

# UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

---

Meta-características para Aprendizado por Reforço

*Felipe Alves Siqueira*

---



São Carlos – SP



# Meta-características para Aprendizado por Reforço

**Felipe Alves Siqueira**

***Orientador:* Prof. Dr. André C. P. L. F. de Carvalho**

Monografia final de conclusão de curso apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como requisito parcial para obtenção do título de Bacharel em Computação.

*Área de Concentração:* Aprendizado de Máquina

**USP – São Carlos**  
**Novembro de 2020**

Siqueira, Felipe Alves

Meta-características para Aprendizado por Reforço /  
Felipe Alves Siqueira. - São Carlos - SP, 2020.

50 p.; 29,7 cm.

Orientador: André C. P. L. F. de Carvalho.

Monografia (Graduação) - Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos - SP, 2020.

1. Meta-aprendizado por reforço. 2. Meta-Aprendizado.  
3. Aprendizado por reforço. 4. Meta-características.  
5. Aprendizado de Máquina. I. Carvalho, André C. P.  
L. F. de. II. Instituto de Ciências Matemáticas e de  
Computação (ICMC/USP). III. Título.

*“ Uma caracterização apropriada dos conjuntos de dados pode elucidar a interação entre os mecanismos de aprendizado e a tarefa sob análise. ”*  
(Vilalta et al., 2004, “Using Meta-Learning to Support Data Mining”)



# RESUMO

SIQUEIRA, F. A.. **Meta-características para Aprendizado por Reforço**. 2020. 50 f. Monografia (Graduação) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Meta-aprendizado, ou “aprender a aprender”, é um campo de pesquisa muito ativo no momento. No contexto de aprendizado de máquina por reforço, geralmente usa-se redes neurais recorrentes para induzir meta-modelos que capturam padrões em características de problemas similares retirados de uma distribuição pré-determinada. Após treinadas, estas redes recorrentes são capazes de “implementar” automaticamente algoritmos de aprendizado por reforço que exploram características específicas a cada exemplo desta distribuição. Neste trabalho, é explorado uma perspectiva alternativa de meta-aprendizado por reforço, orientada ao acúmulo de meta-conhecimento e livre de redes neurais recorrentes. Foi conduzido um experimento de recomendação de algoritmos que resultou em evidências que esta perspectiva é possível e talvez seja um tema de pesquisa a ser explorado melhor futuramente.

**Palavras-chave:** Meta-aprendizado por reforço, Meta-Aprendizado, Aprendizado por reforço, Meta-características, Aprendizado de Máquina.





# ABSTRACT

SIQUEIRA, F. A.. **Meta-características para Aprendizado por Reforço**. 2020. 50 f. Monografia (Graduação) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Meta-learning, or “learning to learn”, is currently a very active research area. In the context of reinforcement learning, recurrent neural networks are usually used to create meta-models which captures patterns in characteristics of similar problems from a pre-determined distribution. After trained, those recurrent networks are capable of automatically “implement” reinforcement learning algorithms which exploit specific characteristics from every sample from the distribution. In the present work, an alternative meta-reinforcement learning perspective is explored, based on storing meta-knowledge and also recurrent networks free. It was driven an experiment which gave evidence that this adopted perspective is possible and maybe can be a research subject to be explored better in the future.

**Key-words:** Meta-reinforcement learning, Meta-Learning, Reinforcement Learning, Meta-Characteristics, Machine Learning.



## LISTA DE ILUSTRAÇÕES

- Figura 1 – Esquema de uma tarefa típica de aprendizado de máquina por reforço. 19
- Figura 2 – Exemplo em que um agente guloso recebe um retorno sub-ótimo (0) seguindo pelo caminho  $s_0 \rightarrow s_2 \rightarrow s_3$ , sendo o caminho ótimo  $s_0 \rightarrow s_1 \rightarrow s_3$  com retorno total de 10. Neste dígrafo, o agente sempre inicia no estado  $S_0$ . As ações ( $a \in \{0, 1\}$ ) e recompensas ( $r$ ) associadas a cada transição estão representadas nas arestas correspondentes. Assume-se que qualquer ação tomada no estado  $S_3$  encerra o processo. 20
- Figura 3 – Exemplo de um *Gridworld* gerado aleatoriamente. Na figura, as dimensões do meio são 10x10. O agente encontra-se em seu estado inicial (definido aleatoriamente também durante a geração do meio) e é representado na figura pelo pequeno círculo vermelho. As células pretas são bloqueadas e impedem a passagem do agente, e todas as outras células estão disponíveis. As células objetivo (recompensa positiva) estão representadas em verde e um símbolo ★, enquanto que as células armadilhas estão representadas em cor vermelha e com um símbolo ✖. . . . . 31
- Figura 4 – Impacto dos hiper-parâmetros “Probabilidade de Movimento Ruidoso” (eixo horizontal) e “Probabilidade de Desativar Busca Guiada” (eixo vertical). O agente só pode movimentar-se por *pixels* claros. As figuras no canto esquerdo-inferior possuem ambos os parâmetros iguais a zero; as figuras superiores possuem probabilidade de desativação de busca guiada no máximo valor permitido nos experimentos (0.25); e as figuras no canto direito possuem a probabilidade de movimento ruidoso configuradas no máximo permitido nos experimentos (0.50). Em (b), os hiper-parâmetros das figuras intermediárias são interpolações lineares entre os valores extremos. . . . . 32
- Figura 5 – Distribuição de (a) convergência e (b) recompensa acumulada em um único episódio dos algoritmos de aprendizado base após o treinamento. Cada coluna representa um algoritmo distinto (SARSA, Monte Carlo Control, Q-Learning e Double Q-Learning, nesta ordem da esquerda para a direita), e cada linha representa números distintos de episódios de treinamento (500, 1000 e 3000, nesta ordem de cima para baixo). . . . . 35

Figura 6 – Relação entre os meta-atributos independentes $X$ e alvo $y$ . Cada conjunto de meta-atributo alvo $y$ possui dois conjuntos de meta-atributos independentes $X$ . Na figura, $(m, n)$ indica que o conjunto de dados correspondente possui $m$ linhas e $n$ colunas. Nos experimentos foram usados $m \in \{500, 1000, 3000\}$ . Cada par $(X, y)$ é usado para induzir um meta-classificador e um meta-regressor distinto. . . . .	36
Figura 7 – <i>Feature importance</i> das meta-características de ambas as famílias, extraídas utilizando um meta-classificador nos meta-dados de 500 episódios de treinamento. . . . .	40
Figura 8 – Impacto na performance (AUC) no conjunto de teste para todos os algoritmos base sobrepostos. A linha de referência é $AUC = 0.5$ . . .	48
Figura 9 – Impacto na performance de teste (AUC) no meta-classificador treinado com meta-características ruidosas para deduzir a convergência dos algoritmos SARSA (a) e Monte Carlo Control (b). As linhas tracejadas superiores (vermelha) marcam a AUC de teste sem inserção de ruídos dos respectivos algoritmos. As linhas pontilhadas superior e inferior marcam o intervalo de confiança (desvio-padrão) da AUC de referência. . . . .	49
Figura 10 – Impacto na performance de teste (AUC) no meta-classificador treinado com meta-características ruidosas para deduzir a convergência dos algoritmos Q-Learning (a) e Double Q-Learning (b). As linhas tracejadas superiores (vermelha) marcam a AUC de teste sem inserção de ruídos dos respectivos algoritmos. As linhas pontilhadas superior e inferior marcam o intervalo de confiança (desvio-padrão) da AUC de referência. . . . .	50

---

## LISTA DE QUADROS

---

Quadro 1 – Principais propriedades de algoritmos baseados em Programação Dinâmica (PD) e amostragens Monte Carlo (MC) e <i>Bootstrapping</i> (BTSP) para estimar a função de valor $V$ . (*): assumindo infinitas repetições. . . . .	23
Quadro 2 – Exemplos de meta-características de uma série temporal. Na figura, a série temporal $S$ original é representada pelo gráfico superior. O seu componente de tendência $T$ ( <i>trend</i> ) é exibido separadamente no gráfico do meio, e o componente de ruído $R$ no gráfico inferior, onde $S = T + R$ . As meta-características foram extraídas usando a biblioteca <i>ts-Pymfe</i> (disponível em: <a href="https://github.com/FelSiq/ts-pymfe">https://github.com/FelSiq/ts-pymfe</a> ) para a linguagem de programação Python. . . . .	27
Quadro 3 – Hiper-parâmetros associados com a distribuição probabilística dos meios utilizados durante os experimentos. Valores mínimos são inclusos, diferente dos valores máximos. Todos os hiper-parâmetros foram selecionados de forma independente e uniforme em relação ao conjunto de valores válidos correspondente. . . . .	31
Quadro 4 – Hiper-parâmetros do algoritmo <i>XGBoost</i> usados na etapa de treinamento dos meta-modelos. Todos os parâmetros que não apareceram explicitamente neste quadro foram definidos como os valores padrão da biblioteca <i>xgboost</i> para a linguagem de programação Python. . . . .	36
Quadro 5 – Resultados dos experimentos para a família de meta-características “elaboradas”. A coluna da esquerda indica o número de episódios de treinamento para extrair os meta-dados. O símbolo $\sigma$ representa o desvio-padrão da quantidade que ele está associado. Em particular, $\sigma_y$ representa o desvio-padrão do meta-atributo alvo. “ACC” significa “acurácia” e “AUC” significa “Area Under the Curve” da curva “ROC” (“Receiver Operating Characteristic”). . . . .	38

Quadro 6 – Resultados dos experimentos para a família de meta-características “artificial”. A coluna da esquerda indica o número de episódios de treinamento para extrair os meta-dados. O símbolo  $\sigma$  representa o desvio-padrão da quantidade que ele está associado. Em particular,  $\sigma_y$  representa o desvio-padrão do meta-atributo alvo. “ACC” significa “acurácia” e “AUC” significa “Area Under the Curve” da curva “ROC” (“Receiver Operating Characteristic”). . . . . 39

# SUMÁRIO

---

1	INTRODUÇÃO . . . . .	15
1.1	Motivação deste trabalho . . . . .	15
1.2	Meta-aprendizado . . . . .	15
1.3	A necessidade de meta-aprendizado . . . . .	16
1.4	Objetivo deste trabalho . . . . .	16
1.5	Organização deste trabalho . . . . .	17
2	FUNDAMENTOS TEÓRICOS . . . . .	19
2.1	Aprendizado por reforço . . . . .	19
2.2	Meta-aprendizado para aprendizado por reforço . . . . .	24
2.3	Meta-aprendizado e mineração de dados . . . . .	26
2.4	Meta-características para aprendizado por reforço . . . . .	27
3	EXPERIMENTOS . . . . .	29
3.1	Proposta . . . . .	29
3.2	Regras do meio <i>Gridworld</i> . . . . .	30
3.3	Configuração e geração dos <i>Gridworlds</i> . . . . .	30
3.4	Extração de meta-características e meta-dados . . . . .	33
3.4.1	<i>Meta-atributos independentes X</i> . . . . .	33
3.4.2	<i>Meta-atributos alvo y</i> . . . . .	34
3.4.3	<i>Juntando as informações</i> . . . . .	34
3.5	Meta-modelo . . . . .	34
3.6	Resultados . . . . .	37
4	CONCLUSÃO . . . . .	41
	REFERÊNCIAS . . . . .	43
	APÊNDICE A            EFEITO DE RUÍDOS NAS META-CARACTERÍSTICAS . . . .	47





---

# INTRODUÇÃO

---

## 1.1 Motivação deste trabalho

Algoritmos de aprendizado por reforço no estado-da-arte podem apresentar performance equivalente ou além de performance humana em algumas tarefas (MNIH *et al.*, 2013; SILVER *et al.*, 2016), porém não compartilham da mesma eficiência humana de aprender com poucos dados (LAKE *et al.*, 2016); pelo contrário: enormes quantidades de dados são necessárias para induzir modelos com boa performance. Para isso, diferentes técnicas de “meta-aprendizado por reforço” foram sugeridas na literatura para reduzir o custo de treinamento com algoritmos de aprendizado por reforço (SCHMIDHUBER; ZHAO; WIERING, 1996; THRUN; PRATT, 1998; SCHWEIGHOFER; DOYA, 2003; DUAN *et al.*, 2016; WANG *et al.*, 2017; HOSPEDALES *et al.*, 2020). Levando isso em consideração, este trabalho busca contribuir investigando a viabilidade da união do aprendizado de máquina por reforço e uma perspectiva específica de meta-aprendizado baseada em acúmulo de meta-conhecimento.

## 1.2 Meta-aprendizado

Há muitas definições distintas de “meta-aprendizado” na literatura científica (LEMKE; BUDKA; GABRYS, 2015; HOSPEDALES *et al.*, 2020). No presente trabalho, é conveniente apresentar meta-aprendizado sob a definição de Vilalta *et al.* (2004) e Brazdil *et al.* (2009). Além disso, para contextualizar adequadamente este texto introdutório, é também oportuno constatar as principais características do aprendizado supervisionado tradicional de acordo com Mitchell (1997).

O objetivo de aprendizado de máquina supervisionado é aproximar uma função desconhecida que descreve uma distribuição probabilística conjunta entre os atributos independentes  $X_{\text{base}}$  e o atributo alvo  $y_{\text{base}}$ . Em outras palavras, deseja-se mapear da forma mais precisa quanto possível  $X_{\text{base}}$  em  $y_{\text{base}}$ . Esta aproximação ocorre por meio do uso de um algoritmo de aprendizado de máquina dentre muitos disponíveis. Este processo é particularmente desafiador pois, em um caso típico, 1) há ruídos puramente aleatórios nos dados e 2) há acesso a somente uma amostra bastante limitada da distribuição latente.

De acordo com a definição de meta-aprendizado em (VILALTA *et al.*, 2004; BRAZDIL *et al.*, 2009), o objetivo do meta-aprendizado de máquina é aproximar uma função que descreve uma distribuição probabilística conjunta latente entre os meta-atributos independentes  $X_{\text{meta}}$  e o meta-atributo alvo  $y_{\text{meta}}$ . Os meta-atributos  $X_{\text{meta}}$  são coletados por meio da extração de meta-características de muitos  $(X_{\text{base}}, y_{\text{base}})$  distintos, isto é, estatísticas que capturam características específicas presentes nos dados base. O meta-atributo alvo  $y_{\text{meta}}$  consiste, por exemplo, na performance (de acordo com alguma métrica) de algoritmos de aprendizado de máquina base mapeando  $X_{\text{base}}$  em  $y_{\text{base}}$ , ou seja,  $y_{\text{meta}}$  é construído após múltiplas tarefas de aprendizado de máquina supervisionado.

### 1.3 A necessidade de meta-aprendizado

As aplicações de meta-aprendizado de máquina podem consistir em sistemas capazes de automatizar a criação de *pipelines* de aprendizado de máquina (THORNTON *et al.*, 2013; FEURER *et al.*, 2015; HUTTER; KOTTHOFF; VANSCHOREN, 2018) ou sugerir algoritmos ou procedimentos por meio de meta-modelos que agregam informação com o tempo, tornando o desenvolvimento mais rápido e, quem sabe, menos subjetivo (BRAZDIL *et al.*, 2009).

No contexto de aprendizado por reforço o meta-aprendizado torna-se especialmente importante devido a necessidade de uma grande quantidade de dados para desenvolver um agente apto a solucionar o problema adequadamente (MNIH *et al.*, 2013; SILVER *et al.*, 2016; YARATS *et al.*, 2020), o que implica na necessidade de grande poder computacional e tempo dado que os dados de treinamento em tarefas de aprendizado por reforço são tipicamente coletados durante a otimização do agente (ver seção 2.1). Atualmente, o meta-aprendizado por reforço é realizado por meio de redes neurais recorrentes, onde a rede é treinada para “implementar” algoritmos de aprendizado por reforço que exploram configurações específicas retiradas da distribuição de treino (ou distribuições similares) (ver seção 2.2).

### 1.4 Objetivo deste trabalho

Este texto irá se concentrar em unir o aprendizado de máquina por reforço e a perspectiva de meta-aprendizado de Vilalta *et al.* (2004) e Brazdil *et al.* (2009) (ver seção 2.3). Assim sendo, este trabalho procura investigar a viabilidade de unir técnicas de meta-aprendizado por reforço orientado ao acúmulo de meta-conhecimento sem o uso de Redes Neurais Recorrentes (RNNs). São duas as motivações por trás disso: 1) esta perspectiva de meta-aprendizado não necessariamente se opõe ao uso de RNNs,

podendo potencialmente serem aplicadas em conjunto, e 2) é um problema interessante por si próprio.

Para isto, o objeto de estudo deste trabalho será recomendação de algoritmos de aprendizado por reforço sobre uma família artificial de problemas de *Gridworlds* (ver capítulo 3) com extração de meta-características com semântica específica ao domínio de problema selecionado (ver seções 2.3 e 2.4). Embora este seja talvez um cenário controlado e planejado demais se comparado a um típico problema onde aprendizado por reforço é aplicado, espera-se que o experimento apresentado seja suficiente como prova de conceito para motivar esta linha de pesquisa em trabalhos futuros.

## 1.5 Organização deste trabalho

Este trabalho foi organizado como segue: o capítulo 2 expõe todas as bases teóricas sobre aprendizado de máquina por reforço, como meta-aprendizado por reforço é geralmente utilizado na literatura científica atualmente, e qual a definição de “meta-aprendizado” adotada neste texto. O capítulo 3 propõe um experimento de recomendação de algoritmos para avaliar os efeitos de meta-aprendizado, segundo a definição adotada neste texto, no contexto de aprendizado por reforço. Por fim, o capítulo 4 encerra com as considerações finais do autor e ideias de trabalhos prospectivos que poderiam dar continuidade a linha de pesquisa apresentada neste texto.



## FUNDAMENTOS TEÓRICOS

### 2.1 Aprendizado por reforço

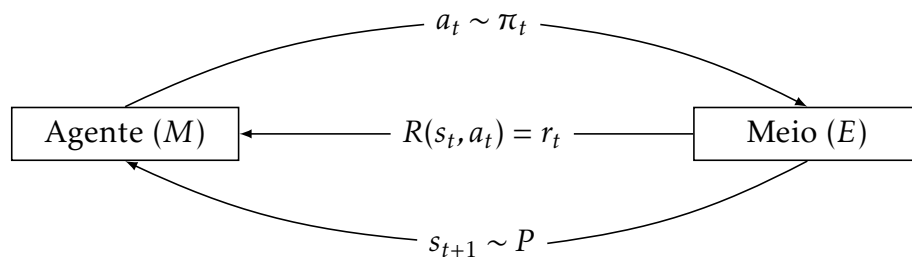
Esta seção é dedicada a introduzir aos leitores e leitoras os jargões e as definições adotadas pela literatura científica na área de aprendizado de máquina por reforço.

Toda tarefa de aprendizado por reforço apresenta três componentes principais:

1. O meio (*environment*) de atuação  $E$ ,
2. Uma função de recompensa (*reward function*)  $R$ , e
3. Um agente (*agent*)  $M$ , que procura maximizar sua recompensa atuando em  $E$ .

Em termos simples, o objetivo do agente  $M$  é encontrar uma estratégia (*policy*) ótima  $\pi^*$  de como atuar no ambiente  $E$  que maximiza a sua recompensa total ao longo do tempo. Para isso, o ambiente  $E$  reage às ações de  $M$  fornecendo recompensas imediatas  $r$  sob alguma regra  $R$  (que pode ou não ser conhecida de antemão) que quantificam a qualidade, ou a pertinência, de cada ação individual de  $M$  no momento que foram tomadas. Baseando-se nos valores de  $r$ , o agente pode ajustar a sua estratégia atual  $\pi$  de modo a se aproximar iterativamente da estratégia ótima  $\pi^*$ . Esta interação entre agente e meio pode ser visualizada na figura 1. Nesta figura, pode-se observar que, para cada

Figura 1 – Esquema de uma tarefa típica de aprendizado de máquina por reforço.



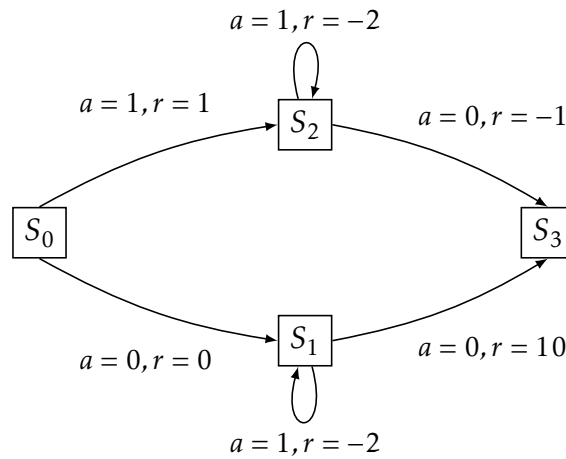
Fonte: Elaborada pelo autor.

tempo  $t$  o agente interage com o meio por meio de ações  $a_t \in A$  (sendo  $A$  um conjunto pré-determinado de ações que o agente pode executar) amostradas de uma distribuição probabilística conjunta dos estados  $S$  e conjunto de ações  $A$  definida pela estratégia

atual  $\pi_t$  do agente:  $\pi_t : S \times A \rightarrow [0, 1]$ . Em resposta, o meio  $E$  entrega ao agente um novo estado  $s_{t+1}$  determinado por um modelo de transições  $P : S \times A \times S \rightarrow [0, 1]$  e uma recompensa imediata  $r_t$  referente a  $(s_t, a_t)$ . Este ciclo pode se repetir indefinidamente, até que o agente tenha concluído a sua tarefa, ou até que o agente esteja incapacitado de concluí-la. O número de passos até a quebra do ciclo chama-se “horizonte”. Quando este ciclo se repete interminavelmente, estamos em um cenário de horizonte infinito. Caso contrário, o horizonte  $T$  é finito e cada sequência individual (isto é, cada vez que o agente voltar a operar após concluir sua tarefa anteriormente, com ou sem sucesso) de estados, ações e recompensas  $(s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$  são chamadas de “episódios”.

Repare que um agente que age sempre de forma gulosa, isto é, adotando a estratégia  $\pi_{\text{greedy}}$  ao maximizar somente sua recompensa imediata não é muito interessante, pois o máximo global de recompensa total pode estar associado a uma sequência de ações que não possuem recompensas imediatas interessantes. De forma análoga, ações associadas a máximos locais podem levar a recompensas totais sub-ótimas. Por exemplo, suponha um agente que pode caminhar entre os nós do dígrafo representado na figura 2. Claramente, um agente que atua seguindo a estratégia gulosa  $\pi_{\text{greedy}}$  receberá um retorno total sub-ótimo. Assim sendo, é interessante orientar o agente a maximizar sua

Figura 2 – Exemplo em que um agente guloso recebe um retorno sub-ótimo (0) seguindo pelo caminho  $s_0 \rightarrow s_2 \rightarrow s_3$ , sendo o caminho ótimo  $s_0 \rightarrow s_1 \rightarrow s_3$  com retorno total de 10. Neste dígrafo, o agente sempre inicia no estado  $S_0$ . As ações ( $a \in \{0, 1\}$ ) e recompensas ( $r$ ) associadas a cada transição estão representadas nas arestas correspondentes. Assume-se que qualquer ação tomada no estado  $S_3$  encerra o processo.



Fonte: Elaborada pelo autor.

recompensa acumulada (isto é, total de recompensa a longo prazo). Para isso, define-se o retorno (*return*) do agente no tempo  $t$  e seguindo a estratégia  $\pi$ ,  $G_t^\pi$ , como a soma das recompensas imediatas de  $t$  em diante com decaimento geométrico  $\gamma \in [0, 1]$  (equação 2.1), e a Função de Valor  $V_t^\pi : S \rightarrow \mathbb{R}$  que mapeia o valor esperado de  $G_t^\pi$  com o agente

estando no estado  $s$  no tempo  $t$  (equação 2.2).

$$G_t^\pi = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{j=t} \gamma^{j-t} r_j \quad (2.1)$$

$$V_t^\pi(s') = \mathbb{E} \left[ \sum_{j=t} \gamma^{j-t} r_j \mid s_t = s' \right] = \mathbb{E}[G_t \mid s_t = s'] \quad (2.2)$$

Para processos com horizonte infinito,  $V_t^\pi = V^\pi$  e  $G_t^\pi = G^\pi$  para todo  $t$ , isto é, são funções estacionárias (invariantes no tempo). Repare que ambas as quantidades,  $G_t^\pi$  e  $V_t^\pi$ , são dependentes da estratégia  $\pi$  do agente. Um agente que opera de forma ótima, isto é, segue a estratégia ótima  $\pi^*$  em todos os estados tem a sua função de valor  $V^*(s) = V^{\pi^*}(s) \geq V^\pi(s)$  para todo  $s \in S$  e estratégia  $\pi$ .

A questão do agente agir de forma gulosa (seguindo a melhor estratégia encontrada até então) ou tentar procurar alternativas melhores corresponde a uma questão delicada na área de aprendizado de máquina por reforço, conhecida como “Dilema Aproveitamento-Exploração” (*Exploration-Exploitation Dilemma*). Como observado anteriormente, um agente que age de forma gulosa seguindo a função de valor  $V$  é ótimo caso siga a função de valor ótima  $V^*$ . Entretanto, para a maioria dos problemas de complexidade considerável, é comum termos apenas aproximações de  $V^*$ , e seguir estimativas sub-ótimas de forma gulosa pode levar a soluções sub-ótimas. Portanto, em uma tarefa de aprendizado de máquina por reforço, é necessário tomar ações que *parecem* ser sub-ótimas de acordo com a nossa estimativa atual  $V$  de  $V^*$ , para que talvez possamos encontrar  $V^*$  e consequentemente  $\pi^*$ . Um método que balanceia *aproveitamento* (seguir a estimativa atual  $V$  de forma gulosa) e *exploração* (tomar decisões que contrariam a estimativa atual  $V$ ) simples porém efetivo e muito utilizado na prática é o algoritmo  $\epsilon$ -greedy (SUTTON; BARTO, 1998):

$$\epsilon\text{-greedy}(\pi(s)) = \begin{cases} \arg \max_a \pi(a \mid s) & \text{com probabilidade } 1 - p \\ a \sim \text{Uniforme}(A) & \text{com probabilidade } p \end{cases}$$

Tipicamente problemas de aprendizado por reforço são formalizados usando Processos de Decisão de Markov e descritos por uma 5-tupla  $(S, A, P, \gamma, R)$ <sup>1</sup>, onde

1.  $S$ : conjunto de estados. Entende-se como “estado” um conjunto suficiente de estatísticas que descrevem a situação atual do agente perante ao meio e que garante a propriedade Markoviana ( $P(s_{t+1} \mid s_t, a_t) = P(s_{t+1} \mid s_0, a_0, s_1, \dots, s_t, a_t)$  para todo  $t \geq 0$ ) do processo<sup>2</sup>. Por exemplo, um robô capaz de se movimentar em

<sup>1</sup> Há formas distintas porém equivalentes de definir um Processo de Decisão de Markov.

<sup>2</sup> Uma estatística suficiente que garante a propriedade Markoviana sempre existe, pois podemos definir trivialmente cada estado como sendo o histórico completo de toda a trajetória do agente a partir de  $t = 0$ . Na prática, buscamos representações mais simples para os estados.

diferentes velocidades pode ter seu estado atual definido como (posição  $x$ , posição  $y$ , velocidade, aceleração, bateria restante).

2.  $A$ : conjunto de ações que o agente pode executar em cada um dos estados para interagir com o meio, podendo ser um conjunto discreto (e.g. um robô que pode escolher mover-se para {"leste", "oeste", "norte", "sul"}) ou contínuo (e.g. um carro autônomo que pode ajustar sua aceleração em  $\mathbb{R}$ ).
3.  $P : S \times A \times S \rightarrow [0, 1]$ : modelo de transições do processo. Define uma distribuição de probabilidade conjunta entre o estado atual do agente  $s_t$  e a ação que o agente execute  $a_t$  e o próximo estado  $s_{t+1}$ .
4.  $\gamma \in [0, 1]$ : fator de desconto. Tipicamente utilizado para impedir que o retorno do agente em processos com horizonte potencialmente infinitos divirjam, i.e.,  $G_t \rightarrow \infty$ .
5.  $R : S \times A \rightarrow \mathbb{R}$ : função de recompensa, que mapeia cada transição dada por  $(s_t, a_t)$  a um número real<sup>3</sup>. Por exemplo, no jogo Pong de Atari pode-se definir a função de recompensa como  $R(\text{Vitória}, a) = 1$  e  $R(\text{Derrota}, a) = 0$ , sendo " $a$ " qualquer ação. Em um outro cenário, onde o agente precisa encontrar o caminho entre a posição inicial e uma posição de destino pré-estabelecida, podemos definir  $R(s, a) = -1$  (onde  $s$  e  $a$  são quaisquer estados e ações, respectivamente) para estimular o agente a encontrar o menor caminho possível.

Os algoritmos de aprendizado de máquina por reforço que usam modelos do meio (isto é, possuem acesso a função de transição  $P$  e função de recompensa  $R$ ) são "Baseados em Modelo" (*Model-based algorithms*), e tipicamente envolvem elementos de programação dinâmica (PD). Evidentemente, nem sempre temos um modelo do meio disponível. Para situações como esta, a literatura de aprendizado de máquina por reforço apresenta diversos algoritmos que não dependem do conhecimento de prévio de  $P$  e/ou  $R$  e sendo, portanto, classificados como "Livres de Modelo" (*Model-free algorithms*), e geralmente são baseados em técnicas de amostragem Monte Carlo (MC) e *Bootstrapping* (BTSP). Amostragem Monte Carlo possui viés baixo porém variância alta, enquanto que amostragem por *Bootstrapping* possui variância baixa e viés alto.

Lembrando que o objetivo primário é encontrar a estratégia ótima  $\pi^*$  que determina quais ações o agente toma em relação a cada estado. Existem diversas formas de estimar  $\pi^*$  direta ou indiretamente. As principais estratégias encontradas na literatura são:

<sup>3</sup> Note que há outras formas de definir a função de recompensa  $R$  como, por exemplo,  $R : S \times A \times S \rightarrow \mathbb{R}$  sendo em função de  $(s_t, a_t, s_{t+1})$ , ou simplesmente  $R : S \rightarrow \mathbb{R}$ .



- Iteração de estratégia (*Policy Iteration*): procura-se  $\pi^*$  iterativamente, melhorando a estratégia atual  $\pi$ . É baseado em modelo. **Exemplos de algoritmos:** Policy Iteration (HOWARD, 1960; PUTERMAN, 1994).
- Estimativa da função de valor  $V$  (*Value Iteration*): primeiro estima-se a função de valor  $V$  para cada estado, e em seguida deriva-se uma estratégia  $\pi$  de  $V$ . O quadro 1 sumariza as principais propriedades de cada tipo de algoritmo para estimar a função de valor ótima  $V^*$ . **Exemplos de algoritmos:** Value Iteration (PD) (BELLMAN, 1957), Monte Carlo Control (MC), SARSA (BTST) (RUMMERY; NIRANJAN, 1994; SUTTON, 1996), Q-Learning (BTST) (WATKINS, 1989), Double Q-Learning (BTST) (HASSELT, 2010).
- Gradiente de estratégia (*Policy Gradient*): parametriza-se  $\pi$  com um conjunto de parâmetros  $\theta$ , reduzindo a busca por  $\pi^*$  para um problema de otimização em função de  $\theta$ . Neste caso, é necessário assumir uma classe de funções para  $\pi_\theta$  como, por exemplo, funções lineares (neste caso,  $\pi(s, a; \theta) = [s \ a]^T \theta$ ) ou funções mais complexas, como redes neurais profundas (neste caso,  $\theta$  sumariza todo o conjunto de parâmetros da rede). **Exemplos de algoritmos:** REINFORCE (WILLIAMS, 1992). Para mais exemplos, sugere-se o artigo de Weng (2018).

Quadro 1 – Principais propriedades de algoritmos baseados em Programação Dinâmica (PD) e amostragens Monte Carlo (MC) e *Bootstrapping* (BTSP) para estimar a função de valor  $V$ . (\*): assumindo infinitas repetições.

Propriedade	PD	MC	BTSP
Requer modelo do meio	✓	×	×
Assume propriedade Markoviana	✓	×	✓
Requer horizonte finito (processo episódico)	×	✓	×
Convergência garantida para o valor real (*)	✓	✓	✓

Até agora, assumimos que o conjunto de estados  $S$  e o conjunto de ações  $A$  são finitos e *relativamente* pequenos, e podem ser representados explicitamente de uma forma tabular para cada par  $(s, a) \in S \times A$ . Entretanto, quando ao menos um destes conjuntos é muito grande ou até mesmo infinito, a representação explícita torna-se muito custosa ou até mesmo impossível. Para isso, recorreremos a *representações aproximadas* parametrizadas por  $\theta$  da função de valor  $V_\theta$  com algoritmos de *Value Function Approximation* ou restringimos a classe de possíveis estratégias  $\pi_\theta$  e então utilizamos *Policy Gradient*.

Além da representação compacta, outra vantagem deste tipo de representação é que o agente torna-se capaz generalizar para pares de estado-ação  $(s, a)$  nunca vistos anteriormente. Algoritmos do estado-da-arte na área de aprendizado de máquina por reforço usam deste recurso como, por exemplo, *Deep Q-Learning* (MNIH *et al.*, 2013) e suas variantes (como *Double Deep Q-Learning* (HASSELT; GUEZ; SILVER, 2015) e

*Dueling Deep Q-Learning* (WANG *et al.*, 2016)). Como nota final, “*deep reinforcement learning*” normalmente refere-se a algoritmos de aprendizado por reforço com representações aproximadas usando redes neurais profundas (LECUN; BENGIO; HINTON, 2015).

Para uma abordagem aprofundada sobre os tópicos abordados nesta seção, sugere-se Sutton e Barto (2018).

## 2.2 Meta-aprendizado para aprendizado por reforço

O termo “meta-aprendizado” foi originalmente proposto em Schmidhuber (1987) e Hinton e Plaut (1987) de forma independente e historicamente foi utilizado por muitos autores e autoras para se referir a diferentes definições do que é “aprender a aprender” e como atingir tal objetivo. Esta seção é dedicada a mencionar brevemente alguns trabalhos relacionados especificamente a meta-aprendizado no contexto de aprendizado de máquina por reforço ou então de alguma forma pertinentes para contextualizar este texto. Para pesquisas extensivas do uso do termo “meta-aprendizado” na literatura, sugere-se Lemke, Budka e Gabrys (2015) e Hospedales *et al.* (2020).

Adaptando a definição de aprendizado de máquina de Mitchell (1997) para o contexto de meta-aprendizado, em Thrun e Pratt (1998) define-se que um algoritmo  $A$  “aprende a aprender” se, dado:

1. Uma família de tarefas  $T$ ,
2. Experiência de treinamento  $E$  obtida por cada uma das tarefas em  $T$ , e
3. Uma família de medidas de performance  $P$  (e.g. uma para cada tarefa em  $T$ ),

diz-se que o algoritmo  $A$  aprende se sua performance (avaliada com  $P$ ) melhora com a experiência  $E$  e o número de tarefas  $|T|$ . Assim sendo, de acordo esta definição em particular, um algoritmo que não se beneficia com o aumento do número de tarefas distintas (porém de alguma forma similares entre si) a serem aprendidas não “aprende a aprender”. Entretanto, quando um algoritmo “aprende a aprender”, isto é, realiza meta-aprendizado sob a definição de Thrun e Pratt (1998), então há algum grau de transferência da experiência obtida  $E_t$  em uma tarefa particular  $t \in T$  pelo algoritmo de aprendizado de máquina entre as outras tarefas em  $T$ , alavancando assim a sua capacidade de aprender o conjunto completo de tarefas e exibindo custo total de treinamento menor do que a soma dos custos de aprendizado de cada tarefa separada.

Trabalhos anteriores demonstraram que redes neurais recorrentes com pesos fixos ainda podem aprender (COTTERN; CONWELL, 1990; YOUNGER; CONWELL; COTTER, 1999), isto é, adaptar-se a novas situações mesmo após o término da etapa de

treinamento justamente por conta de sua recorrência interna (PROKHOROV; FELDKARNP; TYUKIN, 2002). Seguindo esta ideia, em Hochreiter, Younger e Conwell (2001) foi usada pela primeira vez uma Rede Neural Recorrente (RNN) do tipo *Long-Short Term Memory* (LSTM) (HOCHREITER; SCHMIDHUBER, 1997) no contexto de meta-aprendizado supervisionado (seguindo a definição de meta-aprendizado em Thrun e Pratt (1998)). Santoro *et al.* (2016) utilizam Máquinas de Turing Neurais (GRAVES; WAYNE; DANIHELKA, 2014) para meta-aprendizado também no contexto de aprendizado supervisionado, e explora o ganho de eficiência na tarefa de aprendizado ao usar memórias externas como suporte às memórias internas dos modelos de RNN.

O primeiro trabalho que reúne os termos “meta-aprendizado” e aprendizado por reforço foi Schmidhuber, Zhao e Wiering (1996). Em Schweighofer e Doya (2003), os autores entendem “meta-aprendizado” como a otimização de hiper-parâmetros (ou, como descrito pelos autores, “meta-parâmetros”) relacionados a algoritmos de aprendizado por reforço:  $\gamma$  (fator de desconto),  $\alpha$  (taxa de aprendizado) e  $\beta$  (“temperatura inversa”, expoente na função *soft-max* utilizada para seleção estocástica de ações em domínio discreto), bem como novos hiper-parâmetros introduzidos pelos autores.

Duan *et al.* (2016) e Wang *et al.* (2017) ampliam o uso de RNN para o contexto de meta-aprendizado por reforço. Estes dois últimos trabalhos foram feitos de forma independente, mas exploram o mesmo conceito: codificar um algoritmo genérico de meta-aprendizado nos pesos de uma RNN que captura similaridades entre tarefas distintas e seja capaz de gerar algoritmos de aprendizado por reforço que explorem particularidades em cada tarefa. A diferença entre os dois trabalhos está nos tipos de tarefas consideradas para a análise de meta-aprendizado.

No primeiro trabalho (DUAN *et al.*, 2016) os autores apresentam o algoritmo  $RL^2$ , que consiste no uso de uma RNN do tipo *Gated Recurrent Unit* (GRU) (CHO *et al.*, 2014), onde a cada passo a rede é re-alimentada com um *embedding*  $\phi(s_{t+1}, a_t, r_t, d_t)$  que agrega informação do próximo estado  $s_{t+1}$ , da ação  $a_t$  e recompensa  $r_t$  anteriores, e um valor booleano  $d_t$  indicando se o episódio foi encerrado. O objetivo é ajustar os pesos da RNN de modo que, dado uma tarefa similar às tarefas utilizadas durante o treinamento (isto é, tarefas nunca antes vistas da mesma distribuição amostral de treino ou uma distribuição levemente modificada), a rede possa produzir um algoritmo de aprendizado de máquina por reforço que leva em consideração as características específicas da amostra atual.

No segundo trabalho (WANG *et al.*, 2017), meta-aprendizado por reforço é chamado de “Meta-aprendizado *profundo* por reforço” para deixar claro a distinção do termo em relação a “meta-aprendizado por reforço” em trabalhos anteriores, e sugere que as principais características de meta-aprendizado (profundo) por reforço são:

1. Possuir uma rede neural com memória interna (e.g. RNN, LSTM, GRU, Máquinas de Turing Neurais) treinada por meio de algum algoritmo de aprendizado por reforço, usando um
2. conjuntos de tarefas inter-relacionadas (isto é, há intersecção em suas propriedades e características), e
3. re-alimentada com informações sobre a ação tomada e recompensa imediata obtida no passo anterior.

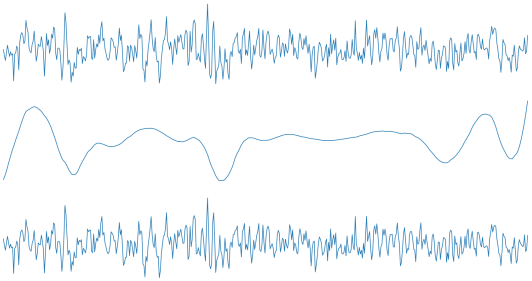
Com estas três características, os autores afirmam que o algoritmo pode resolver um problema de meta-aprendizado (profundo) por reforço. Assim sendo, assim como no trabalho anterior, a RNN é capaz de implementar algoritmos de aprendizado de máquina por reforço que levam as características específicas a cada amostra, enfatizando que o próprio treinamento da RNN é um problema de aprendizado por reforço completo.

## 2.3 Meta-aprendizado e mineração de dados

No contexto de mineração de dados [Vilalta \*et al.\* \(2004\)](#) e [Brazdil \*et al.\* \(2009\)](#) definem meta-aprendizado como o acúmulo de uma base de meta-conhecimento agregada por um meta-modelo. Este meta-conhecimento é acumulado ao extrair meta-características de dados de treinamento de problemas distintos. Dado o acúmulo suficiente de meta-conhecimento, o meta-modelo poderá eventualmente ser usado para guiar ou automatizar decisões sobre processamento de dados e aprendizado de máquina em futuros problemas, bastando extrair suas meta-características e utilizando o meta-modelo para deduzir o resultado esperado sem precisar efetivamente implementar diversas soluções candidatas e comparar os resultados. Assim, pode-se evitar a necessidade de perícia, interferência ou opiniões humanas em partes ou até mesmo em todo o processo de desenvolvimento, além de poupar recursos como poder computacional e tempo.

Repare que esta definição possui diversas similaridades com a definição em [Thrun e Pratt \(1998\)](#). Em particular, em ambas as definições o meta-algoritmo se beneficia com o número de “tarefas”  $|T|$ , onde aqui entende-se como “tarefas” os problemas base em que as meta-características são extraídas, e em ambas as visões o meta-modelo adquire experiência de uma família de tarefas distintas. Entretanto, em [Vilalta \*et al.\* \(2004\)](#) e [Brazdil \*et al.\* \(2009\)](#) há uma clara ênfase na extração de meta-características de cada problema base, diferente da definição de [Thrun e Pratt \(1998\)](#).

Quadro 2 – Exemplos de meta-características de uma série temporal. Na figura, a série temporal  $S$  original é representada pelo gráfico superior. O seu componente de tendência  $T$  (*trend*) é exibido separadamente no gráfico do meio, e o componente de ruído  $R$  no gráfico inferior, onde  $S = T + R$ . As meta-características foram extraídas usando a biblioteca *ts-Pymfe* (disponível em: <https://github.com/FelSiq/ts-pymfe>) para a linguagem de programação Python.

Série temporal (original, tendência e ruído)	Meta-característica	Valor
	acf.mean	-0.0027
	acf.sd	0.0788
	acf_detrended.mean	-0.0034
	emb_dim_cao	4.0000
	emb_lag	2.0000
	lz_complexity	0.9154
	model_arima_010_c.mean	0.4831
	moving_acf.mean	0.4565
	opt_boxcox_coef	1.1639
	pacf_detrended.mean	-0.0130
	pacf_diff.sd	0.0881
	period	1.0000

Fonte: Elaborada pelo autor.

Estas meta-características devem garantir interpretabilidade e coerência com a sua definição matemática entre múltiplos domínios de dados, isto é, devem gerar informação independentemente da natureza de cada conjunto de dados base específico.

Seguindo esta definição de meta-aprendizado, em Rivolli *et al.* (2019) meta-características são formuladas segundo a equação 2.3, onde  $f$  é uma função responsável por extrair um conjunto de valores intermediários dos dados utilizando argumentos adicionais  $h_f$ ,  $\sigma$  é uma função de sumarização, que mapeia a imagem de  $f$  a um espaço de dimensão fixa (tipicamente  $\mathbb{R}$ ) utilizando argumentos adicionais  $h_\sigma$ . A composição  $\sigma \circ f$  mapeia dados a meta-características, i.e.,  $m$  (equação 2.3) é uma meta-característica extraída de  $(X, y)$ .

$$m = \sigma(f(X, y, h_f), h_\sigma) \quad (2.3)$$

Como detalhe adicional, só faz sentido sumarizar valores intermediários que ainda não estão em  $\mathbb{R}$ , isto é, se  $f$  extrai um único valor real de  $(X, y)$  então não há necessidade de uma função de sumarização. Para manter consistência com a equação 2.3, neste caso assume-se que  $\sigma(x) = x$  é a função identidade. Exemplos de meta-características extraídas de uma única série temporal podem ser vistas no quadro 2.

## 2.4 Meta-características para aprendizado por reforço

Diferente do método geralmente adotado para meta-aprendizado em conjunto com aprendizado por reforço na literatura, este trabalho propõe unir aprendizado por reforço e meta-aprendizado seguindo a linha de meta-aprendizado de Vilalta *et*

*al.* (2004) e Brazdil *et al.* (2009). Esta visão de meta-aprendizado procura explorar o histórico de aplicações de aprendizado de máquina, usando o conhecimento acumulado para tornar o desenvolvimento de *pipelines* de aprendizado de máquina mais rápidos e eficientes, e até automáticos por meio do uso de *AutoML* (THORNTON *et al.*, 2013; FEURER *et al.*, 2015; HUTTER; KOTTHOFF; VANSCHOREN, 2018).

O meta-conhecimento será gerado extraindo informações estatísticas do meio. Neste trabalho, esta ideia será apresentada no contexto de um problema artificial de *Gridworlds* e usando algoritmos base tabulares (ver capítulo 3). Mesmo assim, acredita-se ser possível extrapolar estas ideias para cenários mais complicados e utilizando algoritmos de aprendizado por reforço mais sofisticados.

As meta-características apresentadas neste texto serão específicas a *Gridworlds* (ver seção 3.4). Contudo, espera-se que esta perspectiva possa despertar novas ideias em trabalhos prospectivos durante a resolução de problemas com algoritmos de aprendizado por reforço e também inspirar a pesquisa de meta-características genéricas que façam sentido em múltiplas famílias de meios.

Note que a linha de pesquisa seguida neste texto não necessariamente se opõe aos métodos geralmente adotados na literatura de meta-aprendizado por reforço. Mais especificamente, as ideias propostas neste texto podem naturalmente ser aplicadas em conjunto com as ideias de meta-aprendizado utilizando redes neurais recorrentes, que assumiriam o papel de “modelos base” (apesar de serem, de certo modo, “meta-modelos”).

Portanto, de agora em diante, o termo “meta-aprendizado” irá se referir especificamente a definição presente em Vilalta *et al.* (2004) e Brazdil *et al.* (2009). Consequentemente, “meta-características” seguirão a formulação de Rivolli *et al.* (2019) (ver equação 2.3). “Meta-conjunto de atributos independentes” irá se referir a meta-características extraídas do meio e denotado por  $X$ , “meta-atributo alvo” será a performance relacionadas aos algoritmos de aprendizado base e denotado por  $y$ , e “meta-modelo” irá se referir a modelos que mapeiam  $X$  em  $y$ .

---

## EXPERIMENTOS

---

### 3.1 Proposta

Para unir aprendizado de máquina por reforço e meta-aprendizado sob a perspectiva de Vilalta *et al.* (2004) e Brazdil *et al.* (2009), será proposto um experimento de recomendação de algoritmos de aprendizado por reforço em meios artificiais parametrizados aleatoriamente de *Gridworlds*. Neste experimento, serão utilizados quatro algoritmos base de aprendizado por reforço com representação tabular (i.e., há representação explícita para todo  $(s, a) \in S \times A$ ): *SARSA*, *Q-Learning*, *Double Q-Learning* e *Monte Carlo Control*. Para cada um dos algoritmos, múltiplos agentes serão treinados em amostras da distribuição dos meios por um número de episódios pré-determinado, formando o meta-conjunto de dados.

Para todo agente, após sua etapa de treinamento, uma execução de validação será executada com sua estratégia (*policy*) fixa, e a recompensa total será registrada. Estas recompensas finais irão compor o meta-atributo alvo  $y$ . Em paralelo, meta-características serão extraídas dos meio em que os agentes são treinados para compor os meta-atributos independentes  $X$ . Por fim, um meta-modelo será construído para mapear  $X$  em  $y$  de modo que, dado uma configuração de meta-características de uma nova amostra de meio nunca antes vista, o meta-modelo possa deduzir a recompensa final do agente para cada um dos algoritmos base.

As seções seguintes são organizadas da forma que segue: a seção 3.2 define as regras de um meio *Gridworld*. A seção 3.3 descreve exatamente como os meios são gerados. A seção 3.4 descreve como os meta-dados ( $X$  e  $y$ ) serão coletados. A seção 3.5 descreve como o meta modelo fora configurado e seus hiper-parâmetros ajustados. Por fim, a seção 3.6 expõe os resultados obtidos.

Todos os códigos, testes e meta-dados utilizados nos experimentos estão disponíveis *online* no Github<sup>1</sup>.

---

<sup>1</sup> <https://github.com/FelSiq/meta-reinforcement-learning>



## 3.2 Regras do meio *Gridworld*

- Em um ambiente de *Gridworld*, o objetivo do agente é encontrar um caminho até um dos objetivos, i.e., células de recompensa positiva em um tabuleiro bidimensional de dimensões  $(h, w)$ , evitando armadilhas, i.e., células de recompensa negativa.
- O espaço de ações do agente é  $A = \{\text{“Oeste } \leftarrow\text{”}, \text{“Leste } \rightarrow\text{”}, \text{“Norte } \uparrow\text{”}, \text{“sul } \downarrow\text{”}\}$ .
- O espaço de estados é definido pelas possíveis posições do agente  $S = \{1, \dots, h\} \times \{1, \dots, w\}$  e, portanto, um estado é um par ordenado  $(y, x)$  que descreve a posição atual do agente.
- O agente só pode movimentar-se para células não bloqueadas e dentro dos limites do *Gridworld*. Ao tentar movimentar-se para uma célula indisponível, o agente permanecerá na mesma posição. Não foi estabelecida uma recompensa particular para este cenário.
- Ao atingir uma célula objetivo o agente recebe recompensa positiva (+1) e o episódio termina imediatamente.
- Ao atingir uma célula armadilha o agente recebe recompensa negativa (-1) e o episódio termina imediatamente.
- Durante a geração do *Gridworld*, é garantido que existe ao menos um caminho entre o agente e todas as células objetivo e armadilhas.

Na figura 3 pode-se observar uma amostra da distribuição de *gridworlds* usada neste texto.

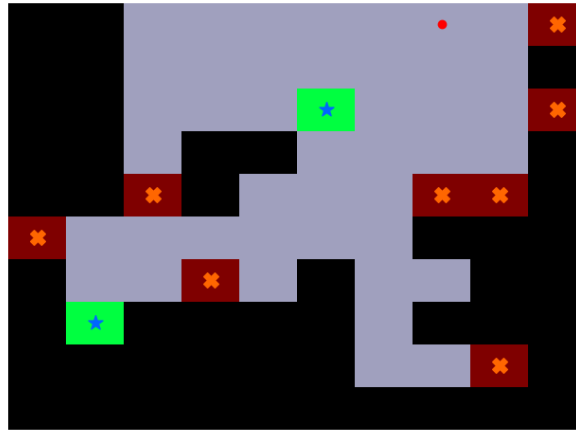
## 3.3 Configuração e geração dos *Gridworlds*

A distribuição de *Gridworlds* usada neste texto possui seis hiper-parâmetros mostrados no quadro 3 com seus respectivos valores mínimo (inclusivo) e máximo (exclusivo). Para cada meio, os hiper-parâmetros são amostrados de uma distribuição uniforme entre os valores limites pré-estabelecidos.

Durante a geração de cada *Gridworld*, cada meio é iniciado com todas as suas células bloqueadas. A posição do agente é escolhida aleatoriamente (uniforme em todas as células). Nas células restantes, as células objetivo e armadilhas são selecionadas uma a uma, sem reposição. Em seguida, variantes da busca *Hill Climbing* são iniciadas em cada célula objetivo e célula armadilha com destino a posição inicial do agente, garantindo assim que toda célula final (objetivo e armadilha) sejam acessíveis ao agente. Todas as



Figura 3 – Exemplo de um *Gridworld* gerado aleatoriamente. Na figura, as dimensões do meio são 10x10. O agente encontra-se em seu estado inicial (definido aleatoriamente também durante a geração do meio) e é representado na figura pelo pequeno círculo vermelho. As células pretas são bloqueadas e impedem a passagem do agente, e todas as outras células estão disponíveis. As células objetivo (recompensa positiva) estão representadas em verde e um símbolo ★, enquanto que as células armadilhas estão representadas em cor vermelha e com um símbolo ✖.



células bloqueadas visitadas por qualquer execução destas buscas são convertidas para células disponíveis ao agente.

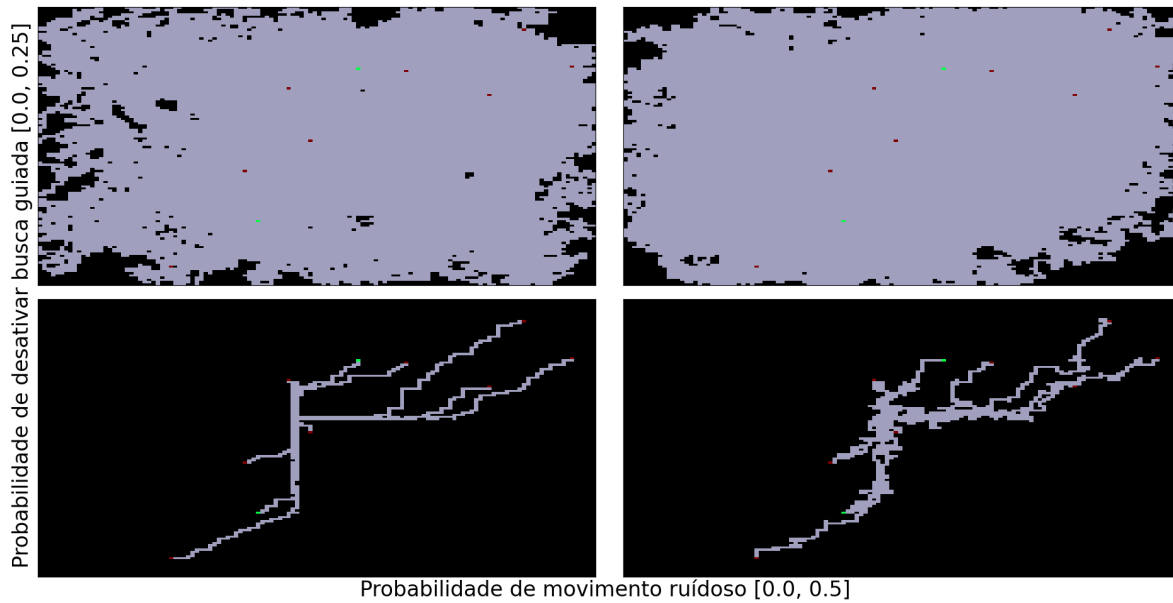
Entretanto, nesta variante de *Hill Climbing* em toda iteração há uma probabilidade  $p_a$  (“probabilidade de movimento ruidoso”) do movimento atual ser convertido em um movimento ruidoso, i.e., movendo-se para a direção oposta. Além disso, há também uma outra probabilidade  $p_b$  (“probabilidade de desativar movimento guiado”) da posição da célula objetivo (posição inicial do agente) ser esquecida, tornando-se assim uma busca cega pelas próximas iterações subsequentes. A posição da célula objetivo pode ser lembrada também com probabilidade  $p_b$  nas iterações seguintes. Os efeitos de ambos os parâmetros podem ser visualizados com clareza na figura 4.

Quadro 3 – Hiper-parâmetros associados com a distribuição probabilística dos meios utilizados durante os experimentos. Valores mínimos são inclusos, diferente dos valores máximos. Todos os hiper-parâmetros foram selecionados de forma independente e uniforme em relação ao conjunto de valores válidos correspondente.

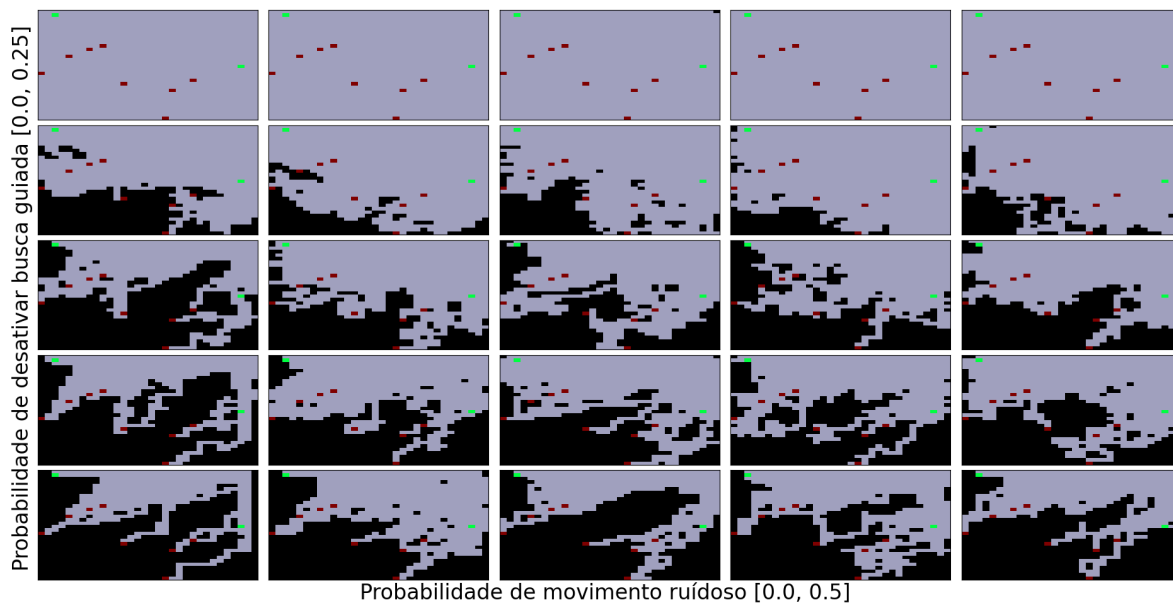
Hiper-parâmetro da distribuição do meio	Mínimo (inc)	Máximo (exc)
Largura (em células)	10	32
Altura (em células)	10	32
Quant. de células armadilha (recompensa negativa)	8	32
Quant. de células objetivo (recompensa positiva)	1	6
Probabilidade de movimento ruidoso	0.0	0.50
Probabilidade de desativar movimento guiado	0.0	0.25

Como detalhes finais, o agente recebe recompensa  $r = -0.001$  por cada ação tomada, para incentivá-lo a encontrar o menor caminho entre seu ponto de partida e uma célula de objetivo e a priorizar células objetivo mais próximas do ponto inicial, e

Figura 4 – Impacto dos hiper-parâmetros “Probabilidade de Movimento Ruidoso” (eixo horizontal) e “Probabilidade de Desativar Busca Guiada” (eixo vertical). O agente só pode movimentar-se por *pixels* claros. As figuras no canto esquerdo-inferior possuem ambos os parâmetros iguais a zero; as figuras superiores possuem probabilidade de desativação de busca guiada no máximo valor permitido nos experimentos (0.25); e as figuras no canto direito possuem a probabilidade de movimento ruidoso configuradas no máximo permitido nos experimentos (0.50). Em (b), os hiper-parâmetros das figuras intermediárias são interpolações lineares entre os valores extremos.



(a) Meio com dimensões 128x128



(b) Meio com dimensões 32x32

também foi pré-estabelecido um máximo de 1000 unidades de tempo por episódio, isto é, o agente pode tomar até 1000 ações durante um mesmo episódio, caso contrário o episódio é encerrado com recompensa negativa (-1).

## 3.4 Extração de meta-características e meta-dados

Esta seção é dedicada a descrever o processo de coleta de meta-dados. A subseção 3.4.1 descreve as meta-características extraídas de cada meio (configuração distinta de *Gridworld*) e que compõe os meta-atributos independentes  $X$ , enquanto que a subseção 3.4.2 descreve como os meta-atributos alvo  $y$  foram coletados. Por fim, a subseção 3.4.3 sumariza esta seção, reapresentando as principais informações.

### 3.4.1 Meta-atributos independentes $X$

Para compor o meta-conjunto de atributos independentes  $X$ , duas famílias de meta-características serão utilizadas: “artificial” e “elaboradas”. A família de meta-características “artificial” é composta pelos próprios hiper-parâmetros usados para gerar cada configuração de *Gridworld* (ver quadro 3). A família de meta-características “elaboradas” foram projetadas seguindo a formulação de meta-característica em Rivolli *et al.* (2019), isto é, há uma combinação entre uma função  $f$  que extrai valores intermediários de cada configuração de *Gridworld*, e uma função de sumarização  $\sigma$  que mapeia estes valores intermediários em números reais (assumindo que  $f$  não mapeia a configuração de *Gridworld* aos números reais - caso contrário,  $\sigma(x) = x$  como definido na seção 2.3). As funções que extraem valores intermediários são:

1. **EG**: Distância Euclidiana entre a posição inicial do agente e células objetivo;
2. **MG**: Distância Manhattan entre a posição inicial do agente e células objetivo;
3. **EA**: Distância Euclidiana entre a posição inicial do agente e células armadilha;
4. **MA**: Distância Manhattan entre a posição inicial do agente e células armadilha;
5. **REG**: EG filtrado apenas para células objetivo com distância Euclidean até  $(\text{altura} + \text{largura})/4$  da posição inicial do agente;
6. **REA**: EA filtrado apenas para células armadilha com distância Euclidean até  $(\text{altura} + \text{largura})/4$  da posição inicial do agente; e
7. **PropWP**: Proporção de células bloqueadas em um recorte retangular do meio, centralizado na posição inicial do agente, e se estendendo em até 10% de cada dimensão correspondente em ambos os sentidos.

As funções de sumarização são: valor máximo, mínimo, média, desvio padrão, mediana, curtose, obliquidade, soma e dimensão do valor intermediário. No total, combinando de todas as maneiras todas as funções de extração e sumarização descritas anteriormente, o resultado final são 55 meta-características distintas (6 funções de sumarização  $\times$  9 funções sumarizáveis + 1 função não sumarizável).

### 3.4.2 Meta-atributos alvo $y$

Em relação aos atributos alvo  $y$ , cada algoritmo foi executado centenas de vezes nos mesmos meios durante números pré-determinados de episódios, e as recompensas acumuladas finais de cada agente coletadas em seguida em uma única iteração de validação com a estratégia do agente (*policy*) fixa. Mais especificamente, foram amostrados 2000 *Gridworlds* distintos em que cada algoritmo treinou um agente por 500 episódios, 750 *Gridworlds* distinto em que cada algoritmo treinou um agente por 1000 episódios, e 250 *Gridworlds* com agentes treinados por 3000 episódios por algoritmo. A figura 5 sumariza os resultados obtidos em todos os cenários e para os quatro algoritmos base.

### 3.4.3 Juntando as informações

Dado a descrição de ambas as subseções anteriores, temos então:

- 6 meta-conjunto de atributos independentes  $X$ : meta-características extraídas para 3000, 1000 e 500 episódios de treinamento de ambas as famílias de meta-características: “artificial” (6 atributos) e “elaboradas” (55 atributos).
- 3 meta-conjuntos de atributos alvo  $y$ : recompensa acumulada de cada agente após treinamento em 3000, 1000 e 500 episódios. Note que cada conjunto  $y$  possui quatro colunas, uma para cada algoritmo base considerado no experimento.

A figura 6 ilustra o cenário descrito.

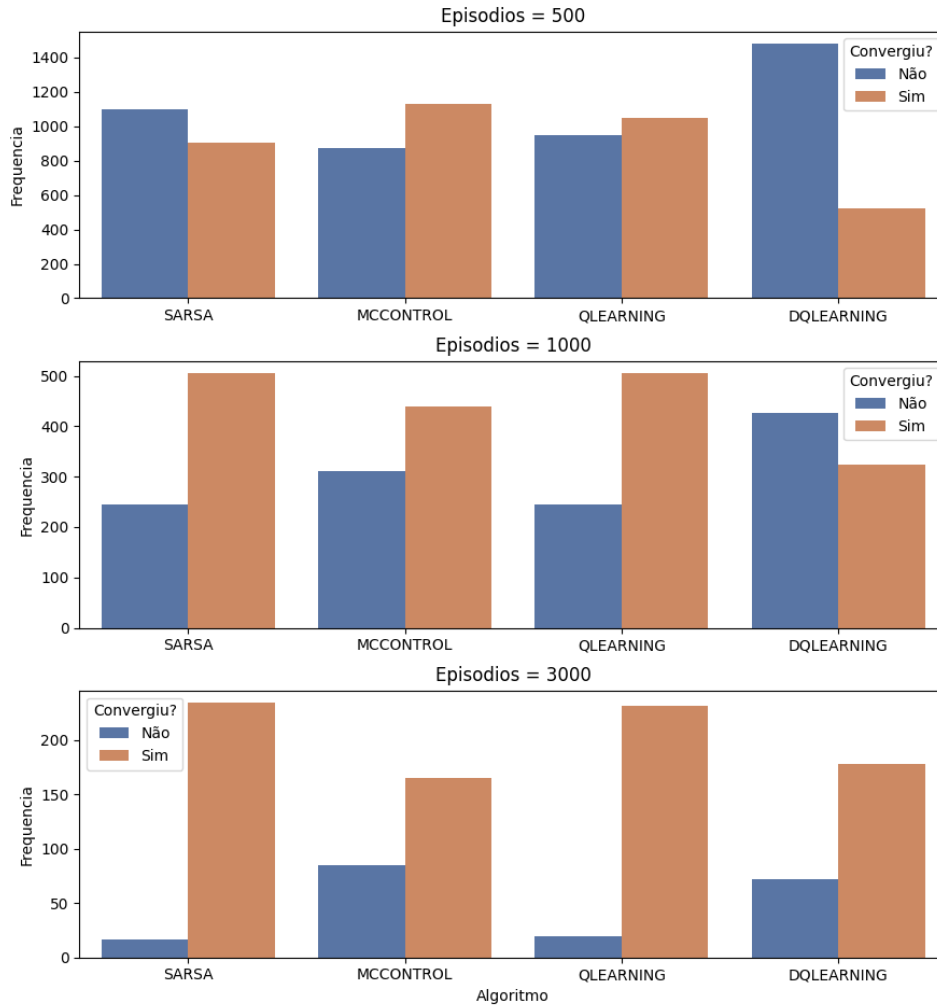
## 3.5 Meta-modelo

O algoritmo utilizado para induzir os meta-modelos foi o *Extreme Gradient Boosting* (XGBoost) (CHEN; GUESTRIN, 2016). Para a recomendação de algoritmos dois tipos de meta-aprendizado supervisionado foram realizados: meta-classificação e meta-regressão.

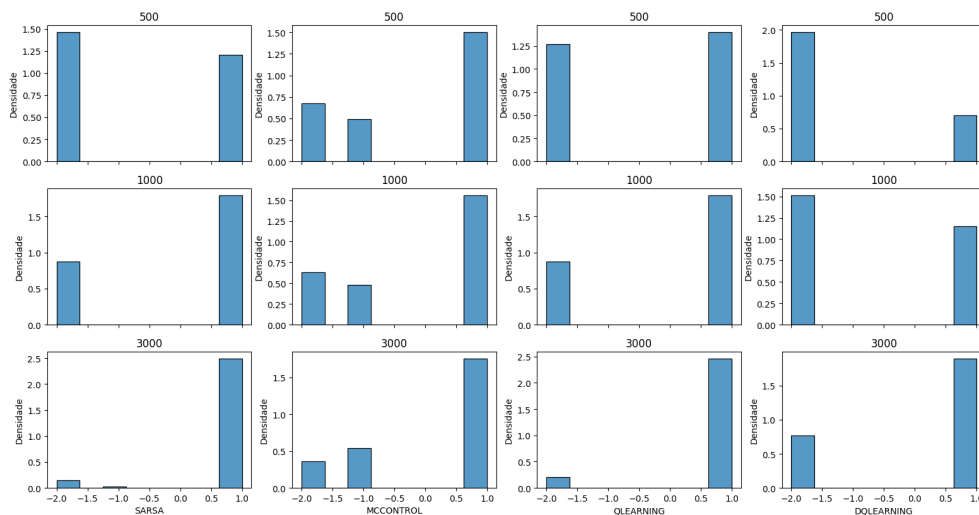
Mais precisamente, os meta-classificadores devem prever se cada algoritmo de aprendizado base irá convergir dentro do limite máximo de episódios de treinamento pré-estabelecido dado as meta-características de um *Gridworld* novo, isto é, se a recompensa acumulada final seguindo a estratégia (*policy*) do agente aprendida durante o treinamento será estritamente positiva.

Os meta-regressores realizam uma tarefa similar, com a diferença que é necessário que deduzam de forma precisa a recompensa acumulada do agente após o treinamento com cada algoritmo base. Esta distinção é importante pois permite diferenciar os algoritmos base com granularidade mais fina: entre dois algoritmos que

Figura 5 – Distribuição de (a) convergência e (b) recompensa acumulada em um único episódio dos algoritmos de aprendizado base após o treinamento. Cada coluna representa um algoritmo distinto (SARSA, Monte Carlo Control, Q-Learning e Double Q-Learning, nesta ordem da esquerda para a direita), e cada linha representa números distintos de episódios de treinamento (500, 1000 e 3000, nesta ordem de cima para baixo).

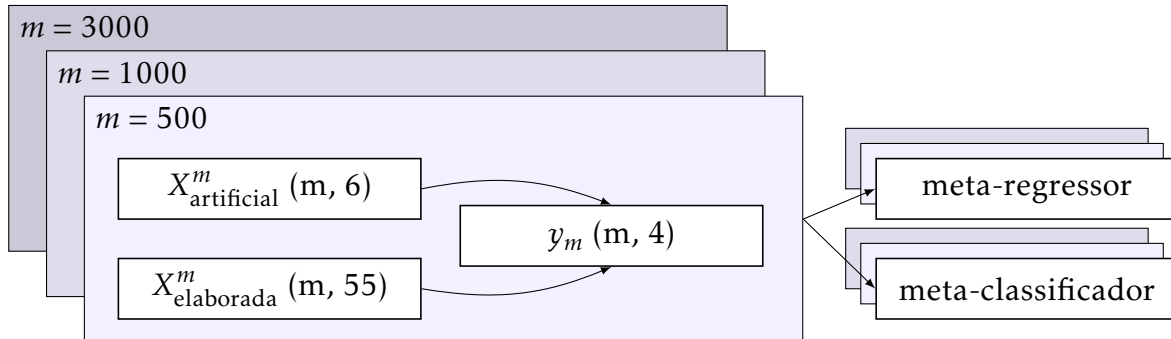


(a) Distribuição de convergência para uma solução válida (i.e., recompensa acumulada estritamente superior a 0 em execução de teste).



(b) Distribuição de recompensa final.

Figura 6 – Relação entre os meta-atributos independentes  $X$  e alvo  $y$ . Cada conjunto de meta-atributo alvo  $y$  possui dois conjuntos de meta-atributos independentes  $X$ . Na figura,  $(m, n)$  indica que o conjunto de dados correspondente possui  $m$  linhas e  $n$  colunas. Nos experimentos foram usados  $m \in \{500, 1000, 3000\}$ . Cada par  $(X, y)$  é usado para induzir um meta-classificador e um meta-regressor distinto.



Fonte: Elaborada pelo autor.

Quadro 4 – Hiper-parâmetros do algoritmo *XGBoost* usados na etapa de treinamento dos meta-modelos. Todos os parâmetros que não apareceram explicitamente neste quadro foram definidos como os valores padrão da biblioteca *xgboost* para a linguagem de programação Python.

Hiper-parâmetro do <i>XGBoost</i>	Episódios de treinamento		
	500	1000	3000
learning_rate	0.08	0.06	0.1
n_estimators (num_boost_round)	8000	8000	8000
max_depth	5	5	5
min_child_weight	30	20	20
gamma	0.5	0.5	1.0
alpha	0.5	0.5	0.0
lambda	400	200	3
subsample	0.7	0.7	0.8
colsample_bytree	0.3	0.4	0.8
scale_pos_weight	0.90	0.85	0.50
early_stopping_rounds	500	500	500
objective	reg:squarederror (meta-regressor)		
	binary:logistic (meta-classificador)		

convergem após o treinamento (isto é, os respectivos agentes acumulam recompensa positiva seguindo a estratégia encontrada), o melhor algoritmo é o que encontra uma estratégia capaz de gerar maior recompensa acumulada (no contexto do *Gridworld* estas estratégias direcionam o agente até a célula objetivo mais próxima por meio do menor caminho possível).

O quadro 4 sumariza os hiper-parâmetros utilizados do *XGBoost*. A implementação de *XGBoost* utilizada está disponível na biblioteca *xgboost* para a linguagem de programação Python<sup>2</sup>.

<sup>2</sup> Disponível em: <https://github.com/dmlc/xgboost>

## 3.6 Resultados

Para avaliar os meta-modelos propostos, foram realizadas 30 repetições de *10-fold Cross Validation* (10fCV), tanto para os meta-classificadores quanto para os meta-regressores. Para os meta-classificadores, as partições de cada 10fCV são estratificadas. Os resultados obtidos foram sumarizados no quadro 5 para a família de meta-características “elaboradas” e no quadro 6 para a família de meta-características “artificial”.

A figura 7 ilustra a relevância (*feature importance*) das meta-características de ambas as famílias. Estes valores foram extraídos utilizando o conjunto de meta-dados de 500 episódios de treinamento e um meta-classificador utilizando os hiper-parâmetros do quadro 4 (para 500 episódios de treinamento) exceto por “*n\_estimators*”, fixado em 3000 nesta operação e sem *early stopping*.

Observar-se que para ambas as famílias de meta-características, os meta-modelos relacionados aos meta-dados de 500 episódios de treinamento apresentam resultados expressivos, evidentemente superiores a performance base (proporção de classe majoritária para meta-classificadores, e desvio padrão do rótulo para meta-regressores). Para os meta-classificadores da família de meta-características “elaboradas”, a maior margem entre a performance base e a acurácia de teste correspondem ao algoritmo base Q-Learning, com  $15.4\% \pm 3\%$  de ganho. Entretanto, não há evidência de um ganho expressivo para prever a convergência do algoritmo Double Q-Learning em relação a performance base. Todos os meta-regressores possuem performance levemente superior a performance base. Em relação as meta-características da família “artificial”, os resultados dos meta-modelos de 500 episódios de treinamento são análogos, porém com ganhos em relação aos valores de referência menos expressivos.

Um cenário similar ao descrito acima ocorre para os meta-modelos treinados nos meta-dados de 1000 episódios de treinamento. Para a família de meta-características “elaboradas”, os meta-classificadores podem prever a convergência do algoritmo Double Q-Learning com superioridade em relação ao valor de referência. Acredita-se que a razão disso esteja em função do melhor balanceamento de classes neste cenário em relação ao cenário anterior. No geral, as margens entre a performance dos meta-modelos e os valores de referência são menores que as margens do cenário anterior. É possível que o menor número de meta-exemplos seja a principal causa desta discrepância. Para a família de meta-características “artificial”, apenas o meta-classificador para o algoritmo Double Q-Learning apresenta um ganho de performance estatisticamente superior a linha de base.

Para os meta-modelos treinados em meta-dados de 3000 episódios de treinamento, não há evidências de ganhos significativos para nenhum algoritmo base nem

Quadro 5 – Resultados dos experimentos para a família de meta-características “elaboradas”. A coluna da esquerda indica o número de episódios de treinamento para extrair os meta-dados. O símbolo  $\sigma$  representa o desvio-padrão da quantidade que ele está associado. Em particular,  $\sigma_y$  representa o desvio-padrão do meta-atributo alvo. “ACC” significa “acurácia” e “AUC” significa “Area Under the Curve” da curva “ROC” (“Receiver Operating Characteristic”).

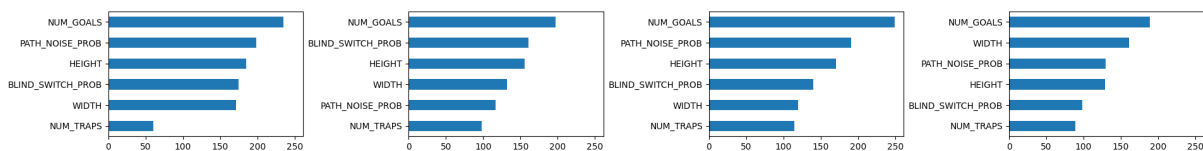
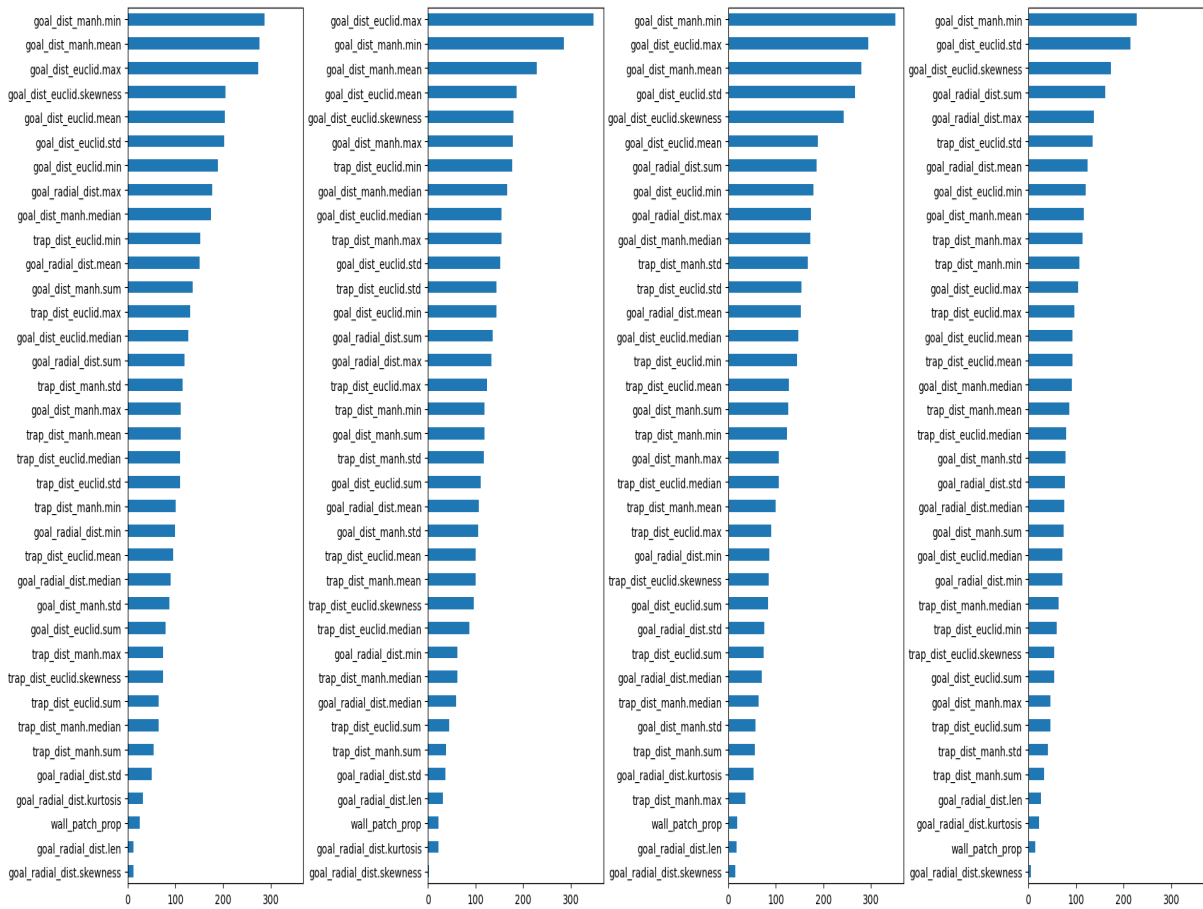
Meta-características “Elaboradas” ( $30 \times 10$ -fold Cross Validation)					
500	Meta-classificação				
	Algoritmo	SARSA	MC Control	Q-Learning	D. Q-Learning
	ACC <sub>maioria</sub>	0.549	0.564	0.525	0.738
	ACC <sub>treino</sub> $\pm \sigma$	$0.736 \pm 0.005$	$0.759 \pm 0.004$	$0.768 \pm 0.005$	$0.776 \pm 0.003$
	ACC <sub>teste</sub> $\pm \sigma$	$0.647 \pm 0.033$	$0.680 \pm 0.029$	$0.679 \pm 0.030$	$0.751 \pm 0.015$
	AUC <sub>treino</sub> $\pm \sigma$	$0.823 \pm 0.003$	$0.841 \pm 0.002$	$0.856 \pm 0.003$	$0.804 \pm 0.003$
	AUC <sub>teste</sub> $\pm \sigma$	$0.702 \pm 0.033$	$0.740 \pm 0.030$	$0.750 \pm 0.032$	$0.677 \pm 0.043$
	Meta-regressão				
	Algoritmo	SARSA	MC Control	Q-Learning	D. Q-Learning
	$\sigma_y$	1.491	1.318	1.496	1.318
	RMSE <sub>treino</sub> $\pm \sigma$	$1.229 \pm 0.005$	$1.105 \pm 0.004$	$1.097 \pm 0.005$	$1.133 \pm 0.006$
	RMSE <sub>teste</sub> $\pm \sigma$	$1.401 \pm 0.035$	$1.227 \pm 0.038$	$1.347 \pm 0.038$	$1.256 \pm 0.055$
1000	Meta-classificação				
	Algoritmo	SARSA	MC Control	Q-Learning	D. Q-Learning
	ACC <sub>maioria</sub>	0.673	0.585	0.673	0.568
	ACC <sub>treino</sub> $\pm \sigma$	$0.786 \pm 0.007$	$0.763 \pm 0.007$	$0.797 \pm 0.007$	$0.741 \pm 0.008$
	ACC <sub>teste</sub> $\pm \sigma$	$0.712 \pm 0.045$	$0.660 \pm 0.052$	$0.732 \pm 0.045$	$0.633 \pm 0.047$
	AUC <sub>treino</sub> $\pm \sigma$	$0.853 \pm 0.005$	$0.836 \pm 0.004$	$0.866 \pm 0.004$	$0.822 \pm 0.005$
	AUC <sub>teste</sub> $\pm \sigma$	$0.731 \pm 0.062$	$0.700 \pm 0.056$	$0.757 \pm 0.057$	$0.664 \pm 0.061$
	Meta-regressão				
	Algoritmo	SARSA	MC Control	Q-Learning	D. Q-Learning
	$\sigma_y$	1.405	1.304	1.406	1.485
	RMSE <sub>treino</sub> $\pm \sigma$	$1.024 \pm 0.008$	$1.035 \pm 0.007$	$0.927 \pm 0.008$	$1.201 \pm 0.007$
	RMSE <sub>teste</sub> $\pm \sigma$	$1.295 \pm 0.070$	$1.228 \pm 0.061$	$1.252 \pm 0.079$	$1.430 \pm 0.057$
3000	Meta-classificação				
	Algoritmo	SARSA	MC Control	Q-Learning	D. Q-Learning
	ACC <sub>maioria</sub>	0.936	0.660	0.924	0.712
	ACC <sub>treino</sub> $\pm \sigma$	$0.936 \pm 0.002$	$0.774 \pm 0.008$	$0.924 \pm 0.001$	$0.744 \pm 0.007$
	ACC <sub>teste</sub> $\pm \sigma$	$0.936 \pm 0.020$	$0.709 \pm 0.060$	$0.924 \pm 0.012$	$0.721 \pm 0.031$
	AUC <sub>treino</sub> $\pm \sigma$	$0.541 \pm 0.093$	$0.874 \pm 0.008$	$0.650 \pm 0.128$	$0.822 \pm 0.008$
	AUC <sub>teste</sub> $\pm \sigma$	$0.529 \pm 0.096$	$0.676 \pm 0.105$	$0.624 \pm 0.154$	$0.669 \pm 0.113$
	Meta-regressão				
	Algoritmo	SARSA	MC Control	Q-Learning	D. Q-Learning
	$\sigma_y$	0.708	1.172	0.795	1.359
	RMSE <sub>treino</sub> $\pm \sigma$	$0.618 \pm 0.022$	$1.021 \pm 0.014$	$0.609 \pm 0.017$	$1.027 \pm 0.015$
	RMSE <sub>teste</sub> $\pm \sigma$	$0.667 \pm 0.251$	$1.156 \pm 0.134$	$0.745 \pm 0.238$	$1.316 \pm 0.139$



Quadro 6 – Resultados dos experimentos para a família de meta-características “artificial”. A coluna da esquerda indica o número de episódios de treinamento para extrair os meta-dados. O símbolo  $\sigma$  representa o desvio-padrão da quantidade que ele está associado. Em particular,  $\sigma_y$  representa o desvio-padrão do meta-atributo alvo. “ACC” significa “acurácia” e “AUC” significa “Area Under the Curve” da curva “ROC” (“Receiver Operating Characteristic”).

Meta-características “Artificial” ( $30 \times 10$ -fold Cross Validation)					
500	Meta-classificação				
	Algoritmo	SARSA	MC Control	Q-Learning	D. Q-Learning
	ACC <sub>maioria</sub>	0.549	0.564	0.525	0.738
	ACC <sub>treino</sub> $\pm \sigma$	0.651 $\pm$ 0.004	0.647 $\pm$ 0.005	0.650 $\pm$ 0.005	0.742 $\pm$ 0.002
	ACC <sub>teste</sub> $\pm \sigma$	0.632 $\pm$ 0.030	0.634 $\pm$ 0.032	0.633 $\pm$ 0.032	0.739 $\pm$ 0.007
	AUC <sub>treino</sub> $\pm \sigma$	0.706 $\pm$ 0.004	0.687 $\pm$ 0.004	0.714 $\pm$ 0.004	0.688 $\pm$ 0.004
	AUC <sub>teste</sub> $\pm \sigma$	0.682 $\pm$ 0.032	0.665 $\pm$ 0.035	0.692 $\pm$ 0.036	0.658 $\pm$ 0.041
	Meta-regressão				
	Algoritmo	SARSA	MC Control	Q-Learning	D. Q-Learning
	$\sigma_y$	1.491	1.318	1.496	1.318
1000	RMSE <sub>treino</sub> $\pm \sigma$	1.381 $\pm$ 0.004	1.249 $\pm$ 0.003	1.379 $\pm$ 0.004	1.254 $\pm$ 0.006
	RMSE <sub>teste</sub> $\pm \sigma$	1.423 $\pm$ 0.027	1.277 $\pm$ 0.029	1.418 $\pm$ 0.029	1.284 $\pm$ 0.049
	Meta-classificação				
	Algoritmo	SARSA	MC Control	Q-Learning	D. Q-Learning
	ACC <sub>maioria</sub>	0.673	0.585	0.673	0.568
	ACC <sub>treino</sub> $\pm \sigma$	0.734 $\pm$ 0.008	0.645 $\pm$ 0.008	0.726 $\pm$ 0.008	0.687 $\pm$ 0.007
	ACC <sub>teste</sub> $\pm \sigma$	0.722 $\pm$ 0.040	0.619 $\pm$ 0.048	0.710 $\pm$ 0.042	0.673 $\pm$ 0.048
	AUC <sub>treino</sub> $\pm \sigma$	0.783 $\pm$ 0.006	0.696 $\pm$ 0.006	0.771 $\pm$ 0.006	0.731 $\pm$ 0.006
	AUC <sub>teste</sub> $\pm \sigma$	0.754 $\pm$ 0.052	0.652 $\pm$ 0.057	0.744 $\pm$ 0.056	0.702 $\pm$ 0.059
	Meta-regressão				
3000	Algoritmo	SARSA	MC Control	Q-Learning	D. Q-Learning
	$\sigma_y$	1.405	1.304	1.406	1.485
	RMSE <sub>treino</sub> $\pm \sigma$	1.183 $\pm$ 0.008	1.188 $\pm$ 0.007	1.208 $\pm$ 0.008	1.296 $\pm$ 0.007
	RMSE <sub>teste</sub> $\pm \sigma$	1.284 $\pm$ 0.068	1.259 $\pm$ 0.054	1.298 $\pm$ 0.069	1.392 $\pm$ 0.054
	Meta-classificação				
	Algoritmo	SARSA	MC Control	Q-Learning	D. Q-Learning
	ACC <sub>maioria</sub>	0.936	0.660	0.924	0.712
	ACC <sub>treino</sub> $\pm \sigma$	0.936 $\pm$ 0.002	0.663 $\pm$ 0.005	0.924 $\pm$ 0.001	0.714 $\pm$ 0.005
	ACC <sub>teste</sub> $\pm \sigma$	0.936 $\pm$ 0.020	0.659 $\pm$ 0.022	0.924 $\pm$ 0.012	0.707 $\pm$ 0.023
	AUC <sub>treino</sub> $\pm \sigma$	0.500 $\pm$ 0.000	0.703 $\pm$ 0.011	0.503 $\pm$ 0.022	0.737 $\pm$ 0.014
	AUC <sub>teste</sub> $\pm \sigma$	0.500 $\pm$ 0.000	0.589 $\pm$ 0.119	0.503 $\pm$ 0.025	0.648 $\pm$ 0.116
3000	Meta-regressão				
	Algoritmo	SARSA	MC Control	Q-Learning	D. Q-Learning
	$\sigma_y$	0.708	1.172	0.795	1.359
	RMSE <sub>treino</sub> $\pm \sigma$	0.624 $\pm$ 0.021	0.978 $\pm$ 0.014	0.682 $\pm$ 0.020	1.145 $\pm$ 0.015
	RMSE <sub>teste</sub> $\pm \sigma$	0.647 $\pm$ 0.252	1.142 $\pm$ 0.123	0.728 $\pm$ 0.241	1.306 $\pm$ 0.131

Figura 7 – *Feature importance* das meta-características de ambas as famílias, extraídas utilizando um meta-classificador nos meta-dados de 500 episódios de treinamento.



no cenário de meta-classificação e nem em tarefas de meta-regressão, para ambas as famílias de meta-características. Atribui-se este resultado ao desbalanceamento de classes (severo em alguns casos) combinado com o número relativamente baixo de meta-exemplos.

No apêndice A é realizada uma breve análise sobre como a presença de ruídos nas meta-características impacta na performance dos meta-classificadores.

---

## CONCLUSÃO

---

Este trabalho procurou seguir uma perspectiva alternativa de meta-aprendizado com aprendizado de máquina por reforço encontrada na literatura, que geralmente utiliza redes neurais recorrentes, e mostrou que é possível utilizar conhecimento acumulado sobre uma distribuição de problemas similares para auxiliar o uso de aprendizado por reforço. Em particular, utilizando *Gridworlds* e uma seleção de meta-características simples extraídas diretamente do meio, um sistema de recomendação de algoritmos foi capaz de deduzir, com performance acima da linha de base, se um algoritmo de aprendizado por reforço irá ou não convergir dado um limite máximo de episódios, potencialmente poupando tempo e custo computacional.

Reconhece-se que o experimento realizado neste trabalho é bastante simples se comparado a problemas reais. Entretanto, este trabalho busca inspirar trabalhos prospectivos a estudar outras formas de extrair meta-características em diversos outros tipos de meios e, quem sabe, meta-características gerais que possuem sentido semântico em múltiplas distribuições de problemas.

Note que a definição de meta-aprendizado neste texto não tem como objetivo competir com o meta-aprendizado utilizando redes neurais recorrentes, mas sim oferecer um novo tipo de ferramenta que possa ser usada em conjunto com o estado-da-arte. Portanto, este texto também procura inspirar trabalhos futuros a estudar o impacto da união de ambas as perspectivas de meta-aprendizado: a adotada neste texto e o meta-aprendizado por reforço com redes neurais recorrentes.



## REFERÊNCIAS

---

BELLMAN, R. **Dynamic Programming**. 1. ed. Princeton, NJ, USA: Princeton University Press, 1957. Disponível em: <<http://books.google.com/books?id=fyVtp3EMxasC&pg=PR5&dq=dynamic+programming+richard+e+bellman&client=firefox-a#v=onepage&q=dynamic%20programming%20richard%20e%20bellman&f=false>>.

Citado na página 23.

BRAZDIL, P.; GIRAUD-CARRIER, C.; SOARES, C.; VILALTA, R. **Metalearning - Applications to Data Mining**. [S.l.: s.n.], 2009. ISBN 978-3-540-73262-4. Citado 5 vezes nas páginas 15, 16, 26, 28 e 29.

CHEN, T.; GUESTRIN, C. Xgboost. **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**, ACM, Aug 2016. Disponível em: <<http://dx.doi.org/10.1145/2939672.2939785>>. Citado na página 34.

CHO, K.; MERRIENBOER, B. van; GULCEHRE, C.; BAHDANAU, D.; BOUGARES, F.; SCHWENK, H.; BENGIO, Y. **Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation**. 2014. Citado na página 25.

COTTERN, E.; CONWELL, P. R. Fixed-weight networks can learn. In: **1990 IJCNN International Joint Conference on Neural Networks**. [S.l.: s.n.], 1990. p. 553–559 vol.3. Citado na página 24.

DUAN, Y.; SCHULMAN, J.; CHEN, X.; BARTLETT, P. L.; SUTSKEVER, I.; ABBEEL, P. **RL<sup>2</sup>: Fast Reinforcement Learning via Slow Reinforcement Learning**. 2016. Citado 2 vezes nas páginas 15 e 25.

FEURER, M.; KLEIN, A.; EGGENSPERGER, K.; SPRINGENBERG, J.; BLUM, M.; HUTTER, F. Efficient and robust automated machine learning. In: CORTES, C.; LAWRENCE, N. D.; LEE, D. D.; SUGIYAMA, M.; GARNETT, R. (Ed.). **Advances in Neural Information Processing Systems 28**. Curran Associates, Inc., 2015. p. 2962–2970. Disponível em: <<http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf>>. Citado 2 vezes nas páginas 16 e 28.

GRAVES, A.; WAYNE, G.; DANIHELKA, I. **Neural Turing Machines**. 2014. Citado na página 25.

HASSELT, H. V. Double q-learning. In: LAFFERTY, J. D.; WILLIAMS, C. K. I.; SHAWE-TAYLOR, J.; ZEMEL, R. S.; CULOTTA, A. (Ed.). **Advances in Neural Information Processing Systems 23**. Curran Associates, Inc., 2010. p. 2613–2621. Disponível em: <<http://papers.nips.cc/paper/3964-double-q-learning.pdf>>. Citado na página 23.

HASSELT, H. van; GUEZ, A.; SILVER, D. **Deep Reinforcement Learning with Double Q-learning**. 2015. Citado na página 23.

HINTON, G. E.; PLAUT, D. C. Using fast weights to deblur old memories. In: **IN PROCEEDINGS OF THE 9TH ANNUAL CONFERENCE OF THE COGNITIVE SCIENCE SOCIETY**. [S.l.]: Erlbaum, 1987. p. 177–186. Citado na página 24.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Comput.**, MIT Press, Cambridge, MA, USA, v. 9, n. 8, p. 1735–1780, nov. 1997. ISSN 0899-7667. Disponível em: <<https://doi.org/10.1162/neco.1997.9.8.1735>>. Citado na página 25.

HOCHREITER, S.; YOUNGER, A. S.; CONWELL, P. R. Learning to learn using gradient descent. In: **IN LECTURE NOTES ON COMP. SCI. 2130, PROC. INTL. CONF. ON ARTI NEURAL NETWORKS (ICANN-2001)**. [S.l.]: Springer, 2001. p. 87–94. Citado na página 25.

HOSPEDALES, T.; ANTONIOU, A.; MICAELLI, P.; STORKEY, A. **Meta-Learning in Neural Networks: A Survey**. 2020. Citado 2 vezes nas páginas 15 e 24.

HOWARD, R. A. **Dynamic Programming and Markov Processes**. Cambridge, MA: MIT Press, 1960. Citado na página 23.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. (Ed.). **Automated Machine Learning: Methods, Systems, Challenges**. [S.l.]: Springer, 2018. In press, available at <http://automl.org/book>. Citado 2 vezes nas páginas 16 e 28.

LAKE, B. M.; ULLMAN, T. D.; TENENBAUM, J. B.; GERSHMAN, S. J. **Building Machines That Learn and Think Like People**. 2016. Citado na página 15.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, v. 521, p. 436–44, 05 2015. Citado na página 24.

LEMKE, C.; BUDKA, M.; GABRYS, B. Metalearning: A survey of trends and technologies. **Artif. Intell. Rev.**, Kluwer Academic Publishers, USA, v. 44, n. 1, p. 117–130, jun. 2015. ISSN 0269-2821. Disponível em: <<https://doi.org/10.1007/s10462-013-9406-y>>. Citado 2 vezes nas páginas 15 e 24.

MITCHELL, T. M. **Machine Learning**. New York: McGraw-Hill, 1997. ISBN 978-0-07-042807-2. Citado 2 vezes nas páginas 15 e 24.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERS-TRA, D.; RIEDMILLER, M. **Playing Atari with Deep Reinforcement Learning**. 2013. Citado 3 vezes nas páginas 15, 16 e 23.

PROKHOROV, D.; FELDKARNP, L.; TYUKIN, I. Adaptive behavior with fixed weights in rnn: An overview. In: . [S.l.: s.n.], 2002. v. 3, p. 2018 – 2022. ISBN 0-7803-7278-6. Citado na página 25.

PUTERMAN, M. L. **Markov Decision Processes: Discrete Stochastic Dynamic Programming**. New York: John Wiley & Sons, 1994. Disponível em: <<http://dx.doi.org/10.1002/9780470316887>>. Citado na página 23.

RIVOLLI, A.; GARCIA, L. P. F.; SOARES, C.; VANSCHOREN, J.; CARVALHO, A. C. P. L. F. de. **Characterizing classification datasets: a study of meta-features for meta-learning**. 2019. Citado 3 vezes nas páginas 27, 28 e 33.

RUMMERY, G.; NIRANJAN, M. On-line q-learning using connectionist systems. **Technical Report CUED/F-INFENG/TR 166**, 11 1994. Citado na página 23.

SANTORO, A.; BARTUNOV, S.; BOTVINICK, M.; WIERSTRA, D.; LILLICRAP, T. Meta-learning with memory-augmented neural networks. In: BALCAN, M. F.; WEINBERGER, K. Q. (Ed.). New York, New York, USA: PMLR, 2016. (Proceedings of Machine Learning Research, v. 48), p. 1842–1850. Disponível em: <<http://proceedings.mlr.press/v48/santoro16.html>>. Citado na página 25.

SCHMIDHUBER, J. **Evolutionary Principles in Self-Referential Learning. On Learning now to Learn: The Meta-Meta-Meta...-Hook**. Dissertação (Diploma Thesis) — Technische Universitat Munchen, Germany, 14 maio 1987. Disponível em: <<http://www.idsia.ch/~juergen/diploma.html>>. Citado na página 24.

SCHMIDHUBER, J.; ZHAO, J.; WIERING, M. **Simple Principles Of Metalearning**. [S.l.], 1996. Citado 2 vezes nas páginas 15 e 25.

SCHWEIGHOFER, N.; DOYA, K. Meta-learning in reinforcement learning. **Neural Networks**, v. 16, n. 1, p. 5 – 9, 2003. ISSN 0893-6080. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0893608002002289>>. Citado 2 vezes nas páginas 15 e 25.

SILVER, D.; HUANG, A.; MADDISON, C.; GUEZ, A.; SIFRE, L.; DRIESSCHE, G.; SCHRITTWIESER, J.; ANTONOGLOU, I.; PANNEERSHELVAM, V.; LANCTOT, M.; DIELEMAN, S.; GREWE, D.; NHAM, J.; KALCHBRENNER, N.; SUTSKEVER, I.; LILLICRAP, T.; LEACH, M.; KAVUKCUOGLU, K.; GRAEPEL, T.; HASSABIS, D. Mastering the game of go with deep neural networks and tree search. **Nature**, v. 529, p. 484–489, 01 2016. Citado 2 vezes nas páginas 15 e 16.

SUTTON, R. Generalization in reinforcement learning: Successful examples using sparse coarse coding. 08 1996. Citado na página 23.

SUTTON, R. S.; BARTO, A. G. Reinforcement learning: An introduction. **Trans. Neur. Netw.**, IEEE Press, v. 9, n. 5, p. 1054, set. 1998. ISSN 1045-9227. Disponível em: <<https://doi.org/10.1109/TNN.1998.712192>>. Citado na página 21.

\_\_\_\_\_. **Reinforcement Learning: An Introduction (2nd ed.)**. Cambridge, MA, USA: A Bradford Book, 2018. ISBN 0262039249. Citado na página 24.

THORNTON, C.; HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In: **Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: Association for Computing Machinery, 2013. (KDD '13), p. 847–855. ISBN 9781450321747. Disponível em: <<https://doi.org/10.1145/2487575.2487629>>. Citado 2 vezes nas páginas 16 e 28.

THRUN, S.; PRATT, L. Learning to learn: Introduction and overview. In: \_\_\_\_\_. **Learning to Learn**. Boston, MA: Springer US, 1998. p. 3–17. ISBN 978-1-4615-5529-2. Disponível em: <[https://doi.org/10.1007/978-1-4615-5529-2\\_1](https://doi.org/10.1007/978-1-4615-5529-2_1)>. Citado 4 vezes nas páginas 15, 24, 25 e 26.

- VILALTA, R.; GIRAUD-CARRIER, C.; BRAZDIL, P.; SOARES, C. Using meta-learning to support data mining. **International Journal of Computer Science & Applications**, v. 1, 01 2004. Citado 5 vezes nas páginas 15, 16, 26, 28 e 29.
- WANG, J. X.; KURTH-NELSON, Z.; TIRUMALA, D.; SOYER, H.; LEIBO, J. Z.; MUNOS, R.; BLUNDELL, C.; KUMARAN, D.; BOTVINICK, M. **Learning to reinforcement learn**. 2017. Citado 2 vezes nas páginas 15 e 25.
- WANG, Z.; SCHAUL, T.; HESSEL, M.; HASSELT, H. van; LANCTOT, M.; FREITAS, N. de. **Dueling Network Architectures for Deep Reinforcement Learning**. 2016. Citado na página 24.
- WATKINS, C. Learning from delayed rewards. 01 1989. Citado na página 23.
- WENG, L. Policy gradient algorithms. **lilianweng.github.io/lil-log**, 2018. Disponível em: <<https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html>>. Citado na página 23.
- WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In: **Machine Learning**. [S.l.: s.n.], 1992. p. 229–256. Citado na página 23.
- YARATS, D.; ZHANG, A.; KOSTRIKOV, I.; AMOS, B.; PINEAU, J.; FERGUS, R. **Improving Sample Efficiency in Model-Free Reinforcement Learning from Images**. 2020. Citado na página 16.
- YOUNGER, A. S.; CONWELL, P. R.; COTTER, N. E. Fixed-weight on-line learning. **IEEE Transactions on Neural Networks**, v. 10, n. 2, p. 272–283, 1999. Citado na página 24.



---

APÊNDICE A

## EFEITO DE RUÍDOS NAS META-CARACTERÍSTICAS

---

Este apêndice descreve brevemente o impacto na performance de meta-classificadores com a presença de ruídos gaussianos nas meta-características. Foram utilizados somente as meta-características da família “elaboradas” e com 500 episódios de treinamento para a análise.

Todos os resultados neste apêndice foram coletados usando  $30 \times 10$ -fold *Cross Validation* com partições estratificadas e meta-classificadores induzidos pelo algoritmo *XGBoost* com os hiper-parâmetros do quadro 4 (500 episódios de treinamento).

Os ruídos inseridos foram amostrados das distribuições gaussianas  $\mathcal{N}(0, (p_t \sigma_j)^2)$ , em que  $\sigma_j$  é o desvio-padrão do  $j$ -ésimo atributo, e  $p_t$  é um fator de proporção fixo para todos os atributos no tempo  $t$ . Em outras palavras, a cada iteração  $t$ , ruídos proporcionais ao desvio-padrão de cada atributo foram amostrados de forma independente e inseridos nos respectivos atributos. O fator de proporção foi incrementado linearmente a cada iteração  $t$  entre 0 (sem ruído) até 10 (cada atributo recebe ruídos gaussianos com dez vezes seu próprio desvio-padrão).

A figura 8 mostra o decaimento da AUC de teste para todos os algoritmos sobrepostos. As figuras 9 e 10 exibem os mesmos resultados da figura anterior, porém para cada algoritmo em gráficos distintos. Como pode ser observado nas figuras, a performance dos meta-classificadores decaem rapidamente com a presença de ruídos. Entretanto, nota-se que, para todos os algoritmos base, a AUC parece estabilizar acima da linha de base ( $AUC = 0.5$ ).

Figura 8 – Impacto na performance (AUC) no conjunto de teste para todos os algoritmos base sobrepostos. A linha de referência é AUC = 0.5.

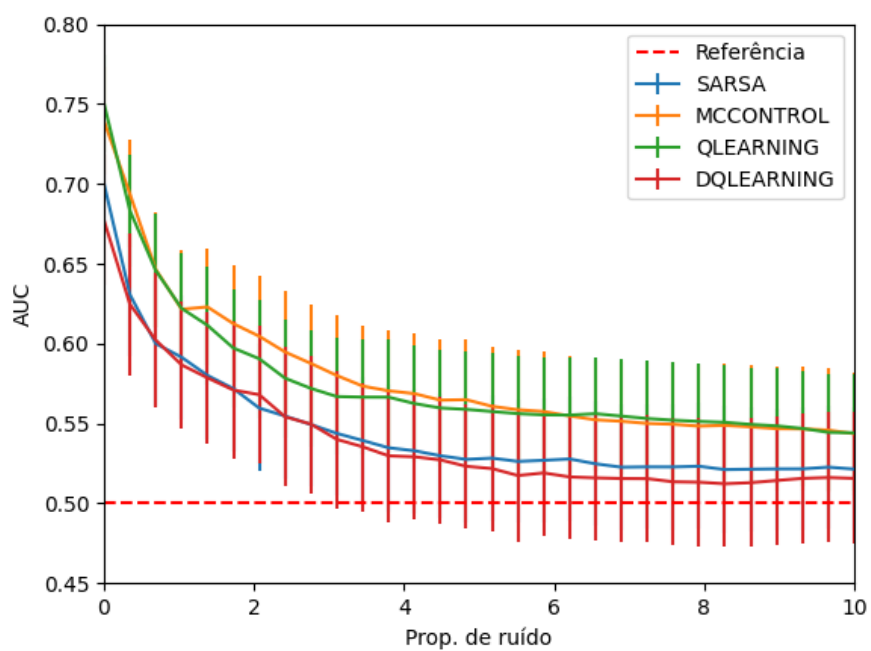
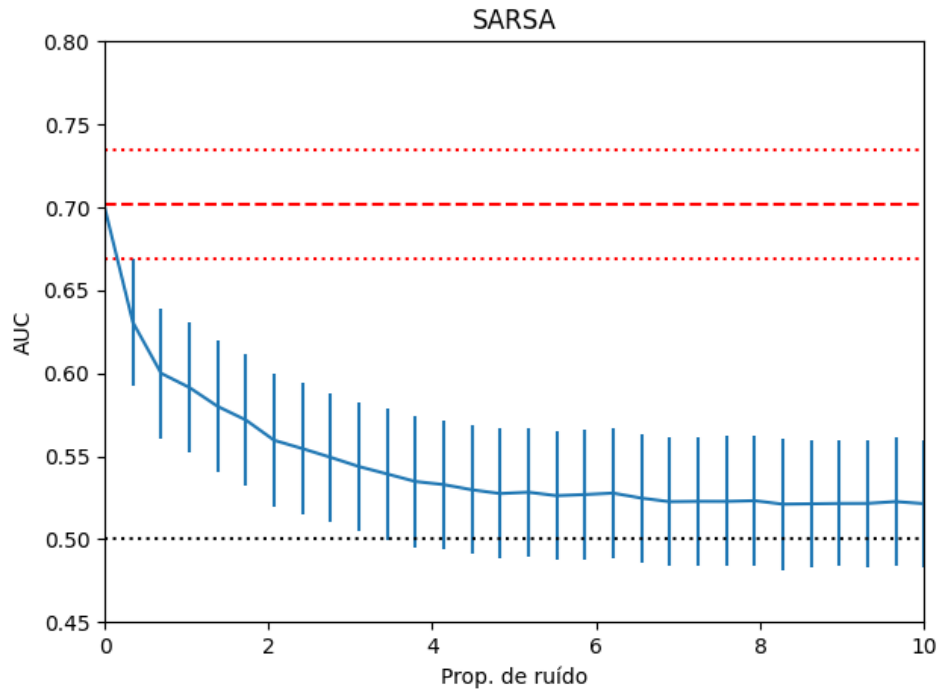
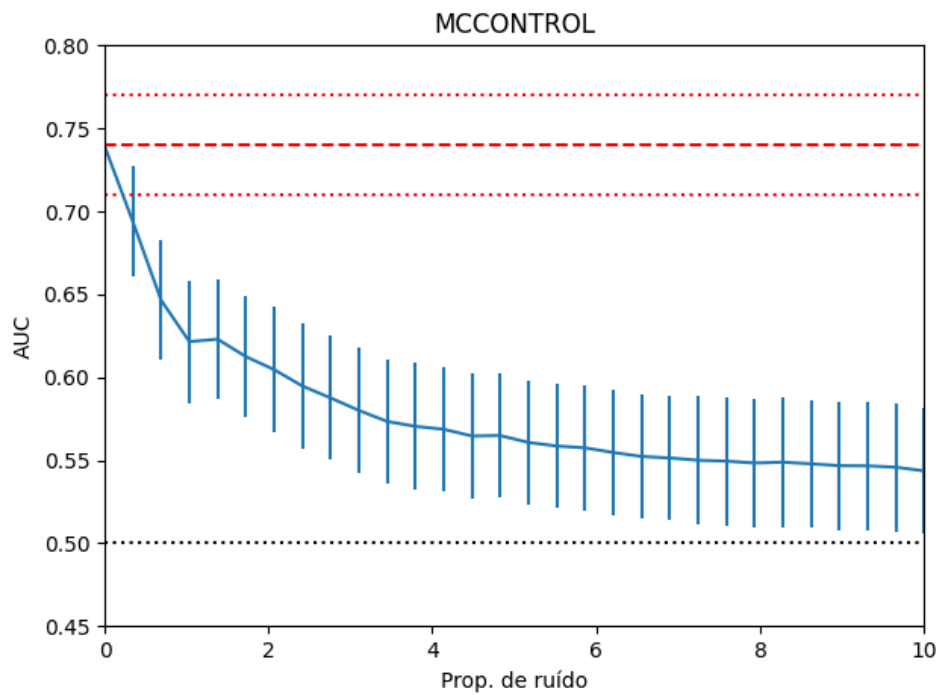


Figura 9 – Impacto na performance de teste (AUC) no meta-classificador treinado com meta-características ruidosas para deduzir a convergência dos algoritmos SARSA (a) e Monte Carlo Control (b). As linhas tracejadas superiores (vermelha) marcam a AUC de teste sem inserção de ruídos dos respectivos algoritmos. As linhas pontilhadas superior e inferior marcam o intervalo de confiança (desvio-padrão) da AUC de referência.

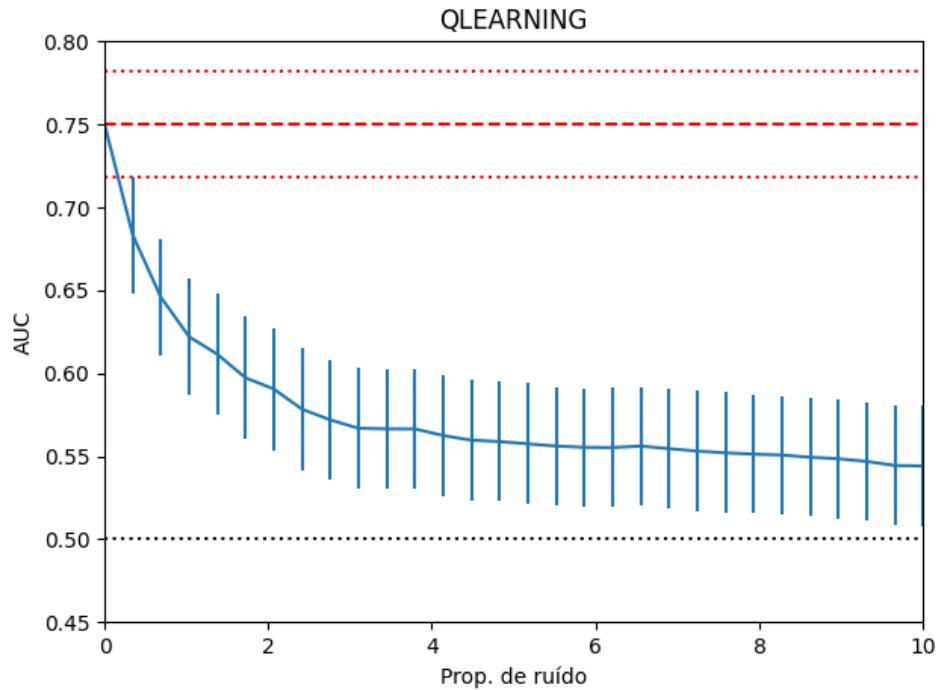


(a) Algoritmo SARSA

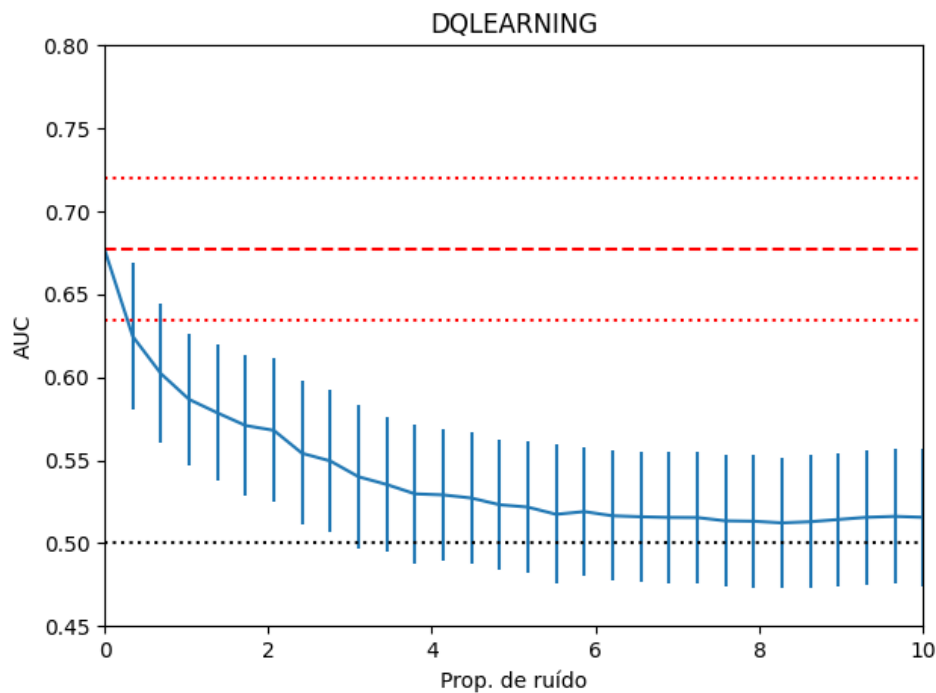


(b) Algoritmo Monte Carlo Control

Figura 10 – Impacto na performance de teste (AUC) no meta-classificador treinado com meta-características ruidosas para deduzir a convergência dos algoritmos Q-Learning (a) e Double Q-Learning (b). As linhas tracejadas superiores (vermelha) marcam a AUC de teste sem inserção de ruídos dos respectivos algoritmos. As linhas pontilhadas superior e inferior marcam o intervalo de confiança (desvio-padrão) da AUC de referência.



(a) Algoritmo Q-Learning



(b) Algoritmo Double Q-Learning