1. INTRODUÇÃO À APRENDIZAGEM COMPUTACIONAL

(documento de trabalho complementar aos slides do Cap. 1)

Aprendizagem Computacional é a tradução mais adequada para o termo Inglês *Machine Learning*, embora também se use a tradução literal Aprendizagem Máquina.

1.1 Conceitos introdutórios

Este Capítulo baseia-se nos Capítulos 1 e 2 do livro *Machine Learning* de Tom Michel. Sendo um livro já com alguns anos, parece-me ser o que introduz os conceitos fundamentais de uma forma simultaneamente com rigor formal e simplicidade.

A Aprendizagem Computacional é composta por uma grande variedade de métodos e de ferramentas computacionais, é um vasto campo científico e tecnológico. Mas todas a metodologias visam desenvolver programas computacionais (que implementam algoritmos) capazes de, postos a correr, darem resultados cada vez melhores, isto é, são capazes de melhorar o seu desempenho com o decorrer da sua própria experiência, tal como acontece com os humanos.

Por simplicidade, use-se apenas o termo aprendizagem significando aprendizagem computacional.

Os algoritmos de aprendizagem são treinados com um conjunto de dados conhecidos, chamado conjunto de treino, muitas vezes resultado de medições de variáveis físicas, financeiras, sociais, etc. E obter uma boa coleção de dados de treino para um certo problema, é crucial para o sucesso do algoritmo a treinar. Este é o primeiro problema que tem que ser bem resolvido. Pensemos por exemplo num algoritmo para detetar transações fraudulentas com cartões de crédito; o algoritmo tem que nos dizer se uma certa transação é fraudulenta ou não; assim podemos dizer que implementa uma função booleana (dois valores, 0 ou 1). E essa função tem um certo número de parâmetros que têm de ser calculados. O treino é feito com um conjunto de transações umas fraudulentas, outras legítimas. Quanto maior for o número de dados de treino, melhor será o desempenho do algoritmo.

O conjunto de dados de treino tem informação embebida, isto é, informação que está implícita nos dados, mas nem sempre é visível diretamente. Para extrair essa informação, geralmente extraem-se propriedades dos dados, chamadas características (*features*); essas características, extraídas do conjunto de dados e treino, são usadas pelo algoritmo de treino da função booleana para ajustar os parâmetros desta.

Quando aparece uma nova transação o algoritmo classifica-a como legítima ou fraudulenta. Se dá uma resposta errada (o que só se sabe depois da transação feita), esse erro é usado para alterar os parâmetros da função de decisão de modo a diminuir o número de erros ao longo do tempo, melhorando assim o desempenho do algoritmo, ou seja a sua performance. A aprendizagem inicial é feita com o conjunto de dados de treino, mas depois disso a aprendizagem continua à medida que novos dados vão surgindo.

Podemos então adotar a definição de aprendizagem de Tom Mitchell (p. 2), a mais sucinta e rigorosa que conheço:

Aprendizagem: Diz-se que um programa de computador aprende com a experiência *E* em relação a alguma classe de tarefas *T* e uma medida de performance *P*, se a sua performance nas tarefas em *T*, medida por *P*, melhora com a experiência E.

Esta definição muito geral pode-se aplicar a qualquer tipo de problemas. Por exemplo:

Reconhecimento de palavras escritas manualmente:

Tarefa T: reconhecer e classificar palavras manuscritas em imagens.

Medida de desempenho P: percentagem de palavras classificadas corretamente.

Experiência de treino T: uma base de dados de palavras manuscritas com a sua classificação definida (i.e, a identificação das palavras).

A Aprendizagem Computacional aplica-se hoje em dia em praticamente todos os campos da atividade humana: identificação de pessoas em imagens de multidões, deteção de fraudes na banca e na finança, avaliação do risco nos seguros, tradução automática, diagnóstico e prognóstico médicos, controlo da produção industrial, etc.

A **aprendizagem de um conceito** é inferir uma função booleana usando um conjunto de exemplos de treino. Cada exemplo contém as entradas e a saída da função a inferir. Através da aprendizagem, encontram-se os parâmetros da função booleana. Por agora consideramos uma função boleana binária. No entanto em capítulos posteriores veremos que as funções possíveis são muito mais gerais.

Num dado problema, chamam-se **instâncias** a um conjunto X de itens nos quais o conceito é definido. Seja X o conjunto de todos os dias possíveis. Cada dia (instância), do ponto de vista do tempo, tem um conjunto de atributos tais como: aspeto do céu, temperatura, humidade, vento, chuva.

O conceito alvo *(target*) a aprender pode ser por exemplo "um bom dia para jogar ténis"[1].

Compilando os exemplos do conjunto de dados de treino, correndo o algoritmo, aprende-se o conceito alvo. Para que isso seja feito com sucesso há alguns requisitos essenciais a que os dados de treino devem obedecer. Eles devem ser representativos da todas as instâncias possíveis, e por isso devem conter exemplos positivos (bons dias) e negativos (maus dias).

Mas o conceito alvo só pode ser conhecido com precisão se o conjunto de treino contiver todas as instâncias possíveis no contexto do problema. Ora isto é impossível porque, por um lado, seria muito grande, por outro lado, e principalmente, porque as instâncias futuras não são conhecidas no presente.

Consequentemente o melhor que podemos obter a partir do conjunto de treino é uma hipótese, ou uma estimativa do conceito alvo.

E assim surge a **hipótese da aprendizagem indutiva** (Mitchell, p. 23): qualquer hipótese encontrada, depois do treino, que aproxime bem o conceito alvo num número suficientemente grande de exemplos de treino, também aproximará bem a função alvo noutros exemplos não observados. Isto é, se o treino for bem feito, o algoritmo tem capacidade de generalização.

Esta hipótese é a base de todos os algoritmos de aprendizagem computacional.

[1] O exemplo de jogar ténis de Tom Mitchell é muito usado por outros autores, nem sempre citando a fonte; há até autores que, usando os mesmos dados numéricos do exemplo de Mitchell, falam em badmington em vez de ténis.

Existem muitas realidades, muitos tipos de conceitos, muitas funções de desempenho, muitas funções alvo (não só booleanas), e ao longo dos anos desenvolveram-se por isso muitas metodologias e muitas técnicas, e continuam a desenvolver-se.

1.2. O processo de aprendizagem computacional.

Como vimos no parágrafo anterior a aprendizagem computacional passa por várias etapas sintetizadas a seguir.

1.2.1 Recolha dos dados brutos (*raw data*)

A obtenção de um conjunto de dados de treino é a primeira operação a fazer. Normalmente deve ter muitos exemplos e todos os tipos possíveis no problema em estudo. Além disso é necessário também recolher um outro conjunto de dados que não será usado para treino mas sim para teste da função alvo, depois de treinado o algoritmo. Isto é verdade para os diversos tipos de problemas (predição, classificação, decisão, etc.). Estes dados recolhidos inicialmente são dados brutos, tal como foram gerados.

A quantidade e qualidade dos dados brutos é determinante do resultado final. Eles devem espalhar-se por todo o domínio do problema em estudo, i.e., devem ser suficientemente representativos de todos os dados possíveis.

1.2.2 Pré-processamento dos dados brutos

O projetista não tem controlo total sobre a aquisição dos dados. Muitas vezes não tem mesmo qualquer controlo, eles são gerados pela realidade. Por isso, muito frequentemente, há problemas nos dados que têm que ser corrigidos de algum modo, para que o algoritmo se possa aplicar. Há dados anormais que se diferenciam drasticamente de todos os outros (por exemplo muito maiores ou muito menores do que os restantes), e por isso se chamam ***outliers*** (fora de comum), e devem ser pré-processados, normalmente eliminados.

Os dados experimentais, provindos por exemplo de sensores, são contaminados por ruídos e interferências diversas, e por isso **a filtragem** de dados é uma operação frequente.

Quando os dados têm diversas dimensões, são vetores, pode haver diferentes escalas nas diferentes dimensões, o que pode provocar problemas no processamento matemático, em que os dados grandes dominam as operações matemáticas, mas os dados pequenos podem ser tão importantes para a saída da função alvo como os grandes. Por isso é usual reduzir as diversas dimensões à mesma escala, isto é, **normalizar os dados,** para obter um problema matematicamente bem condicionado.

Quando se trata de problemas de classificação convém que, sempre que possível, haja no conjunto de treino o mesmo número de exemplos de cada classe. Nos dados brutos isso acontece raramente. Por isso é necessário **balancear as classes.**

1.2.3 Extração de características (*features*)

Normalmente os algoritmos de aprendizagem não processam os dados brutos, processam sim certas características que são deles extraídas. As características devem ser extraídas dos dados porque não são diretamente observáveis, mas estão embebidas nos próprios dados. Há muitos tipos de características, conforme o problema e a natureza dos dados. Podem ser definidas no domínio do tempo (por exemplo momentos estatísticos), no domínio da frequência (parâmeros do espectro de

potências, coeficientes da transformada de Fourier, etc.), ou no domínio híbrido tempo-frequência (ex. coeficientes de onduletas (*wavelets*), ou de outra natureza inerente ao problema em estudo.

1.2.4 Redução da dimensão

Se o número de características é grande (por exemplo na ordem de várias dezenas ou centenas, ou mesmo milhares), se os recursos computacionais não são suficientes, pode acontecer que o custo computacional seja exorbitante, e por isso é aconselhável a redução do número de características. Existem diversas técnicas para isso que não abordaremos aqui. Em muitos problemas um reduzido número de características é suficiente para que se obtenha uma boa performance.

Depois de se aplicarem estas etapas (e outras eventualmente) no pré-processamento dos dados, obtém-se um conjunto de **dados estruturados** com que se fará o treino do algoritmo. Nem sempre é um processo linear, sendo por vezes necessário refazer todo pré-processamento, e mesmo nova recolha de dados brutos, porque a performance final não foi satisfatória.

1.2.5 Por o bom modelo em produção no problema específico

Quando se obtém um algoritmo (ou modelo) com bom desempenho, põe-se "em produção", trabalhando com novos dados, nunca vistos pelo modelo.

O seu desempenho é medido constantemente, e se se degradar com o tempo (o que acontece em situações dinâmicas), é necessário retreinar o modelo, o que pode obrigar a voltar ao princípio.

1.2.6. Retreinar para mudanças dinâmicas ou derivas nos conceitos (*concept drift*)

Eventualmente o retreino pode ser uma tarefa permanente, e todo o processo se deve repetir frequentemente para captar as mudanças na dinâmica do processo que gera os dados e a deriva nos conceitos (que muitas vezes é provocada por alterações dinâmicas). Isto é, a realidade com que se trabalha está sempre a variar, mais ou menos lentamente, e a aprendizagem deve seguir essas mudanças.

A sequência de operações (por vezes chamado *pipeline*) do desenvolvimento de um modelo de aprendizagem computacional pode ser ilustrada pela Figura 1.1.

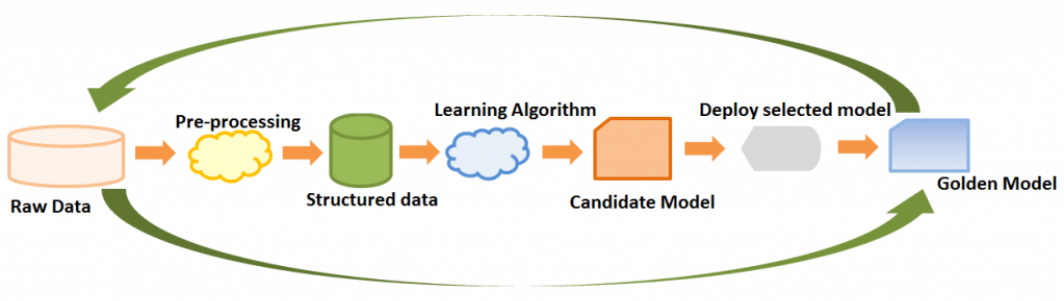


Figura 1.1. O processo de desenvolvimento de modelos de aprendizagem computacional

(https://elearningindustry.com/machine-learning-process-and-scenarios , 6 setembro 2023)

1.3 *Hard Computing e Soft Computing*

O termo *soft computing*, cunhado por Lofti Zadeh[2], é frequentemente utilizado em aprendizagem computacional. Vejamos o que significa.

Suponhamos que vamos desenvolver um piloto automático de um carro autónomo. Poderíamos encarar a possibilidade de desenvolver um modelo matemático (baseado nas leis da mecânica, dinâmica de fluidos, etc.) do carro e da estrada, resultando num elevado número de equações matemáticas de grande complexidade e que, por mais complexas que sejam, seriam sempre uma aproximação da realidade. O piloto automático usaria essas equações para conduzir o carro. Esta abordagem chama-se frequentemente de *hard computing* pelo facto de ser computacionalmente muito pesada devido à complexidade matemática.

Mas nós, os humanos, quando aprendemos a conduzir um carro usamos essa abordagem ? Claro que não, não usamos *hard computing*, não resolvemos permanentemente equações às derivadas parciais …. Simplesmente treinamos conduzir, repetidamente, em diversas circunstâncias, e depois o nosso cérebro é capaz de generalizar, de modo que conduziremos mesmo em circunstâncias que não treinámos antes. Usamos uma abordagem de *soft computing*, explorando fundamentalmente a experimentação empírica.

As metodologias de *soft computing* visam isso mesmo: construir modelos de aprendizagem com pouco peso computacional e que sejam treináveis, num processo do tipo da Figura 1.1. E os carros autónomos usam intensivamente essa abordagem; os seus pilotos automáticos usam intensamente *soft computing*. As metodologias associadas toleram a imprecisão, a incerteza, a verdade parcial, a aproximação, e além disso permitem obter tratabilidade , robustez, e soluções computacionais de baixo custo [Zadeh].

Como escreve Lofi Zadeh, “The role model for soft computing is the human mind “[3]

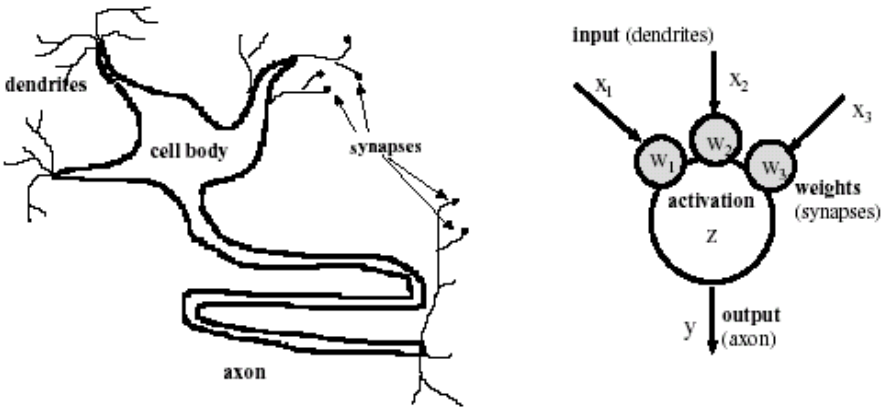
As metodologias principais de *soft computing* baseiam-se em redes neuronais, em sistems difusos, e suas combinações.

1.3. Neurónios biológicos e redes neuronais artificiais.

Os neurónios biológicos, como os nossos, são estruturas básicas de processamento da informação. A Figura 1. 2 ilustra a sua constituição.

[2] S*oft computing and fuzzy logic*, L. A. Zadeh, in *IEEE Software*, vol. 11, no. 6, pp. 48-56, Nov. 1994, doi: 10.1109/52.329401.

[3] *In* A Definition of Soft Computing- adapted from L.A. Zadeh http://www.soft-computing.de/def.html



(a) (b)

Figura 1.2. Um neurónio biológico (a) e o seu modelo matemático (b)

(de Brause, R., Medical Analysis and Diagnosis by Neural Networks, 2001)

O processador do neurónio é o corpo celular (*cell body*). As dendrites são terminais de ligação a outros neurónios e transportam impulsos elétricos (da ordem dos microvolt) desses outros neurónios. Quando a soma dos impulsos elétricos das dendrites ultrapassa um certo patamar, o corpo celular é ativado e dispara um impulso elétrico que é transmitido pelo axónio a outros neurónios. Os pontos de ligação, quer das dendrites quer dos axónios, chamam-se sinapses e podem ter uma intensidade de ligação variável (como se fossem pesadas).

A parte (b) da figura representa um modelo matemático do neurónio biológico, e por isso se chama neurónio artificial. As entradas (inputs) transmitem sinais que são multiplicados por um peso w e somados, passando a sua soma por uma função z, chamada função de ativação, que fornece o sinal de saída y. Ainda hoje se usa esta terminologia básica (entradas, pesos, função de ativação, saída, etc.). Depois usando diversos neurónios ligados em rede, em combinações série-paralelo, obtêm-se as redes neuronais artificiais que estudaremos nos Capítulos 4 e 5.

1.4. O cérebro difuso e os sistemas difusos

O nosso cérebro é capaz de tomar decisões com base em informação imprecisa. Não funciona sempre como uma máquina de calcular.

Por exemplo o nosso sistema de visão funciona de um modo difuso, em que as variáveis não são definidas com valores precisos, de acordo com a teoria de Young-Helmholtz[4] dos recetores visuais.

De acordo com esta teoria temos três tipos de recetores de luz na retina: o recetor 1 mais sensível ao magenta (entre o vermelho e o laranja), o recetor 2 ao verde, e o recetor 3 ao azul.

[4] https://en.wikipedia.org/wiki/Young-Helmholtz_theory 6 setembro, 2023

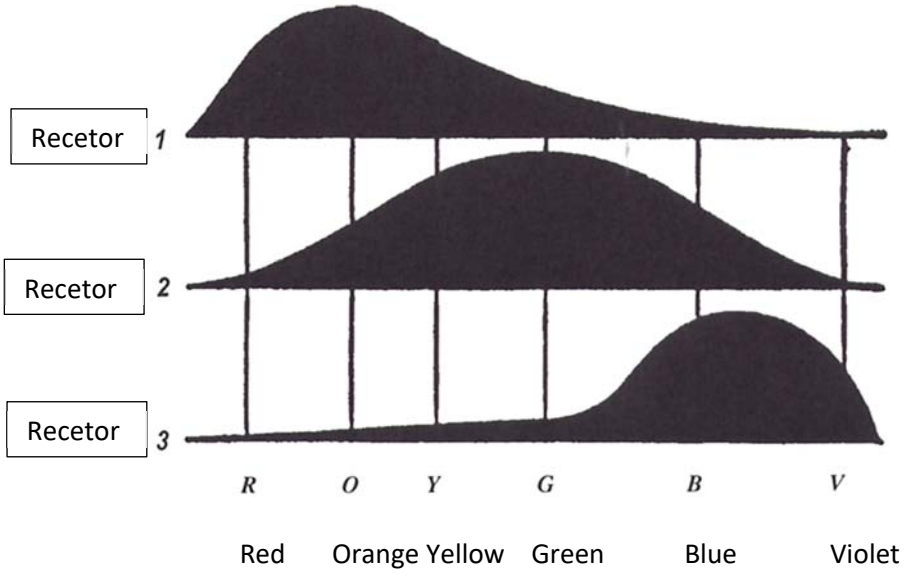


Figura 1.3. Sensibilidade dos três recetores visuais RGB segundo Young.

A sensibilidade dos recetores não varia abruptamente com o comprimento de onda. Para cada um deles vai subindo progressivamente desde zero até ao máximo, e depois decresce também progressivamente até zero. As superfícies negras da Figura ilustram isso mesmo. O máximo está próximo da cor central do recetor.

Cada recetor envia ao cérebro a sua informação (sinais elétricos de intensidade variável). Depois o cérebro combina os três sinais e sintetiza a perceção de uma cor, que nós vemos. Esta é a base da tecnologia RGB de criar cores artificiais (Red, Green, Blue).

Este funcionamento é matematicamente modelizável pelos sistemas difusos, que estudaremos no Capítulo 6 e 7.

1.5 Aprendizagem supervisionada, não-supervisionada, e por reforço.

No parágrafo 1.1 definiu-se aprendizagem de um modo muito genérico. A implementação dessa definição pode ser feita de três modos, dando origem a

- Aprendizagem supervisionada, quando o comportamento desejado do modelo (o alvo, *target,* que cumpre o papel de um supervisor) é conhecido à partida, e a aprendizagem é um algoritmo que ajusta os parâmetros do modelo de modo que esse comportamento desejado seja alcançado tanto quanto possível. O desempenho do modelo mede-se pela diferença entre o *target* e o comportamento real obtido. Para cada instância no domínio do problema conhece-se o seu alvo, isto é, para uma instância concreta o modelo deve produzir um certo alvo. As redes neuronais e os sistemas difusos enquadram-se nesta abordagem.

- Aprendizagem não supervisionada, quando à partida não se especifica um comportamento desejado, não existe qualquer *target*. É o algoritmo de aprendizagem que nos informa do que aprendeu com os dados. No fundo serve para extrair conhecimento dos dados, no interesse do próprio projetista. Os algoritmos de agrupamento (*clustering*) são o exemplo mais representativo desta família de modelos.

- Aprendizagem por reforço (*reinforcement learning*) que é uma abordagem em que o algoritmo computacional é orientado por um objetivo (*goal-directed computational approach*). O computador aprende a desempenhar uma tarefa interagindo com um ambiente dinâmico desconhecido, vai tomando decisões de modo a maximizar uma recompensa acumulada para a tarefa, sem a intervenção humana, e sem ser programado explicitamente para alcançar a tarefa. Não será objeto de estudo nesta disciplina.

1.6 Os objetivos desta disciplina de Aprendizagem Computacional

Nesta disciplina estudaremos:

Aprendizagem não-supervisionada:

Técnicas de agrupamento (*Clustering*) : dado um conjunto de dados, encontrar estruturas topográficas embebidas neles (não visíveis diretamente), com base nas semelhanças e diferenças entre os pontos dos dados, usando uma métrica para essas semelhanças.

O resultado é uma organização dos dados em grupos (*clusters*) distintos, que em muitos problemas são considerados classes no domínio do problema, e por isso pode ser considerado como um método e classificação não-supervisionado.

Aprendizagem supervisionada:

Árvores de decisão.

Redes Neuronais Rasas e Profundas.

Lógica difusa e sistemas difusos.

Sistemas Neuro-difusos.

Estas técnicas têm um vasto domínio de aplicações: ar condicionado, máquinas de lavar e secar, máquinas fotográficas e de filmar, em toda as indústrias, nos veículos autónomos, nos serviços (banca, seguros, marketing, etc.), na internet, em aplicações medicas e clínicas, etc.

1.7 Bibliografia da disciplina

Principal

Machine Learning, Tom Mitchell, McGraw-Hill, 1999

Neural Network Design, Hagan, Demuth and Beale*,* PWS Publishing*, 2nd ed. 2014,* eBook livre e gratuito em *http://hagan.okstate.edu/NNDesign.pdf*

Fuzzy Logic with Engineering Applications, Timothy Ross, 4th Ed., Wiley, 2016.

Deep Learning Toolbox Users´s Guide, The Mathworks, 2023

Fuzzy Logic Toolbox Users´s Guide, The Mathworks, 2023.

Complementar

Machine Learning, An Algorithm Perspective, Marsland, Stephen, CRC Press 2nd Ed 2014

Machine Learning, A Probabilistic Perspective, Murphy, Kevin P., MIT Press 2012.

Fundamentals of Artificial Neural Networks, Hassoun. M. H.,MIT Press, 1994.

Neural and Adaptive Systems, J.C. Príncipe, N.R. Euliano, W. C. Lefevre, Wiley, 2000

Introduction to Neuro-Fuzzy Systems, Robert Fullér, Springer Verlag 2000.

Introduction to Neural Networks, Kevin Gurney, UCL Press, 1997.

Neural Networks: A Comprehensive Foundation, Simon Haykin,Prentice Hall,1999.

Introduction to Neural Networks, Kevin Gurney, UCL Press, 1997.

Fundamentals of Neural Networks, Laurene Fausette, Prentice Hall, 1994.

Fuzzy Modelling and Control, Andrzej Piegat, Springer Verlag, 2001.

Fuzzy Engineering, Bart Kosko, Prentice Hall, 1997.

Fuzzy Sets and Fuzzy Logic, Theory and Applications, George J. Klir and Bo Yuan, Prentice Hall, 1995.

Fuzzy Systems Theory and its Applications, T. Terano, K. Asai and M. Sugeno, Academic Press, 1987.

Fuzzy Logic, H. Zimmermann, Springer Verlag, 1997.

An Introduction to Fuzzy Sets Analysis and Design, Witold Pedrycz and Fernando Gomide, MIT Press, 1998.

Heaven in a Chip, Fuzzy Visions of Society and Science in the Digital Age, Bart Kosko, Three Rivers Press, NY, 1999.

Os trabalhos práticos decorrerão em ambiente Matlab+Toolboxes (pelo menos Deep Learning Toolbox, Fuzzy Logic Toolbox, Optimization Toolbox)