# Sistemas de Aprendizagem

Universidade do Minho, Braga, Portugal
10 de novembro de 2020

**Resumo** Neste artigo serão aprofundados três Sistemas de Aprendizagem, nomeadamente *Reinforcement Learning*, *Particle Swarm Optimization* e *Support Vector Machines*. Para cada um destes temas, será apresentada uma descrição característica, o modo como exibem a capacidade de aprendizagem, as principais ferramentas de desenvolvimento existentes e algumas soluções presentes no mercado.

**Keywords:** *Machine Learning · Reinforcement Learning · Particle Swarm Optimization · Support Vector Machines.*

## 1 Introdução

Os Sistemas de Aprendizagem abordados neste artigo têm diversas aplicações, sejam elas académicas, empresariais ou até para uso pessoal. Permitem o processamento e análise de dados de modo a automatizar e tornar mais fácil o dia-a-dia de milhares de pessoas. A área de aplicação destes Sistemas de Aprendizagem tem vindo a crescer e evoluir exponencialmente ao longo dos últimos anos. Este artigo aborda os temas de *Reinforcement Learning*, *Particle Swarm Optimization* e *Support Vector Machines*, descrevendo as suas características, o modo como exibem a capacidade de aprendizagem, as ferramentas de desenvolvimento existentes e algumas soluções presentes no mercado.

## 2 *Reinforcement Learning*

### 2.1 Descrição característica

A Aprendizagem por Reforço (*Reinforcement Learning*) é uma subclasse de Sistemas de Aprendizagem que difere de outras subclasses, como a Aprendizagem com Supervisão (*Supervised Learning*) e a Aprendizagem sem Supervisão (*Unsupervised Learning*), devido ao facto de esta não necessitar de dados predefinidos nem da interferência de um programador, sendo crucial em problemas onde não existem conhecimento e exemplos anteriores e/ou em problemas onde o agente tem de interagir com um ambiente dinâmico.

Esta baseia-se no comportamento humano, onde a experiência é adquirida através da execução de uma ação e do estudo da resposta baseada nessa ação com a finalidade de maximizar o sinal numérico de recompensa. Assim sendo, podemos

considerar que o método de tentativa e erro e a recompensa adiada são as duas características essenciais deste modelo.

Um dos principais desafios da Aprendizagem por Reforço é a existência de conflitos entre *exploitation* e *exploration*. No que toca a *exploitation*, a escolha baseia-se na informação que o agente dispõe no momento, ou seja, para uma determinada situação o agente escolhe a ação com maior recompensa. À primeira vista parece ser uma boa decisão, mas esta impede que o agente encontre melhores ações para um determinado estado. Quanto a *exploration*, a escolha permite obter novas informações sobre o ambiente, com o objetivo de alcançar níveis de desempenho maiores no futuro, bem como ter uma visão mais abrangente do ambiente onde está inserido, sempre com o intuito de maximizar o seu desempenho. O grande dilema de ambos é que nenhum deles é conseguido sem falhar numa determinada tarefa.

A Aprendizagem por Reforço, apesar de ser demorada e de ser uma solução aplicável a um conjunto limitado de casos, é a maneira mais provável de tornar uma máquina criativa, uma vez que obtém novas e inovadoras soluções para realizar as suas tarefas.

### 2.2 Modo como exibe a capacidade de aprendizagem

Em *Reinforcement Learning* a aprendizagem é feita com o princípio de aprender com a experiência, ou seja, o agente é posto num ambiente e explora-o, atuando nesse ambiente. As ações do agente no ambiente alteram o estado e é na observação deste novo estado que se verifica se a sua ação foi boa ou má. Consequentemente, é atribuída uma recompensa ou punição, respetivamente. Assim, é possível afirmar que *Reinforcement Learning* é dominado pelo método de tentativa e erro onde as ações do agente influenciam não só o ambiente, mas também as ações futuras.

De modo a aprofundar melhor o modo de aprendizagem deste sistema, é necessário introduzir subelementos sendo eles a política, o sinal de recompensa, a função de valor e o modelo do ambiente.

A política de um agente é simplesmente a estratégia do agente num certo estado. Num dado estado, um agente vai ter um certo conjunto de ações que irá tomar para atingir o seu objetivo. De uma forma mais formal, deveríamos primeiro definir os Processos de Decisão de Markov como um tuplo de conjuntos de estados (S), conjuntos de ações (A), uma matriz de transição (P), uma função de recompensa (R) e um fator de desconto (y). Assim, a política reduz-se a $\pi$, uma distribuição de probabilidade de ocorrência de ações dependendo dos estados.

O sinal de recompensa é o valor da recompensa recebida depois de o agente executar uma determinada ação num determinado estado. Este valor pode alterar uma política de comportamento, caso certas ações recebam baixos valores

de recompensa. Os sinais de recompensa são definidos por funções estocásticas que recebem como argumentos o estado do ambiente e as ações realizadas pelo agente em questão.

A função de valor pretende determinar a recompensa a longo prazo, o que a distingue do sinal de recompensa. Esta serve para determinar ações avaliando o presente e outras recompensas provenientes de ações no futuro. Assim, esta função tem um impacto de extrema relevância já que calcula que ações tomar de modo a obter uma melhor recompensa no futuro, ao contrário do sinal de recompensa que tem em consideração apenas o momento.

O modelo do ambiente imita o comportamento do ambiente e permite uma caracterização do mesmo. Deste modo, é possível inferir que ações e comportamentos caracterizam o ambiente, o que é útil para prever próximos estados tendo em conta as ações possíveis.

Com estes dados é possível construir um plano de ações e decisões a tomar no futuro.

### 2.3 Ferramentas de desenvolvimento existentes

Nesta secção, veremos as mais promissoras ferramentas e bibliotecas *open source* para começar a construir projetos com *Reinforcement Learning*, tais como:

- ***OpenAI Gym***: é o ambiente mais popular de desenvolvimento e comparação de modelos de Aprendizagem por Reforço. É compatível com bibliotecas de grande dimensão computacional como *TensorFlow* e oferece suporte para treino de agentes usados em jogos clássicos como *Atari*, usando um ambiente de simulação de Inteligência Artificial baseado em *Python*;

- ***TensorFlow***: esta é uma outra biblioteca *open-source* bem conhecida da *Google*, usada por mais de 95.000 programadores todos os dias nas áreas de processamento de linguagem natural, *chatbots* inteligentes, robótica e muito mais. A comunidade *TensorFlow* desenvolveu uma versão estendida chamada *TensorLayer*, que fornece módulos de *Reinforcement Learning* que podem ser facilmente customizados e montados para enfrentar os desafios de *Machine Learning* do mundo real;

- ***Keras***: apresenta simplicidade na implementação de Redes Neuronais com apenas algumas linhas de código de rápida execução. É também compatível com outras ferramentas desta área;

- ***DeepMind Lab***: é uma plataforma *Google* 3D com personalização para pesquisa de Inteligência Artificial baseada em agentes. É utilizada para entender como agentes inteligentes auto-suficientes aprendem tarefas complicadas em ambientes grandes parcialmente observados. Com a vitória do seu programa

*AlphaGo* contra jogadores profissionais de *Go*, no início de 2016, a plataforma *DeepMind* chamou a atenção do público. Com os seus três centros espalhados por Londres, Canadá e França, a equipa da *DeepMind* concentra-se nos fundamentos básicos de Inteligência Artificial, que incluem a construção de um único sistema de Inteligência Artificial apoiado em métodos de última geração e em Aprendizagem por Reforço distribuída.

## 2.4 Soluções existentes no mercado

Atualmente, existem bastantes aplicações e produtos que utilizam *Reinforcement Learning*. Listam-se alguns de seguida.

### Carros autónomos

Vários artigos propuseram a utilização de *Reinforcement Learning* na condução autónoma. Nos carros autónomos, existem vários aspetos a considerar, como limites de velocidade em vários locais e impedimento de colisões.

Algumas das tarefas da condução autónoma onde a Aprendizagem por Reforço pode ser aplicada incluem a otimização da trajetória, o planeamento do movimento e as políticas de aprendizagem baseadas em cenários para autoestradas.

Por exemplo, o estacionamento pode ser executado aprendendo políticas de estacionamento automático, a mudança de faixa pode ser alcançada usando *Q-Learning* e as ultrapassagens podem ser implementadas aprendendo políticas de ultrapassagem, evitando colisões e mantendo uma velocidade constante feita a ultrapassagem.

### Comércio e finanças

Os métodos de previsão de séries temporais supervisionados podem ser usados para prever vendas futuras e preços de ações. No entanto, esses métodos não determinam a decisão a ser tomada quando é atingido um determinado preço de ação e é aí que entra a Aprendizagem por Reforço. Um agente de Aprendizagem por Reforço pode decidir sobre tal tarefa, ou seja, se deve manter, comprar ou vender a ação. O modelo de *Reinforcement Learning* é avaliado usando padrões de referência de mercado, para que o seu funcionamento ideal seja garantido.

### Recomendação de notícias

As preferências de um utilizador podem mudar com frequência, portanto recomendar notícias aos utilizadores com base em avaliações pode rapidamente tornar-se obsoleto. Com *Reinforcement Learning*, o sistema pode rastrear os comportamentos do leitor e recomendar-lhe as notícias que melhor satisfazem o

seu perfil.

A construção de tal sistema envolve a obtenção das características da notícia, das características do leitor e das características do contexto. As características da notícia incluem o conteúdo, o título, a editora, entre outras. As características do leitor referem-se à forma como este interage com o conteúdo, ou seja, o número de cliques e partilhas que este faz. Já as características do contexto referem-se a aspetos como a atualidade das notícias.

**Manipulação robótica**

O uso de Aprendizagem por Reforço pode treinar robôs cuja função é agarrar vários objetos (mesmo aqueles que não foram vistos durante a fase de treino). Esta aplicação pode ser vantajosa na construção de produtos numa linha de montagem de uma fábrica, por exemplo.

Isto é alcançado combinando otimização distribuída em larga escala e uma variante de *Q-Learning* designada por *QT-Opt*. O suporte prestado por *QT-Opt* para espaços de ação contínua torna-o adequado para este tipo de problemas. Primeiro, o modelo é treinado *offline* e, só depois, implementado e ajustado ao robô real.

## 3 *Particle Swarm Optimization*

### 3.1 Descrição característica

*Particle Swarm Optimization* (*PSO*) foi inicialmente introduzido por Kennedy e Eberhart em 1995 e é um dos paradigmas de inteligência de grupo mais conhecidos. Este paradigma usa um mecanismo simples que imita o comportamento do grupo em bandos de pássaros e cardumes de peixes para orientar cada elemento em busca de soluções globalmente ideais (neste caso, maximizar a disponibilidade de alimento e minimizar o risco de existência de predadores).

Este algoritmo, em vez de usar operadores evolutivos para manipular os indivíduos como noutros algoritmos evolutivos, cada indivíduo é visto como uma partícula que possui uma determinada velocidade e posição iniciais. O percurso que cada uma delas toma no espaço de pesquisa é atualizado não só de acordo com o seu valor para a melhor posição encontrada, mas também de acordo com os valores para as melhores posições encontradas pelos seus companheiros. Desta forma, espera-se que o grupo se mova em direção às melhores soluções e, analogamente, espera-se que o algoritmo atinja o seu objetivo.

### 3.2 Modo como exibe a capacidade de aprendizagem

Em *Particle Swarm Optimization*, cada partícula está associada a dois vetores, isto é, o vetor de velocidade $V_i = [v_i^1, v_i^2, ..., v_i^D]$ e o vetor posição $X_i =$

$[x_i^1, x_i^2, ..., x_i^D]$, onde $D$ representa as dimensões do espaço de pesquisa. A velocidade e a posição de cada partícula são inicializadas por vetores aleatórios dentro dos intervalos correspondentes. Durante o processo, a velocidade e a posição da partícula $i$ na dimensão $d$ são atualizadas segundo

$$v_i^d = v_i^d + c_1 rand_1^d(pBest_i^d - x_i^d) + c_2 rand_2^d(nBest_g^d - x_i^d) \quad (1)$$

$$x_i^d = x_i^d + v_i^d \quad (2)$$

onde $c_1$ e $c_2$ são os coeficientes de aceleração e $rand_1^d$ e $rand_2^d$ são dois números aleatórios uniformemente distribuídos independentemente gerados dentro do intervalo [0, 1] para a dimensão $d$. Em (1), $pBest_i$ é a melhor posição encontrada pela partícula até ao momento e $nBest$ é a melhor posição encontrada na vizinhança. Posto isto, a primeira equação vai calcular e atualizar a velocidade da partícula, enquanto a segunda vai adicionar o valor da velocidade calculada pela primeira equação à posição atual da partícula.

A equação (1) consiste em três partes. A primeira parte indica-nos que a velocidade não pode ser alterada abruptamente e, consequentemente, é alterada de acordo com a velocidade anterior da partícula. A segunda parte representa o pensamento individual da partícula, na medida em que esta atualiza a sua posição com o melhor valor encontrado pela própria. A terceira parte representa a colaboração entre as partículas, uma vez que a posição é atualizada de acordo com a melhor posição encontrada pelos pares da partícula.

Nesta equação, se a soma das três partes exceder um valor constante especificado pelo utilizador, então a velocidade nessa dimensão é atribuída a $V_{max}$, ou seja, as velocidades das partículas em cada dimensão estão limitadas por uma velocidade máxima $V_{max}$. Contudo, um valor muito alto para $V_{max}$ pode resultar na deslocação das partículas para lá das boas áreas de solução, enquanto um valor muito baixo pode resultar no aprisionamento das partículas em mínimos locais, portanto torna-se importante saber escolher um valor ideal para $V_{max}$.

O algoritmo *PSO* é conceptualmente simples, fácil de implementar, computacionalmente eficiente e pode enumerar-se nos seguintes passos:

1. Inicializar uma população de partículas com posições e velocidades aleatórias nas $D$ dimensões do espaço de pesquisa;

2. Para cada partícula, avaliar a função de otimização desejada nas $D$ dimensões;

3. Comparar a avaliação da partícula com o seu $pBest$. Se o seu valor atual for melhor que $pBest$, então igualar o último ao seu valor atual e $x_i$ à sua localização atual;

4. Identificar a partícula da vizinhança com os melhores valores até ao momento e atribuir o seu índice à variável $g$;

5. Mudar a velocidade e a posição da partícula de acordo com as equações (1) e (2);

6. Voltar ao passo 2. até que o critério de paragem seja atingido.

### 3.3 Ferramentas de desenvolvimento existentes

*PSO* é um algoritmo aplicado, sobretudo, a problemas de otimização com restrições, a problemas de otimização multiobjetivo e a problemas de maximização e minimização.

Para o uso e pesquisa deste algoritmo são usadas ferramentas, tais como:

- ***MATLAB - Matrix Laboratory***: fornece uma *interface* e plataforma poderosas para executar extensas pesquisas e experiências com grandes quantidades de amostras. Este inclui processamento paralelo e muitos recursos integrados fáceis de usar. Contém uma ampla gama de utilizadores, desde investigadores a estudantes, sendo das ferramenta mais usadas para este tipo de estudo;

- ***PySwarms***: é um *kit* de ferramentas de pesquisa extensível para *PSO* em *Python*, fácil de usar e possível de se implementar em poucas linhas, o que é uma característica importante, principalmente para utilizadores que desejam um processo de otimização rápido e simples;

- ***SwarmViz***: é uma ferramenta de visualização *open-source* para *PSO*, que permite aos investigadores monitorizarem o progresso de um problema de otimização específico e ajustarem os parâmetros de *PSO* relevantes.

### 3.4 Soluções existentes no mercado

A diversidade de problemas solucionáveis por *PSO* estende-se por várias áreas desde a eletrónica à medicina.

Posto isto, destacam-se as aplicações que se apresentam de seguida.

#### Redes Neuronais

O uso de Redes Neuronais Artificiais (RNAs) tem vindo a ganhar cada vez mais popularidade no que toca ao diagnóstico de doenças. Estas, treinadas usando *PSO*, são utilizadas para uma ampla variedade de problemas de classificação de doenças, onde os resultados mostram também ser um sucesso em termos de

otimização de pesos deste tipo de rede.

***Network Communication***

A aplicação de *PSO* tem-se mostrado bastante eficiente na abordagem de questões de qualidade do serviço no processo de *design*. Os principais objetivos de otimização são o custo de *layout* da rede e o atraso médio de um pacote na rede. Neste procedimento é aplicada uma instância multiobjetivo do algoritmo *PSO* híbrido.

**Sistemas Elétricos de Potência**

Os Sistemas Elétricos de Potência são grandes sistemas de energia que englobam geração, transmissão e distribuição de energia elétrica. A motivação inicial para o uso de *PSO* neste campo de pesquisa deve-se principalmente à sua dificuldade, nomeadamente devido à complexidade de cálculos de fluxo de potência. As subáreas deste tipo de sistemas que exploram os algoritmos de PSO são o controlo de energia reativa, a redução de perdas de energia, o fluxo de potência ideal e problemas no âmbito de *Economic Dispatch*.

# 4 *Support Vector Machines*

## 4.1 Descrição característica

*Support Vector Machines* (*SVM*) é um método de Aprendizagem com Supervisão, que usa algoritmos de classificação e é utilizado para problemas de classificação de dados na área de *Machine Learning*. As *SVM* baseiam-se numa teoria de aprendizagem estatística desenvolvida por Vladimir Vapnik e Alexander Lerner a partir de um artigo que estes elaboraram, onde surgiu o algoritmo *Generalized Portrait*, um algoritmo de reconhecimento de distinção, que serve de base para algoritmos atuais usados pelas *SVM*. O objetivo final de uma *Support Vector Machine* é separar, utilizando um hiperplano, um *dataset* em dois conjuntos de dados, de acordo com a sua classificação. De modo a encontrar o hiperplano mais adequado ao *dataset* é necessário escolher o algoritmo que ofereça os melhores resultados. Os algoritmos apresentados de seguida têm isto como objetivo.

Considere-se o seguinte *dataset* onde foi medida a massa de ratos e os resultados foram sobrepostos num espaço uni-dimensional de acordo com os resultados. O objetivo é classificar os ratos como sendo obesos (vermelho) ou não obesos (verde).

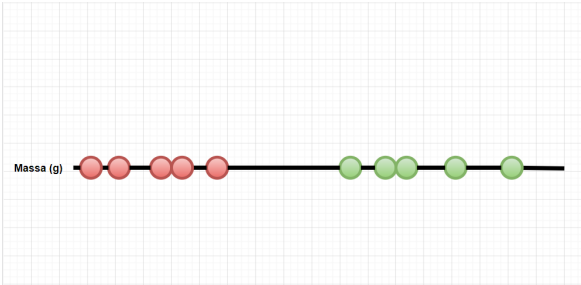


**Figura 1.** Demonstração do *dataset*

### *Maximum Margin Classifier*

Este algoritmo calcula o hiperplano/*threshold* ótimo que irá permitir a margem máxima obtida pelos elementos de fronteira de cada um dos conjuntos. O problema deste algoritmo é que é hipersensível no que toca aos dados de treino, logo, apesar de ter um nível de classificações erradas (*bias*) baixo, vai ter uma variância alta. Estas características tornam este algoritmo pouco flexível e muito limitado e, portanto, outras alternativas nasceram.

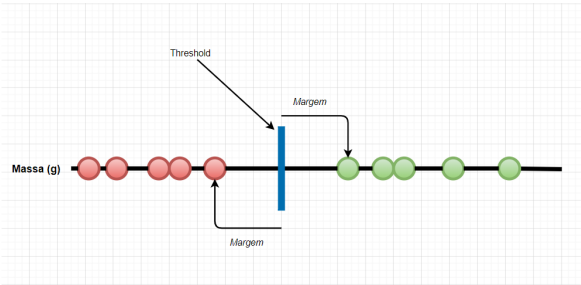


**Figura 2.** Margem máxima

### *Support Vector Classifier*

A diferença entre este algoritmo e o anterior é que este é menos sensível aos dados de treino, a margem permite classificações erradas o que resulta em melhores classificações de dados novos (variância).

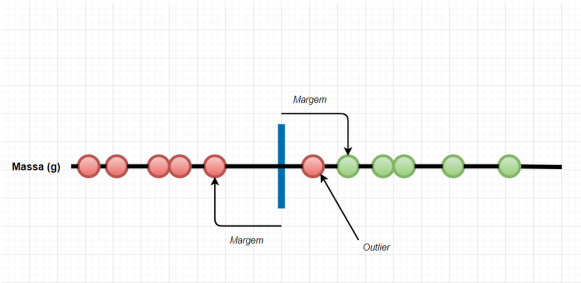


**Figura 3.** Classificação errada em *Support Vector Classifier*

A permissão de classificações erradas cria outros problemas: como é que se determina o hiperplano com a melhor margem? O que significa ser a melhor margem? Começando por responder primeiro à última pergunta, a melhor margem é aquela que, ao receber novos dados, vai permitir o número mínimo de classificações erradas, tal que o número de classificações corretas seja maximizado. Respondendo agora à primeira pergunta, o hiperplano com a melhor margem para o *dataset* em questão, vai ser calculado recorrendo à validação cruzada, ou seja, vai ser escolhida uma margem para ser testada e vai ser analisado quão bem esta se deu neste *dataset*. O mesmo processo vai ser repetido para todas as margens possíveis e, no fim, comparando o desempenho de cada uma das margens, escolhe-se a melhor.

O problema destes classificadores é que são lineares, o que significa que apenas são aplicáveis em situações em que o *dataset* é linearmente separável. No exemplo da Figura 4, foi registada a dose de um certo medicamento dada a pacientes, com o objetivo de classificar os pacientes como estando curados (verde) ou não curados (vermelho). A aplicação do *Maximum Margin Classifier* ou do *Support Vector Classifier* não seria possível neste *dataset*, pois iria causar uma enorme quantidade de classificações erradas.

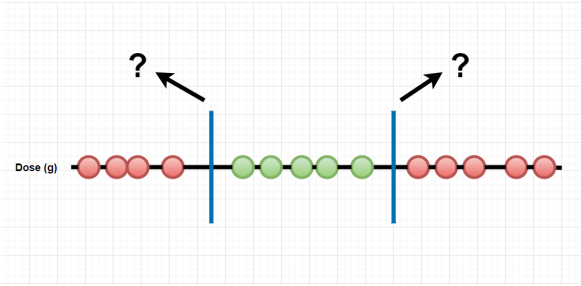


**Figura 4.** Problema de linearidade

***Support Vector Machines***

A introdução de *Support Vector Machines* (*SVM*), veio combater o problema da linearidade dos algoritmos anteriores. As principais ideias por trás das *SVM* são:

1. Começar com os dados numa dimensão relativamente pequena;
2. Mover os dados para uma dimensão maior;
3. Encontrar um *Support Vector Classifier* que separe os dados na dimensão superior em dois grupos.

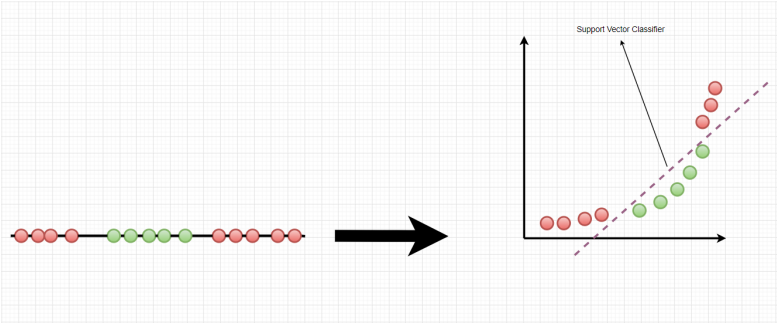


**Figura 5.** Movimentação de dados para uma dimensão superior

Na Figura 5 é possível ver que, quando os dados foram movidos para uma dimensão superior, foi possível separá-los em dois grupos claramente visíveis. Posto isto, como é decidida a transformação dos dados? Como é decidida a dimensão para a qual se vai mover os dados? Respondendo à primeira pergunta, para mover os dados de uma dimensão para outra superior, recorre-se às funções *kernel*. As funções *kernel* computam as relações entre cada par de observações na dimensão inferior como se os dados estivessem na dimensão superior. Estes cálculos são usados para encontrar o *Support Vector Classifier* na dimensão superior. A dimensão para a qual se vai fazer a transformação de dados é escolhida recorrendo à validação cruzada. Como foi dito, as funções *kernel* apenas computam as relações entre os pares de observações em dimensões superiores, não efetuando a transformação em si. A isto chama-se ao truque do *kernel*. Assim, evita-se a computação que vem com a transformação dos dados em si.

## 4.2 Modo como exibe a capacidade de aprendizagem

Visto que as *Support Vector Machines* são um Sistema de Aprendizagem com Supervisão de *Machine Learning*, existe uma função que mapeia um *input* num

*output* e está associada ao cálculo do hiperplano de separação. Esta função utiliza os pontos de fronteira mais adequados de cada um dos grupos e calcula o hiperplano ótimo. Quando introduzidos novos dados a sua classificação irá depender de que lado irão ficar relativamente ao novo hiperplano calculado. Este hiperplano apenas será alterado caso os novos dados contenham novos pontos de fronteira que permitem calcular um melhor hiperplano. Note-se que, quando nos referimos a novos dados, não se tratam dos dados do *training set*, são novos dados introduzidos que não estavam presentes quando o hiperplano original foi calculado.

### 4.3 Ferramentas de desenvolvimento existentes

As ferramentas listadas em baixo vão desde bibliotecas C++ a ambientes gráficos de arrastar e soltar. Algumas são *Support Vector Machines* puras e outras são plataformas de *data mining* que incluem *SVM*, tais como:

- ***KNIME***: é uma interface gráfica baseada em ferramentas de mineração de dados de ampla funcionalidade com suporte *SVM* particularmente fácil de usar;

- ***LIBSVM***: é uma ferramenta *open source* desenvolvida em C++ pela Universidade Nacional de Taiwan para a utilização de *SVM* em contextos de classificação linear/não linear e regressão estatística. Além disso, suporta a classificação em ambientes com mais do que duas classes de dados;

- ***mySVM***: foi escrito em C++ e inclui classificação e regressão *SVM*. Está disponível como código-fonte e binário do *Windows*. As funções *kernel* incluem função de base linear, polinomial, radial, neural e ANOVA;

- ***Orange***: é outra interface gráfica baseada em ferramentas de mineração de dados que inclui um *SVM learner*;

- ***RapidMiner***: é um conjunto de interfaces gráficas baseadas em ferramentas de mineração de dados e, possivelmente, a plataforma mais usada do seu tipo. Inclui várias implementações *SVM* (p. e. LIBSVM);

- ***SVM Light***: é um pacote de classificação e regressão amplamente usado e é distribuído como em C++. É bem conhecido pela sua velocidade de execução e pela eficiente implementação do método do *leave-one-out (LOO) cross-validation*;

- ***Weka***: uma das coleções mais conhecidas de ferramentas de *data mining* que contém uma implementação *SVM*;

- ***Scikit-learn***: é uma biblioteca de *Machine Learning open-source* para a linguagem de programação *Python*. Inclui vários algoritmos de classificação,

regressão e *clustering* incluindo *SVM*, *random forests*, *gradient boosting*, *k-means* e *Density-Based Spatial Clustering of Applications with Noise (DBSCAN)* e é desenhada para interagir com as bibliotecas *NumPy e SciPy* de *Python*.

## 4.4 Soluções existentes no mercado

Como pudemos ver, *Support Vector Machines* está sustentado em algoritmos de Aprendizagem com Supervisão. O objetivo em usar *SVM* é classificar corretamente dados inéditos. Existe um vasto número de aplicações em diversas áreas usando *SVM*. Algumas das aplicações mais comuns de *SVM* na vida real apresentam-se de seguida.

### Reconhecimento facial

É o algoritmo que, a partir de uma imagem ou um vídeo de uma cena, identifica ou verifica uma ou mais pessoas usando um conjunto de dados de faces, criando um quadrado em volta das partes classificadas como cara. Essa identificação pode ser dificultada por diversos fatores, tais como diferenças de luminosidade, óculos, maquilhagem, posicionamento, barba, bigode, cabelo, expressões faciais, etc.

### Classificação de imagens

O uso de *SVM* fornece uma maior precisão na classificação de imagens. Há uma precisão superior comparando com as técnicas tradicionais utilizadas. Traços de vários segmentos $x$ por $x$ *pixels* são extraídos da imagem e classificados como sendo cara ou não.

### Categorização de texto

O algoritmo usa dados de treino para classificar documentos em diferentes categorias, como artigos de notícias, *e-mails* e páginas *web*. Exemplos:

- Classificação de artigos de notícias em "negócios"e "filmes";
- Classificação de páginas *web* em páginas iniciais pessoais e outras.

Para cada documento, calcula uma pontuação e compara-a com um valor limite predefinido. Quando a pontuação de um documento ultrapassa o valor limite, o documento é classificado numa categoria definitiva. Se não ultrapassar o valor limite, considera-o como um documento geral. Classifica novas instâncias calculando a pontuação de cada documento e comparando-o com o limite aprendido.

### Bioinformática

Inclui a classificação de proteínas e cancro. Usa *SVM* para identificar e classificar genes, pacientes baseando-se nos genes e outros problemas biológicos. A deteção remota de homologia de proteínas é um problema na biologia computacional. Algoritmos de Aprendizagem com Supervisão baseados em *SVM* são um dos métodos mais eficazes para deteção remota de homologia. O desempenho destes métodos depende de como as sequências de proteínas são modeladas. É o método usado para calcular a função *kernel* entre eles.

**Reconhecimento de caligrafia**

Uso de *SVM* para reconhecer os caracteres mais frequentemente usados. Também se pode usar *SVM* para reconhecer caracteres escritos à mão que são usados para entrada de dados e validação de assinaturas em documentos.

## 5 Conclusão

Com a realização deste artigo averiguamos que, cada vez mais, cada um destes Sistemas de Aprendizagem tem vindo a ser usado para solucionar problemas do dia-a-dia nas mais diversas áreas. Todos eles se revelam uma mais valia na área da Inteligência Artificial, todavia não deixam de ter os seus prós e contras.

Em primeiro lugar, *Reinforcement Learning* assemelha-se bastante à forma de aprendizagem dos humanos, na medida em que, se o agente obtiver *feedback* positivo, continua a executar a ação e, se receber *feedback* negativo, não a volta a repetir. Porém, trata-se de um Sistema de Aprendizagem limitado a um conjunto reduzido de casos e, por vezes, computacionalmente inexequível, devido ao número de estados que o ambiente pode apresentar.

Em segundo lugar, *Particle Swarm Optimization* já se aproxima do comportamento de cardumes de peixes ou bandos de pássaros. Trata-se de um algoritmo bastante simples e, consequentemente, fácil de implementar. Contudo, pode tornar-se complicado definir os parâmetros iniciais e, por conseguinte, poderá convergir prematuramente para mínimos locais.

Por último, podemos concluir que as *SVM* não só fazem previsões confiáveis, como também reduzem a quantidade de informação redundante. São também relativamente eficientes na gestão da memória em espaços de grande dimensão. No entanto, não são adequadas para grandes *datasets*.

## Referências

1. Coggan, M. (2004). Exploration and Exploitation in Reinforcement Learning. McGill University
2. Z. Zhan, J. Zhang, Y. Li and H. S. Chung, "Adaptive Particle Swarm Optimization,"in IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 39, no. 6, pp. 1362-1381, Dec. 2009, doi: 10.1109/TSMCB.2009.2015956.

3. Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization,"Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Washington, DC, USA, 1999, pp. 1945-1950 Vol. 3, doi: 10.1109/CEC.1999.785511.
4. Bruno Seixas Gomes de Almeida and Victor Coppo Leite (December 3rd 2019). Particle Swarm Optimization: A Powerful Technique for Solving Engineering Problems, Swarm Intelligence - Recent Advances, New Perspectives and Applications, Javier Del Ser, Esther Villar and Eneko Osaba, IntechOpen, DOI: 10.5772/intechopen.89633.
5. Yuhui Shi, "Particle Swarm Optimization", IEEE Neural Networks Society, February 2004.
6. MathWorks, mathworks.com
7. G. Jornod, E. Di Mario, I. Navarro and A. Martinoli, "SwarmViz: An open-source visualization tool for Particle Swarm Optimization,"2015 IEEE Congress on Evolutionary Computation (CEC), Sendai, 2015, pp. 179-186, doi: 10.1109/CEC.2015.7256890
8. PySwarms, pyswarms.readthedocs.io
9. Swarm and Evolutionary Computation, Volume 49, September 2019, Pages 62-74 Meissner, M., Schmuker, M. Schneider, G. Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural network training. BMC Bioinformatics 7, 125 (2006), doi: 10.1186/1471-2105-7-125
10. Papagianni, Chrysa Papadopoulos, Kostas Pappas, Christos Tselikas, Nikolaos Kaklamani, Dimitra Venieris, Iakovos. (2008). Communication network design using Particle Swarm Optimization. 915-920.
11. M. R. AlRashidi and M. E. El-Hawary, "A Survey of Particle Swarm Optimization Applications in Electric Power Systems,"in IEEE Transactions on Evolutionary Computation, vol. 13, no. 4, pp. 913-918, Aug. 2009, doi: 10.1109/TEVC.2006.880326
12. Top 5 tools for reinforcement learning, hub.packtpub.com/tools-for-reinforcement-learning/
13. OpenAI Gym, gym.openai.com/
14. TensorFlow, www.tensorflow.org/
15. Keras, keras.io/
16. DeepMind Lab, deepmind.com/research/publications/deepmind-lab
17. 7 Free SVM Tools, butleranalytics.com
18. KNIME, www.knime.com
19. LIBSVM, www.csie.ntu.edu.tw
20. mySVM, www-ai.cs.tu-dortmund.de/SOFTWARE/MYSVM/index.html
21. Orange, orange.biolab.si
22. RapidMiner, rapidminer.com
23. SVM Light, svmlight.joachims.org
24. Weka, www.cs.waikato.ac.nz/ml/weka
25. textitdata mining Scikit-learn, scikit-learn.org/stable
26. Real-Life Applications of SVM (Support Vector Machines), data-flair.training/blogs/applications-of-svm/
27. Support Vector Machines, Clearly Explained!!! www.youtube.com/watch?v=efR1C6CvhmE
28. MIT 6.S091: Introduction to Deep Reinforcement Learning (Deep RL) www.youtube.com/watch?v=zR11FLZ-O9Mt=2150s