco	Normal	Forte	Não
D7	Nuvens	Fresco	Normal	Fraco	Sim
D8	Sol	Ameno	Elevada	Fraco	Não
D9	Sol	Fresco	Normal	Fraco	Sim
D10	Chuva	Ameno	Normal	Forte	Sim
D11	Sol	Ameno	Normal	Forte	Sim
D12	Nuvens	Ameno	Elevada	Forte	Sim
D13	Nuvens	Quente	Normal	Fraco	Sim
D14	Chuva	Ameno	Elevada	Forte	Não

Figura 3 - Exemplo de casos de treino para a Fig.1

Fonte: http://web.tecnico.ulisboa.pt/ana.freitas/bioinformatics.ath.cx/bioinformatics.ath.cx/indexf23d.html?id

As metodologias de aprendizagem da família TDIDT, são menos complexas do que aquelas empregadas em sistemas que permitem expressar os resultados da sua aprendizagem numa linguagem mais poderosa. No entanto, continua a ser possível gerar conhecimento na forma de árvores de decisão capazes de resolver problemas complicados. Mais concretamente, a estratégia utilizada é a aprendizagem não-incremental a partir de exemplos.[12]

Dado um conjunto C com as classes C1, C2, …, Ck, o algoritmo TDIDT baseia-se nos seguintes pontos.

1. Se todos os elementos pertencem à mesma classe Cj então cria-se uma folha que os englobe a todos identificada pela classe Cj.
2. Se não se verificar o ponto 1, então seleciona-se o atributo com melhor avaliação de acordo com o critério selecionado. Depois de selecionado o melhor atributo, o conjunto T é dividido em subconjuntos T1, T2, …, Tk e posteriormente construídas as subárvores que definem esses conjuntos.
3. O algoritmo é efetuado recursivamente até atingir o caso de paragem.

Basicamente, o algoritmo TDIDT é um algoritmo recursivo de busca gulosa que procura a melhor divisão do conjunto de dados C em subconjuntos.

Os sistemas são apresentados como um conjunto de casos relevantes para uma tarefa de classificação, não necessariamente apresentados pela ordem em que os exemplos são fornecidos. Os sistemas descritos aqui buscam padrões nos exemplos fornecidos e, portanto, devem poder examiná-los e reexaminá-los durante a aprendizagem.

A escolha do atributo na maioria dos algoritmos de construção das árvores de decisão são sem retrocesso (*backtracking*), ou seja, gulosos (*greedy*). Sendo assim, uma vez que um atributo é selecionado são eliminadas todas as escolhas alternativas. Diferentes algoritmos tem diferentes formas de escolher os atributos. Apresentam-se de seguida algumas dessas formas.

- **Aleatória** - Seleciona um atributo aleatoriamente.
- **Maior ou menor quantidade de valores** - Seleciona o atributo que tem mais ou menos quantidade de valores possíveis.
- **Ganho máximo** - Seleciona o atributo que possui o maior ganho de informação esperado.
- **Entre outros**.

Nas subsecções seguintes apresentam-se os principais algoritmos para indução de árvores baseados em TDIDT(*Top-Down Induction of Decision Tree*), nos quais se apresenta: o ID3 (QUINLAN, 1986), o C4.5 (QUINLAN, 1993) e o CART (BREIMAN et al., 1984).

Existem outros tipos de algoritmos que fogem do algoritmo básico TDIDT, como por exemplo o ADTree (FREUND & MASON, 1999), o NBTree (KOHAVI, 1996) e o LMT (LANDWEHR et al., 2005) que não serão abordados nesta pesquisa.

4.3 Algoritmos de Aprendizagem

ID3

Este algoritmo proposto por *Ross Quinlan*, permite criar árvores de decisão a partir de um conjunto de dados, normalmente utilizado em aprendizagem máquina e processamento de linguagem natural. O algoritmo inicia com o conjunto original C no nó raiz. O conjunto C tem de ser dividido em subconjuntos em que cada subconjunto tem a informação de um atributo, de acordo com o caso de estudo. Em cada iteração são iterados todos os atributos não utilizados do conjunto C e calculada a entropia para cada um, H(C). Em seguida, seleciona o que tem menor entropia, neste caso o maior ganho de informação. Para o atributo escolhido (menor entropia), cria-se um nó filho para cada valor possível do atributo. Repete-se o processo para cada nó filho.

A recursão vai acontecendo até que eventualmente pode atingir o caso de paragem, se:

1. cada elemento no subconjunto pertence à mesma classe, esse nó passa a ser uma folha pertencente à mesma classe;
2. caso não existam mais atributos a serem selecionados.

O ganho de informação usa o cálculo da entropia como medida de impureza. Este algoritmo pioneiro utiliza a entropia como medida para determinar quão boa é uma condição de teste. Para isso temos de comparar o grau de entropia antes da divisão no nó pai com o grau de entropia após a divisão nos nós filhos. O atributo que gerar uma diferença maior é o escolhido como condição de teste.

O cálculo do ganho obtém-se com a seguinte fórmula, onde n representa o número de valores do atributo (número de nós filhos), N é o número de objetos do nó pai e N(vj) é o número de exemplos do nó filho (vj).

$$\text{ganho} = \text{entropia(pai)} - \sum_{j=1}^{n}\left[\frac{N(vj)}{N} * entropia(vj)\right] \quad (13)$$

O cálculo da entropia é definido pela seguinte fórmula, onde p(i/nó) é a fração dos registos pertencentes à classe i de um determinado nó e c o número de classes.

$$\text{entropia(nó)} = -\sum_{i=1}^{c} p\left(\frac{i}{nó}\right).\log_2\left(p\left(\frac{i}{nó}\right)\right) \quad (14)$$

Este algoritmo apresenta algumas limitações como o facto de apenas lidar com atributos categóricos não-ordinais, não sendo possível utilizar atributos contínuos sem os discretizar antes. Também não apresenta forma de tratar os valores desconhecidos, ou seja, todos os valores devem ser conhecidos.

Tendo em conta que nos contextos reais a probabilidade de ter grande quantidade de valores desconhecidos é grande, isto torna-se numa grande limitação. O algoritmo é muito parecido ao algoritmo TDIDT pois é baseado nele, mas apresenta como critérios de seleção de atributos as fórmulas do ganho e da entropia. Para terminar, este algoritmo utiliza o ganho mas não considera o número de divisões, o que pode resultar em árvores mais complexas.

4.4 C4.5

O presente algoritmo representa uma significativa evolução relativamente ao anterior (ID3). Tem capacidade de lidar com atributos categóricos (ordinais ou não-ordinais) e com atributos contínuos. Utiliza a razão de ganho em vez do ganho, o que permite obter árvores menos complexas e mais precisas. A razão de ganho resulta da divisão do ganho por a entropia do nó em questão.

$$\text{razão de ganho(nó)} = \frac{ganho}{entropia\ (nó)} \tag{15}$$

Atualmente já existe o C5.0 que é baseado no 4.5, utiliza menos memória e cria conjuntos de regras menores do que o C4.5, o qual permite maior precisão. O algoritmo ID3 é muito suscetível ao ruído. Se os dados contém grande nível de ruído o ID3 vai ter erros superiores a 50%. Para combater esse problema surgiu então o algoritmo C4.5. Este algoritmo permite a utilização de pós-poda com base no fator custo-complexidade. Permite ainda representar valores desconhecidos que são tratados de forma especial, pois esses valores não são considerados nos cálculos de ganho e entropia.

Segue a base do TDIDT, pois é guloso e divide para conquistar.

4.5 CART

Este algoritmo tem a clara vantagem de produzir árvores de regressão e de classificação. Se o atributo fornecido é nominal então produz uma árvore de classificação, caso seja um atributo continuo a árvore gerada é de regressão.

Utiliza a técnica de pesquisa exaustiva na divisão dos atributos contínuos e o método de pós-poda por meio de redução do fator custo-complexidade. Tem grande capacidade de pesquisa de relações entre os dados, mesmo quando elas não são evidentes, produzindo árvores simples e legíveis do tipo binário. Usa um critério de divisão baseado no índice de *Gini*.

O índice de *Gini* mede a heterogeneidade dos dados, se selecionarmos dois itens de uma população aleatoriamente, então eles devem ser da mesma classe e a probabilidade para isto é 1 se a população é pura.

$$I_{g(p)} = \sum_{i=1}^{j} (p_i - p_i^2) \tag{16}$$

em que p de índice i corresponde ao item i e j ao número total de classes.

4.6 Ruído

Se existirem dois ou mais exemplos com a mesma descrição (em termos de atributos), mas com diferentes classificações, não é possível encontrar uma árvore de decisão consistente com os dados.

Este tipo de problema pode ser resolvido se cada folha indicar a classificação por maioria ou fizer uma estimativa da probabilidade para cada classificação.

4.7 Overfitting

Acontece quando o algoritmo de aprendizagem continua a desenvolver hipóteses que reduzem o erro do conjunto de treino ao custo de erros presentes nos testes, ou seja, através de ruído, mas também quando o número de casos de treino é muito pequeno. Também pode ser considerado o sobreajuste excessivo do modelo ao conjunto de dados fornecido. Isto acontece sempre que a árvore estende a sua profundidade até atingir o ponto de classificar individualmente todos os exemplos do conjunto de dados de treino, o qual leva à redução da sua capacidade de generalização.

Para evitar o *overfitting* para além da inserção de casos de treino suficientes (em número suficiente para o exemplo em questão), é conveniente podar a árvore de modo a não testar atributos que não sejam relevantes. Atributos sem relevância são aqueles que oferecem um ganho de informação perto de zero.

Existem dois tipos de poda, a pré-poda e a pós-poda. A pré-poda para de crescer a árvore antes de estar totalmente classificada, enquanto que o pós-poda só para o crescimento quando a árvore está totalmente classificada. A segunda abordagem é mais bem sucedida porque não é fácil estimar com precisão quando parar de crescer a árvore. O objetivo da poda é melhorar a taxa de acerto do modelo para novos casos, os quais não foram utilizados no treino da árvore de decisão.

Boas praticas incluem o uso de conjuntos de dados de treino utilizados para validação após o pós-poda da árvore. Existem outros tipos de teste como o teste do Qui-Quadrado e o princípio do comprimento, ou seja, que para o crescimento quando a árvore o atingir. O método da pós-poda é o mais utilizado.

4.8 Valores Desconhecidos

Nas subsecções anteriores viu-se a inserção de valores corrompidos ou ruidosos. Esta subsecção visa compreender como lidar com os valores desconhecidos.

Uma das formas de tentar contornar o problema é preencher um valor desconhecido, utilizando informação fornecida pelo contexto. Outra forma é a classificação segundo atributos irrelevantes, como por exemplo no caso do lançamento dos dados, utilizar como atributo o dia ou a cor.

Também através de cálculos probabilísticos pode-se tentar adivinhar o valor correto do desconhecido. De acordo com o método de *Bayes* utiliza-se a melhor estimativa das probabilidades. Este método nem sempre é o melhor e como tal pode-

se pensar em utilizar o valor desconhecido na árvore de decisão como outro valor qualquer que seja conhecido.

4.9 Ferramentas de Desenvolvimento

Em seguida apresentam-se algumas ferramentas de desenvolvimento disponíveis atualmente.

- **R** - A linguagem de programação R permite facilmente criar árvores de decisão a partir de casos de treino. Utiliza algoritmos de partição recursiva.

- **Matlab** - Tal como em R também no *Matlab* é possível criar, treinar e executar árvores de decisão. Existem dois tipos de árvores, as árvores de classificação e de regressão. O *matlab* contém vários algoritmos para lidar com as árvores de decisão, sendo que na aprendizagem utiliza o algoritmo de CART e o algoritmo do controlo de profundidade (*Tree Depth Control*) que trata da poda.

- **Python** - Nesta linguagem de programação está disponível um *package* (DecisionTree 3.4.3) que permite a criação de árvores de decisão. Tal como nas linguagens anteriores é possível efetuar treinos e obter os resultados pretendidos.

4.10 Soluções Reais

A inteligência artificial é uma das áreas que está na "moda", e como tal apresentam-se alguns casos práticos da sua aplicação na vida real, tendo em conta apenas a utilização de árvores de decisão. Praticamente todos contactamos diariamente com este tipo de aprendizagem máquina, sem saber que o fazemos, como por exemplo nos filtros de email. Em seguida, apresentam-se alguns casos onde se utiliza as árvores de decisão.

- **Classificação das imagens do telescópio *Hubble Spac*e** - A árvore de decisão utilizada tem a tarefa de aprender a distinguir entre estrelas e raios cósmicos, em imagens recolhidas pelo telescópio.
  Para além da necessidade de alta precisão também se necessita de rapidez a lidar com grande número de classificações. Neste caso foi utilizado um conjunto de treino com cerca de 2211 imagens. Este exemplo torna-se interessante já que mostra que árvores com não mais do que 9 nós podem atingir precisões superiores a 95\%. Surge ainda a possibilidade de aumentar essa precisão com a aplicação de métodos para eliminar o ruído.[6]

- **Caraterização de tumores Leiomiomatosos** - Um leiomioma, também conhecido por fibroides, é um tumor benigno presente no músculo liso que muito raramente se torna cancro (0.1%). Ocorre mais frequentemente no útero, intestino delgado ou no esófago. A aplicação deste tipo de ferramentas é muito útil nestes casos em que a avaliação feita por um patologista seria bastante complicada. O trabalho de Decaestecker et al. as árvores de decisão são aplicadas para distinguir entre tumores benignos e malignos. O treino foi realizado com o algoritmo C4.5 com 23 casos pré-processados por 3 patologistas.[4]

- **Filtrar emails (SPAM) -** Grande parte das técnicas de filtragem são baseadas no texto por métodos de categorização. O algoritmo utilizado neste exemplo é o C4.5, estudado anteriormente.[5]

5. Conclusão

A aprendizagem máquina é um subcampo da ciência da computação que evoluiu do estudo de reconhecimento de padrões e da teoria do aprendizado computacional em inteligência artificial.

Uma árvore de decisão é um exemplo muito simples com grande sucesso de um algoritmo de aprendizagem perfeitamente enquadrado na área de aprendizagem indutiva. São comumente usadas em pesquisas, mais especificamente na análise de decisões e na identificação de uma estratégia com maior probabilidade de atingir um objetivo.

Na verdade, qualquer função Booleana pode ser representada como uma árvore de decisão. As árvores de decisão podem representar funções de modo muito mais compacto. Atualmente, já se aplicam em diversas áreas de elevada importância, como por exemplo, na saúde. Uma das vantagens das árvores de decisão é a facilidade de interpretação.

Por fim, as redes neuroniais artificiais, idealizadas a partir do cérebro humano, representam um método bastante fiável e com uma grande área aplicacional, sendo um dos métodos mais predominantemente utilizados em áreas como o reconhecimento de imagens.

Apesar de, comparativamente com as árvores de decisão, as redes neuronais artificiais serem potencialmente mais lentas, devido ao facto das árvores decisão descartarem informação que não é considerada útil, e de não apresentaram tanta interpretabilidade, conseguem atingir um maior nível de precisão. Isto deve-se, sobretudo, à não linearidade das redes neuronais, o que lhes permite modelar funções mais arbitrárias.

A aprendizagem por reforço encontra-se num paradigma diferente os outros dois métodos, que são usualmente utilizadas num paradigma de aprendizagem supervisionada. A aprendizagem por reforço preocupa-se mais em ter uma maior

perceção sobre o ambiente, através de interações com este, procurando otimizar uma sequência de ações, tendo em conta o estado em que se encontram. Os outros métodos analisam os dados que lhes são fornecidos, realçando padrões e/ou características relevantes para o problema.

References

1. Sutton, Richard S. and Barto, Andrew G., Reinforcement Learning: An Introduction, 1:15-47:142, 2012.
2. Haykin, S., Neural Networks - A Comprehensive Foundation, 23:88, 1999.
3. Cortez, P. and Neves, J., Redes Neuronais Artificiais, 3:16-23:27, 2000.
4. Decaestecker, C., Remmelink, M., Salmon, I., Camby, I., Goldschmidt, D., Petein, M., Van Ham, P., and Pasteels, J. Methodological aspects of using decision trees to characterize leiomyomatous tumors. Cytometry 24:83–92, 1995.
5. V.Christina , S.Karpagavalli , G.SuganyaAuthor, F., Email Spam Filtering using Supervised Machine Learning Techniques, 2010.
6. Salzberg, S., Chandar, R., Ford, H., Murthy, S. K., and White, R. Decision trees for automated identification of cosmic-ray hits in Hubble Space Telescope images. Publ. Astron. Soc. Pacific 107:279–288, 1995.
7. Predictive Analytics Today, Top 30 Artificial Neural Network Software, https://www.predictiveanalyticstoday.com/top-artificial-neural-network-software/, consultado em 21/10/2017
8. International Business Times, What is Prisma? Behind the hip new app that turns your photos into impressionist paintings using AI, http://www.ibtimes.co.uk/what-prisma-behind-hip-new-app-that-turns-your-photos-into-impressionist-paintings-using-ai-1571191, consultado em 22/10/2017
9. Nvidia Developer, Explaining How End-to-End Deep Learning Steers a Self-Driving Car, https://devblogs.nvidia.com/parallelforall/explaining-deep-learning-self-driving-car/, consultado em 22/10/2017
10. Microsoft, https://news.microsoft.com/pt-br/dividir-e-conquistar-como-os-pesquisadores-da-microsoft-usaram-ia-para-dominar-ms-pac-man/, consultado em 23/10/2017
11. Diário de Noticias, https://www.dn.pt/sociedade/interior/computador-derrota-campeao-chines-de-jogo-de-tabuleiro-go-8499621.html, consultado em 23/10/2017
12. Quinlan, J. R., “Induction of Decision Trees”, Machine Learning 1: 81-106, Kluwer Academic Publishers, 1986