Sistemas de Aprendizagem através de Reinforcement Learning, Particle Swarm Optimization e Support Vector Machines

Universidade do Minho, Braga, Portugal

Resumo A automatização é uma das áreas informáticas que tem vindo a crescer e a evoluir rapidamente ao longo dos anos. Esta já está presente no nosso dia-a-dia e incutida em milhares de empresas de todos os setores. Esta automatização é levada a cabo por modelos de inteligência artificial que utilizam algoritmos capazes de aprender de forma interativa, ao realizar uma análise de dados complexa com alta precisão. Desta forma é possível identificar padrões presentes num grande volume de dados de modo a que o sistema que implementa tais algoritmos consiga traçar cenários futuros com precisão e velocidade, estimando assim de forma acertada o comportamento de determinado produto no mercado, de forma a conseguir uma solução altamente fiável para a resolução de vários problemas presentes na indústria e no nosso quotidiano. Existem vários tipos de aprendizagem que foram seriamente explorados e são usados diariamente. Entre estes, iremos abordar três em específico, descrevendo o seu método de aprendizagem, quais as ferramentas usadas nas suas implementações e qual o seu impacto no mercado.

Palavras-chave: Sistemas de Aprendizagem · Reinforcement Learning · Particle Swarm Optimization · Support Vector Machine

1 *Reinforcement Learning*

1.1 Descrição característica

A aprendizagem por reforço (*Reinforcement Learning*) consiste em aprender o que fazer, em determinadas situações, de modo a aumentar o sinal numérico de recompensa. À entidade que está a aprender um determinado comportamento, não lhe são fornecidas quaisquer informações sobre como proceder, ou sobre que ações dever tomar face a um novo problema. Ao invés, a entidade deve descobrir por si mesma, através de várias tentativas, quais serão as melhores ações que deve tomar de modo a maximizar o valor da recompensa. Este conjunto de ações podem não afetar o nível de recompensa imediata, mas podem servir para recompensas futuras. Estes aspetos definem as duas características base da aprendizagem por reforço, ou seja, tentativa e erro e a recompensa adiada [1].

Contrariamente à aprendizagem supervisionada, a aprendizagem por reforço não necessita de um conjunto de dados de treino ou exemplos de casos anteriores fornecidos por um supervisor. Isto dá vantagem à aprendizagem por reforço no

que toca a problemas em que não há conhecimento nem exemplos passados, ou seja, problemas que devem ser resolvidos através da experiência.

A aprendizagem por reforço também é diferente da aprendizagem não supervisionada. Na aprendizagem não supervisionada os dados recebidos não estão organizados e devidamente tratados. Assim, neste tipo de aprendizagem é procedida a deteção de padrões e consequente organização e estruturação dos dados. Já na aprendizagem por reforço tenta-se maximizar o valor da recompensa. Estes dois tipos de aprendizagem tendem a ser confundidos devido ao facto de ambos não receberem exemplos corretos de comportamentos passados.

Uma noção fundamental, no que toca à aprendizagem por reforço, é a distinção entre "*exploitation*"e "*exploration*". Para obter uma elevada recompensa, um agente deve preferir ações que já realizou no passado que levaram a atingir uma determinada meta ou objetivo. Mas, para ter descoberto ações ou comportamentos deste género, o agente teve que tentar seguir certos comportamentos que nunca tinha experimentado. O agente deve então verificar tudo o que realizou no passado de modo a obter uma recompensa (*exploit*), mas também deve explorar o ambiente em que está inserido de modo a determinar quais os melhores comportamentos a realizar no futuro (*explore*). O grande dilema destes dois conceitos é que nenhum deles é conseguido sem que se falhe numa determinada tarefa. Cabe ao agente envolvido escolher, de entre todas as ações, aquelas que aparentam fornecer melhores resultados.

Outro componente muito importante é o agente envolvido na realização de objetivos num ambiente incerto e por descobrir. Todos os agentes envolvidos na aprendizagem por reforço têm objetivos específicos, reagem a estímulos provenientes do ambiente onde se encontram e escolhem ações que influenciam esse mesmo ambiente. Estes agentes podem ser vistos como um sistema completo, como por exemplo um organismo ou um *robot*, ou como componentes independentes de um sistema. Neste último caso, os agentes envolvidos interagem entre si e por sua vez com todo o sistema no qual estão inseridos.

Finalmente, a aprendizagem por reforço é o tipo de aprendizagem mais semelhante com a dos seres humanos. Desde o início da nossa vida que vamos resolvendo casos e descobrindo qual ou quais as melhores ações que devemos realizar através da experiência e de estímulos (dor e prazer, por exemplo). Isto acontece para todos os seres vivos em geral. Estas noções de tentativa e erro, interação com o meio ambiente e maximização do valor da recompensa após uma determinada ação, são as características base de uma aprendizagem por reforço.

1.2 Capacidade de aprendizagem

Além do ambiente e dos agentes, podem-se identificar mais quatro subelementos pertencentes à aprendizagem por reforço, sendo estes: política, sinal de recompensa, função de valor e, opcionalmente, o modelo do ambiente [1].

A política define, essencialmente, o modo de comportamento dos diversos agentes num determinado momento. Isto traduz-se como um mapeamento entre os diversos estímulos recebidos do ambiente onde os agentes estão inseridos e as ações que estes agentes devem executar face a esses estímulos. Uma política

pode ser definida através de uma função pouco complexa ou pode envolver uma computação extremamente sofisticada e extensa. De qualquer forma, a política é uma das bases fundamentais da aprendizagem por reforço e é suficiente, por si só, para caracterizar o comportamento dos agentes.

O sinal de recompensa caracteriza-se como sendo o valor da recompensa recebido após uma determinada ação num determinado estado do ambiente. O objetivo de um determinado agente será a maximização deste valor recebido ao longo do tempo. Este valor recebido é suficiente para alterar uma determinada política de comportamento de um determinado agente. Se certas ações forem presenteadas com baixos valores de recompensa, então é muito provável que a presente política que define o comportamento de um agente seja alterada. Em geral, os sinais de recompensa são definidos à custa de funções estocásticas do estado do ambiente e das ações realizadas.

O valor da função difere-se do valor do sinal da recompensa na medida em que o valor da função tem em consideração a recompensa a longo prazo. O valor da função pode ser visto como a soma total de todos os valores de recompensa que um determinado agente espera atingir no futuro a partir de um determinado estado. Assim, por oposição ao valor do sinal de recompensa, o valor da função serve para determinar ações analisando não só o presente, mas também as possibilidades de outras recompensas que podem surgir após uma determinada ação. O valor da função tem um impacto mais relevante do que o sinal de recompensa, já que é mais proveitoso uma determinada ação, ou comportamento, ser mais proveitosa a longo prazo, do que apenas num determinado momento. No entanto, torna-se claro que os valores de funções, já que estes assentam com base em previsões, são muito mais difíceis de determinar do que os sinais de recompensa.

O último elemento é o modelo do ambiente. Este modelo traduz-se como algo que imita o o comportamento do ambiente ou que permite inferir que ações e comportamentos caracterizam o ambiente. Isto é útil para prever quais serão os próximos estados após um determinada ação e um determinado estado e é essencialmente usado para a construção de um plano de tomadas de decisão e de ações a realizar no futuro.

Estes elementos descritos acima caracterizam a capacidade de aprendizagem por reforço. O objetivo traduz-se em analisar os resultados destes elementos e modificar os comportamentos dos agentes através destes mesmos resultados, de modo a aumentar o valor da recompensa final.

1.3 Ferramentas de desenvolvimento [2]

OpenAI Gym Esta ferramenta é, neste momento, é o ambiente de desenvolvimento mais popular no que toca à criação e comparação de modelos de aprendizagem com reforço. Este *tookit* foi criado em *Python* e fornece vários ambientes de simulação de inteligência artificial e suporte para efetuar treinos a agentes em diversas áreas como jogos clássicos, ciências e física. Oferece também compatibilidade com outras ferramentas de desenvolvimento nesta vertente como o *TenserFlow* [3].

TensorFlow Este popular ambiente de desenvolvimento criado pela *Google* e mantido por um elevado número de programadores oferece vários módulos de aprendizagem por reforço que podem ser facilmente modificados. Além disso esta ferramenta permite que o desenvolvimento seja realizado através de linguagens de programação bem conhecidas como o *Java*, *Python*, *C* e *JavaScript*, por exemplo [4].

Keras Esta ferramenta caracteriza-se pela sua simplicidade na criação de redes neuronais com apenas algumas linhas de código. Esta ferramenta foi desenvolvida com intuito de uma rápida experimentação e execução. É, também, compatível com outras ferramentas desta área [5].

1.4 Soluções existentes no mercado

- **Marketing e Publicidade:** Para personalizar a publicidade que chega ao utilizador mostrando somente a publicidade que lhe possa interessar, podem ser aplicados métodos de *Reinforcement Learning (RL)*.
- **Saúde:** Existem aplicações que utilizam *RL* de maneira a encontrar políticas de tratamento ótimas para os pacientes, para além de controlar as doses de medicamentos e o uso de material médico.
- **Textos e Sistemas de Diálogo:** Outra aplicação existente com base em *RL* tem como função criar resumos de textos, para além de ser também utilizado em aplicações de diálogo, aprendendo com os utilizadores para melhorarem.
- **Automatização de Industria:** Um grande exemplo de uso de *RL* na indústria vem da empresa *Google* que reduziu significativamente o uso de energia elétrica depois de aplicar *RL* no controlo de energia dos seus centros de dados.

Para além dos exemplos anteriores, existem muitos outros usos de *RL* como a pesquisa de métodos que aproximem o máximo possível as capacidades do computador às capacidades humanas em jogos, mas também a possível utilização de *RL* na aprendizagem e treino humano, no controlo financeiro de empresas ou até no controlo de semáforos de maneira a resolver problemas de trânsito [16].

2 *Particle Swarm Optimization*

2.1 Descrição característica

O *Particle Swarm Optimization* (*PSO*) é um algoritmo de otimização baseado em padrões naturais, que envolve uma procura aleatória num dado espaço por várias partículas, inicialmente desenvolvido por *James Kennedy* e *Russel Eberhart* em 1995. A ideia deste algoritmo veio do estudo de bandos de pássaros e cardumes de peixes e da maneira como estes partilham informações entre si quando procuram alimento, sendo cada elemento do grupo beneficiado com as suas próprias descobertas, tal como com as descobertas dos outros elementos do grupo [18].

No *PSO*, os pássaros são substituídos por partículas, que são inicializadas aleatoriamente no espaço onde se é pretendido encontrar a "comida", neste caso substituída pelo objetivo que pretendemos para o algoritmo. Cada partícula tem uma velocidade inicial e uma posição, sendo que a velocidade inicial é gerada aleatoriamente. O caminho que cada partícula toma é atualizado dependendo do seu valor para a melhor posição encontrada e da melhor posição e valor dos seus vizinhos. Sendo assim, o caminho do grupo vai depender de uma melhor posição de todas as partículas que o compõem, o que faz com que o grupo de partículas se vá movendo até à solução ótima.

2.2 Capacidade de aprendizagem

Para se falar mais detalhadamente do *PSO* é necessária uma primeira abordagem às fórmulas utilizadas para calcular as velocidades que as várias partículas podem ter no decorrer da simulação e para atualizar a posição de cada partícula segundo o método de implementação original do *PSO*.

$$V = V + C * rand() * (P_{best} - P) + C * rand() * (G_{best} - P) \tag{1}$$

$$P = P + V \tag{2}$$

Para uma melhor perceção daquilo que as equações significam, daremos os seguintes valores às variáveis aqui presentes. O valor C indica uma constante, à qual, na versão original do *PSO* foi dado o valor de 2; *rand()* é uma função que cria um valor aleatório entre 0 e 1; V é a velocidade da partícula nas várias dimensões do problema; P é o valor da posição da partícula nas várias dimensões; P_{best} é o melhor valor obtido anteriormente pela partícula; G_{best} é o melhor valor obtido pelo grupo ao qual a partícula pertence [15].

Tendo-se identificado as variáveis para as duas equações podemos verificar que a junção das duas equações descreve a trajetória da partícula. Enquanto que a primeira vai calcular e atualizar a velocidade da partícula, a segunda vai simplesmente incrementar o valor da velocidade calculado anteriormente à posição atual da partícula.

Sendo a primeira equação mais complexa que a segunda, pode ser dividida em três partes diferentes de maneira a facilitar a sua compreensão. Na primeira parte da equação temos o *momentum* da partícula, impedindo uma mudança brusca de velocidade, assim a nova velocidade é atualizada de acordo com a velocidade antiga da partícula. A segunda parte trata do pensamento individual da partícula, ao modificar a sua nova posição de acordo com o seu melhor valor individual, ou seja, da sua experiência. A terceira parte da equação trata de calcular o valor da velocidade dependendo do melhor valor do grupo de partículas ao qual esta pertence, sendo assim uma espécie de componente social [15].

De maneira a controlar a velocidade das partículas é necessário a criação de um limite para a velocidade(V_{max}), este valor é importante para se controlar o quanto é que as partículas se podem mover a cada iteração, ajudando a garantir que se aproximam mais do valor ideal. Originalmente, este é o único valor que necessita ser criado pelo utilizador para o funcionamento correto do algoritmo.

As etapas que fazem parte do algoritmo *PSO* são bastante simples e podem ser divididas em cinco partes:

- Inicializar e definir uma população de partículas, cada uma com uma posição aleatória ao longo da zona de procura onde se pretende usar o método, cada partícula tem de ter definidas as suas velocidades nas N dimensões do espaço. Cada uma destas partículas avalia posteriormente a sua posição relativamente ao resultado ótimo calculado pela função utilizada para a resolução do problema.
- É realizada uma comparação entre o valor atual da partícula e o seu melhor valor obtido (P_{best}). Se o valor atual for melhor que o P_{best}, então P_{best} é substituído pelo valor atual da partícula. ($P_i = X_i$ nas N-dimensões do problema).
- Após o calculo do P_{best} da partícula, esta vai comparar o seu valor com as partículas vizinhas, procurando pela que tem o melhor valor. Ao encontrar a que tem o melhor valor vai associar esse valor a uma variável que vai indicar o melhor valor obtido pelo grupo (G_{best}).
- Tendo os melhores valores obtidos até ao momento, tanto o valor individual (P_{best}) como o valor do grupo (G_{best}), a partícula usa equação explicada anteriormente e usa-a para calcular a sua nova velocidade.
- Após o ajuste de velocidade a partícula volta a repetir a comparação e a realizar os cálculos que precisa até atingir o critério que lhe foi dado inicialmente, atingir um valor ótimo ou tendo chegado a um número máximo de iterações.

2.3 Ferramentas de desenvolvimento

PSOLet PSOLeT é uma ferramenta de aprendizagem que implementa o algoritmo *PSO*. A ideia principal deste *software* é facilitar o utilizador na implementação deste algoritmo com novos parâmetros. Tais parâmetros que são possíveis editar, por este *software* desenvolvido pela *MathWorks*, para modificar o comportamento deste algoritmo são: o número de partículas, o número de gerações do algoritmo (critério de paragem). No entanto, não apresenta um resultado gráfico muito sofisticado [18].

PSOVis O *PSOVis* é uma ferramenta que permite a visualização do processo de aprendizagem dos agentes. Com o intuito de ajudar o utilizador no funcionamento deste algoritmo, este programa permite a alteração de vários parâmetros como, o peso inercial inicial e final, coeficiente cognitivo e social, velocidade máxima, número de partículas e numero máximo de iterações, tornando assim mais fácil o acompanhamento da evolução do algoritmo de forma visual [19].

Particle Swarm Optimization Visualization O *PSO Visualization* é uma *applet JAVA* que demonstra visualmente a procura de um valor-objetivo por um enxame de partículas. A cada iteração, um novo ambiente tridimensional é

gerado, onde um enxame se movimenta em busca do valor. A visualização das partículas no espaço é simulada por diferentes cores que demonstram a posição atual e prévia das partículas [19].

2.4 Soluções existentes no mercado

Energia reativa e Controlo de Tensão Elétrica Em geral, as redes de energia elétrica estão propensas a alterações súbitas na sua configuração, tal acontece quando as linhas e as cargas são ligadas e desligadas. Neste sistema dinâmico, manter a tensão elétrica dentro de um intervalo que seja consumível pela população é uma das mais importantes tarefas de tais instalações. Para resolver tal problema, foram criados dispositivos capazes de controlar os geradores, transformadores e reatores elétricos de forma a gerar a quantidade de energia reativa que é capaz de manter a tensão em níveis desejáveis. Este método tem de assegurar a tensão de segurança e minimizar a perda de energia e os custos do processo [13].

Fornecimento Económico / *Economic Dispatch* (*ED*) O *Economic Dispatch* (*ED*) é um problema histórico nos sistemas de energia. O objetivo é encontrar um ótimo *output* de unidades de energia gerado de forma a minimizar os custos operacionais de tal processo. Para satisfazer um custo total mínimo, os geradores com menores custos devem ser usados em primeiro lugar, e em seguida tem de ser fazer um balanceamento entre os custo e a produção de energia dos restantes. É possível obter uma solução de forma automatizada e com pouco poder de computação usando *PSO* como algoritmo. O sistema informático foi carregado com diversos geradores virtuais. Cada um possuindo várias variáveis do ambiente como custo, energia produzida, calor, etc. Com este modelo, foi possível obter uma solução ótima para o problema [14,15].

Foram abordadas várias aplicações do *Particle Swarm Otimization* aplicadas em sistemas elétricos [13]. No entanto, as capacidades deste algoritmo não estão limitadas a tais soluções. Existem outra aplicações deste algoritmo como comportamento social, banca, entre outros.

3 *Support Vector Machines*

3.1 Descrição característica

Support Vector Machines (*SVM*) é um conjunto de modelos correspondentes a algoritmos de aprendizagem supervisionada, utilizado principalmente para a classificação de dados na área de *machine learning*.

As bases do conceito surgiram num artigo desenvolvido por Vladimir Vapnik e Alexander Lerner em que introduziram o algoritmo de *Generalized Portrait* (no qual os algoritmos do *SVM* atual se baseiam) [7].

No contexto da classificação de dados, o objetivo fundamental de qualquer *SVM* é separar elementos de um mesmo *dataset* em dois conjuntos de dados, utilizando, para isso, um hiperplano divisório.

Hard-margin SVM Partindo de um *dataset* de treino que contém pontos de dois grupos sob a forma de (x_1, y_1), ..., (x_n, y_n) em que x_i corresponde a um conjunto de características do elemento de teste e y_i a uma etiqueta que indica a qual dos grupos pertence o elemento, as *SVM* focam-se em encontrar um hiperplano que divide o conjunto de elementos em dois conjuntos independentes correspondentes a estes mesmos grupos. A **Figura 1** ilustra este exemplo [10].

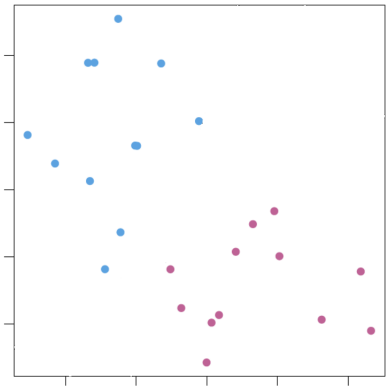

Figura 1. Representação de dois conjuntos de dados [10]

Como estamos num espaço bidimensional, o hiperplano de separação corresponderá a uma reta. Na **Figura 2**, podemos verificar que existem inúmeras possibilidades para uma reta que separe estes dois conjuntos [10].

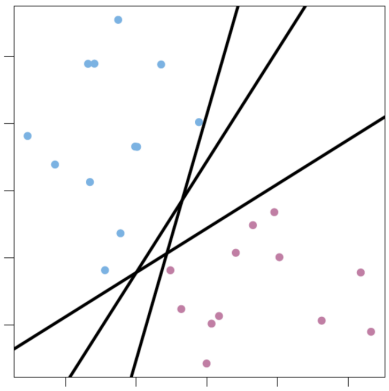

Figura 2. Retas que separam os dois conjuntos [10]

Ao utilizar um *SVM* do tipo *hard-margin*, é recorrido a um algoritmo de margem máxima (*maximum margin classifier* [6]) para calcular a reta ótima que separa os dois conjuntos, maximizando a margem que é obtida através o espaço entre duas retas paralelas que passam pelos elementos de fronteira de cada um dos conjuntos. Após serem calculadas as retas que fornecem a maior margem, o hiperplano de separação dos conjuntos corresponde à reta paralela a estas e que se encontra a meio da região da margem, como se pode verificar pela **Figura 3** [10].

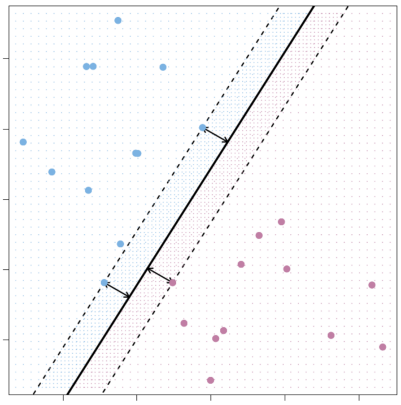

Figura 3. Hiperplano (reta) de separação dos dois conjuntos [10]

Neste processo de cálculo do hiperplano (reta), são utilizados os pontos fronteira – chamados *support vectors* –, que dão nome ao algoritmo.

Soft-margin SVM Nem sempre no mundo real, os dados se apresentam tão bem linearmente separados, pois normalmente existe ruído estatístico (*noise*) e elementos isolados (*outliers*). Assim, em 1995, Vladimir Vapnik e Corinna Cortes introduzem o conceito de *Soft-margin SVM* com a inclusão de variáveis de folga (muitas vezes denominado de fator "C"). Através da definição do valor deste fator, há um *tradeoff* entre a capacidade de generalização dos dados e ausência de viés (*bias*) [9].

SVM não linear Apesar de tudo o que foi dito, os dois algoritmos anteriores correspondem a classificadores lineares, não sendo aplicáveis em muitas situações em que o conjunto de dados não é de todo linearmente separável.

Assim sendo, Vapnik et al. introduziu a criação de classificadores não lineares através do uso da técnica do *kernel trick* [8] em que se ampliam as dimensões do espaço do conjunto de dados, procurando acomodar uma separação não linear (hiperplano) dos dois conjuntos. A figura **Figura 4** ilustra um exemplo em que se utiliza uma função de ampliação ϕ tal que

$$\begin{aligned} \phi : \mathbb{R}^2 &\rightarrow \mathbb{R}^3 \\ (x_1, x_2) &\rightarrow (x_1^2, x_2^2, \sqrt{2}x_1x_2) \end{aligned} \tag{3}$$

para ampliar o espaço do *dataset*.

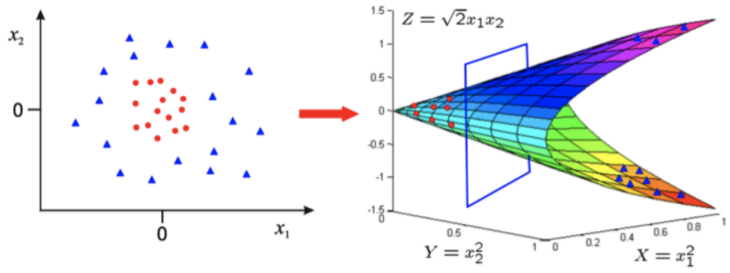

Figura 4. Uso do *kernel trick* para encontrar hiperplano de separação *(fonte)*

3.2 Capacidade de aprendizagem

Como em qualquer sistema de aprendizagem supervisionada, a aprendizagem em *SVM* está associada ao cálculo de uma função que mapeia um *input* num *output*, partindo de pares *input-output* fornecidos *a priori* num conjunto de dados de treino.

Em particular, no caso das *SVM* esta função está associada ao cálculo do hiperplano de separação (linear ou não linear) entre os dois grupos de elementos do conjunto de dados fornecido. Para isso – e como descrito na subsecção anterior –, são utilizados os pontos de fronteira de cada um dos grupos, cujos vetores correspondem aos vetores de suporte (*support vectors*) que dão nome ao algoritmo.

Assim sendo, ao serem introduzidos elementos de dados novos, estes são colocados em cada um dos grupos em função da sua posição relativamente ao hiperplano calculado na fase de treino (que, naturalmente, teve por base apenas os elementos pertencentes ao *dataset* inicial).

Numa futura utilização, ao serem fornecidos novos elementos (i.e. novos pares *input-output*), pode ser alterado o hiperplano de separação, caso estes novos elementos correspondam a novos pontos fronteira de um – ou de ambos – os grupos. Tal acontece porque, como se alteraram os pontos de fronteira, deu-se também uma alteração nos vetores de suporte de um (ou de ambos) os conjuntos de dados.

Ocorre, desta forma, um processo de aprendizagem.

3.3 Ferramentas de desenvolvimento

LIBSVM É uma ferramenta *open source* escrita em *C++* e desenvolvida pela Universidade Nacional de Taiwan para a utilização de *SVM* em contextos de

classificação linear e não linear e regressão estatística. Além disso, suporta a classificação em ambientes com mais do que duas classes de dados (*multi-class classification*) [11].

Scikit-learn É uma ferramenta *open source* desenvolvida na linguagem *Python* que surgiu como um projeto do *Google Summer of Code* de David Cournapeau e que conta atualmente com quase 1200 contribuidores no seu repositório no *GitHub*. Suporta classificação e regressão utilizando *SVM*, fazendo uso de módulos bem conhecidos de *Python* como *NumPy*, *SciPy* e *matplotlib* [12].

3.4 Soluções existentes no mercado

Como referido em capítulos anteriores, *Support Vector Machines* está sustentado em algoritmos de aprendizagem supervisionados. O objetivo do uso do *SVMs* é classificar corretamente dados que não foram analisados. Existe um vasto número de aplicações em diversas áreas usando *SVMs* [17]. Algumas das soluções mais comuns usando este algoritmo são:

- **Reconhecimento facial:** O algoritmo classifica partes da imagem, e classifica partes como face e outras como não-face, desenhando um quadrado à volta das partes classificadas como "face".
- **Categorização de texto:** É possível inferir categorias de um determinado texto presente em documentos de modo a poder classificar o dito cujo. Documentos previamente analisados são usados para que novos possam ser classificados em diferentes categorias. O documento é mapeado em diferentes variáveis, de modo a comparar com resultados prévios.
- **Classificação de imagens:** O uso de *SVMs* oferece uma maior precisão na classificação de imagens. Há um aumento na precisão em comparação com as tradicionais técnicas usadas. Traços de vários segmentos *x por x pixels*, são extraídos da imagem e classificados sendo uma face ou não.
- **Bioinformática:** Inclui a classificação de cancro e proteínas. São usados *SVMs* para a identificação de genes, pacientes a partir de genes e outras questões biológicas.

Referências

1. Richard S. Sutton and Andrew G. Barto, Título: Reinforcement Learning: An Introduction, Second edition, in progress, November 5, 2017
2. Top 5 tools for reinforcement learning, `https://hub.packtpub.com/tools-for-reinforcement-learning/`. Último acesso: 14/10/2018
3. Gym, `https://gym.openai.com/`. Último acessso: 14/10/2018
4. TensorFlow, `https://www.tensorflow.org/`. Último acesso: 14/10/2018
5. Keras, `https://keras.io/`. Último acesso: 14/10/2018
6. Cristianini, N. and Shawe-Taylow, J., "An Introduction to Support Vector Machines and other kernel-based learning methods", Cambridge University Press, 2000

7. Vapnik, V. and Lerner, A., "A Pattern Recognition Using Generalized Portrait", 1963
8. Vapnik, V. et al., "A Training Algorithm for Optimal Margin Classier", 1992
9. Vapnik, V. and Cortes, C., "Support-Vector Networks", 1995
10. James, G. et al., "An Introduction to Statistical Learning"
11. LIBSVM, `www.csie.ntu.edu.tw/~cjlin/libsvm/`. Último acesso: 15/10/2018
12. Scikit-learn, `http://scikit-learn.org`. Último acesso: 15/10/2018
13. A Survey of Particle Swarm Optimization Applications in Electric Power Systems, M. R. AlRashidi, Student Member, IEEE, and M. E. El-Hawary, Fellow, IEEE
14. Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems, Jean-Carlos Hernandez, and Ronald G. Harley, Fellow, IEEE
15. Yuhui Shi, "Particle Swarm Optimization", IEEE Neural Networks Society, February 2004
16. Practical applications of reinforcement learning in industry, `www.oreilly.com/ideas/practical-applications-of-reinforcement-learning-in-industry`. Último Acesso: 15/10/2018
17. Real-Life Applications of SVM, `https://data-flair.training/blogs/applications-of-svm/`. Último Acesso: 15/10/2018
18. SwarmViz: An Open-Source Visualization Tool for Particle Swarm Optimization, Guillaume Jornod, Ezequiel Di Mario, Iñaki Navarro and Alcherio Martinoli, Distributed Intelligent Systems and Algorithms Laboratory
19. PSOView: Simulator Web para o Algoritmo de Otimizaçao Global Particle Swarm Optimization, Ricaro Grunitzki, Frenando dos Santos