Aprendizado por reforço em lote: um estudo de caso para o problema de tomada de decisão em processos de venda

Denis Antonio Lacerda

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Ciência da Computação
Orientadora: Prof^{a}. Dr^{a}. Leliane Nunes de Barros

São Paulo, Dezembro de 2013

Aprendizado por reforço em lote: um estudo de caso para o problema de tomada de decisão em processos de venda

Esta versão da dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 12/12/2013. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Prof^{a}. Dr^{a}. Leliane Nunes de Barros (orientadora) - IME-USP
- Prof. Dr. João Eduardo Ferreira - IME-USP
- Prof^{a}. Dr^{a}. Anna Helena Reali - EP-USP

Agradecimentos

Agradeço primeiramente a Deus por ter me dado saúde e força para superar todas as dificuldades.

Ao IME pelo ambiente criativo que proporciona, e a todo corpo docente por tudo que me ensinaram. Em especial à minha orientadora Leliane pelo apoio e pelo emprenho dedicado à elaboração deste trabalho, e ao professor Alfredo por todos os conselhos.

Á minha mãe Helena Lacerda pelo amor, incentivo e apoio incondicional.

Ao meu chefe Pietro Biselli pelo incentivo.

E a todos que direta e indiretamente fizeram parte de minha formação, o meu muito obrigado.

Resumo

Planejamento Probabilístico estuda os problemas de tomada de decisão sequencial de um agente, em que as ações possuem efeitos probabilísticos, modelados como um processo de decisão markoviano (*Markov Decision Process - MDP*). Dadas a função de transição de estados probabilística e os valores de recompensa das ações, é possível determinar uma política de ações (i.e., um mapeamento entre estado do ambiente e ações do agente) que maximiza a recompensa esperada acumulada (ou minimiza o custo esperado acumulado) pela execução de uma sequência de ações. Nos casos em que o modelo MDP não é completamente conhecido, a melhor política deve ser aprendida através da interação do agente com o ambiente real. Este processo é chamado de aprendizado por reforço. Porém, nas aplicações em que não é permitido realizar experiências no ambiente real, por exemplo, operações de venda, é possível realizar o aprendizado por reforço sobre uma amostra de experiências passadas, processo chamado de aprendizado por reforço em lote (*Batch Reinforcement Learning*). Neste trabalho, estudamos técnicas de aprendizado por reforço em lote usando um histórico de interações passadas, armazenadas em um banco de dados de processos, e propomos algumas formas de melhorar os algoritmos existentes. Como um estudo de caso, aplicamos esta técnica no aprendizado de políticas para o processo de venda de impressoras de grande formato, cujo objetivo é a construção de um sistema de recomendação de ações para vendedores iniciantes.
Palavras-chave: Planejamento Probabilístico, Processo de Decisão Markoviano, Aprendizado por Reforço em Lote, Aprendizado de Processos de Venda.

Abstract

Probabilistic planning studies the problems of sequential decision-making of an agent, in which actions have probabilistic effects, and can be modeled as a Markov decision process (MDP). Given the probabilities and reward values of each action, it is possible to determine an action policy (in other words, a mapping between the state of the environment and the agent's actions) that maximizes the expected reward accumulated by executing a sequence of actions. In cases where the MDP model is not completely known, the best policy needs to be learned through the interaction of the agent in the real environment. This process is called *reinforcement learning*. However, in applications where it is not allowed to perform experiments in the real environment, for example, sales process, it is possible to perform the reinforcement learning using a sample of past experiences. This process is called Batch Reinforcement Learning. In this work, we study techniques of *batch reinforcement learning* (BRL), in which learning is done using a history of past interactions, stored in a processes database. As a case study, we apply this technique for learning policies in the sales process for large format printers, whose goal is to build a action recommendation system for beginners sellers.
Keywords: Probabilistic Planning, Markov Decision Process, Batch Reinforcement Learning, Sales Process Learning.

Sumário

Lista de Abreviaturas **ix**

Lista de Figuras **xi**

Lista de Tabelas **xiii**

1 Introdução **1**
1.1 Motivação 2
1.2 Proposta 3
1.3 Contribuições desse trabalho 4
1.4 Organização do texto 4

2 Planejamento probabilístico **7**
2.1 Processo de decisão markoviano 7
2.1.1 Política ótima 8
2.1.2 Algoritmos para resolver MDPs 9

3 Aprendizado por Reforço **13**
3.1 Q-Learning 15
3.2 SARSA 16
3.3 Dyna-Q 17
3.4 Conclusão 17

4 Aprendizado por reforço em lote (Batch Reinforcement Learning - BRL) **19**
4.1 Fundamentos dos Algoritmos de BRL 20
4.2 Algoritmos 22
4.2.1 BRL Com Repetição de Experiências 23
4.2.2 BRL Ajustado 23
4.3 Aprendizado por reforço em lote crescente 24
4.4 Classificação dos algoritmos 26

5 Estudo de caso: Modelando o Problema do Vendedor de Impressoras **27**
5.1 Processos de Venda 27
5.2 Venda de impressoras Plotters e Sistema de Acompanhamento de Negociações (Forecast) 28

5.3 Modelando o problema do vendedor de impressoras como um problema de aprendizado por reforço em lote . . . 31
5.3.1 Identificando Estados . . . 31
5.3.2 Identificando Ações . . . 32
5.3.3 Definição da função recompensa . . . 34

6 Aprendizado por reforço em lote para o Problema do Vendedor de Impressoras 37
6.1 Reconstruindo o processo: geração do conjunto de experiências . . . 37
6.2 Limpeza dos dados . . . 39
6.3 Dinâmica do ambiente . . . 39
6.4 Resolvendo o problema: melhorias propostas no algoritmo de aprendizado por reforço em lote . . . 39
6.4.1 Aprendizado por reforço em lote crescente sem exploração . . . 40
6.4.2 Aprendizado por reforço em lote com regressão por processo . . . 41
6.4.3 Teste de convergência . . . 42

7 Resultados Experimentais 43
7.1 BRL crescente (sem exploração) sem repetição . . . 43
7.2 BRL sem repetição vs BRL com Repetição . . . 43
7.3 Ordenação do lote de experiências . . . 44
7.4 K-Fold: Verificando a qualidade da amostra . . . 44
7.5 Política dos vendedores versus política aprendida . . . 46

8 Conclusão 47

A Aprendizado Supervisionado 51
A.1 Introdução . . . 51
A.2 Aprendizado supervisionado . . . 52
A.3 Aprendizado não-supervisionado . . . 52
A.4 Validação dos sistemas de aprendizado . . . 53
A.5 Taxa de erro . . . 53
A.6 Estimativa da taxa de erro real . . . 53
A.6.1 Treinar e testar . . . 53
A.6.2 Reamostragem . . . 54
A.6.3 Validação cruzada (Cross Validation) . . . 54

Referências Bibliográficas 55

Índice Remissivo 57

Lista de Abreviaturas

PP	Planejamento probabilístico
MDP	Processo de desisão markoviano (*Markov Decision Process*)
IV	Iteração de valor
RL	Aprendizado por reforço (*Reinforcement Learning*)
DP	Programação Dinâmica (*Dynamic Programming*)
BRL	Aprendizado por reforço em lote (*Batch reinforcement Learning*)
RTDP	Programação dinâmica em tempo real (*Real-time dynamic programming*)
RTDP	Programação dinâmica em tempo real rotulada (*Labeled RTDP*)
UCT	*Upper Confidence Bound applied to trees*
PF	Pessoa física
PJ	Pessoa jurídica

Lista de Figuras

1.1 Modelo de um sistema de suporte à tomada de decisão num processo de vendas . 3

3.1 Modelo teórico de aprendizado por reforço apresentado por (Sutton e Barto, 1998). 13
3.2 Paralelo entre planejamento probabilístico e aprendizado por reforço 14

4.1 Problema geral do aprendizado por reforço em lote adaptado de Sascha Lange e Riedmiller (2012). 19
4.2 Paralelo entre planejamento probabilístico, aprendizado por reforço e aprendizado por reforço em lote 20
4.3 Fases do aprendizado por reforço em lote crescente adaptado de Sascha Lange e Riedmiller (2012). 25
4.4 Classificação dos algoritmos de aprendizado por reforço em lote vs. aprendizado por reforço tradicional de acordo com a perspectiva de uso dos dados e interação ((Sascha Lange e Riedmiller, 2012)). 26

5.1 Ciclo de Vendas: etapas básicas de um processo de venda. 28
5.2 Ciclo de Vendas do Programa Big Impression. Na etapa 08 (Standby) o vendedor aguarda por um tempo antes de retomar o processo de negociação. 29

6.1 Diagrama de banco de dados modelado como um workflow 38
6.2 Diagrama de banco de dados modelado na forma de transições (tuplas $< p, s, a, s', r, h >$) 38
6.3 Tabela auxiliar de execução de ações usada para a construção do banco de dados da Figura 6.2 38
6.4 Fases do aprendizado por reforço em lote crescente usado para resolver o problema do vendedor de impressoras 40

7.1 Aprendizado por reforço em lote crescente sem repetição 44
7.2 Aprendizado por reforço em lote para o problema do vendedor de impressoras variando a quantidade de experiências da base (25%, 50%, 75% e 100%) e o número de repetições (de 0, 1, 2, 3, 6 e 10). 45
7.3 Aprendizado por reforço em lote com regressão vs aprendizado por reforço em sem regressão 45
7.4 K-Fold para verificar a qualidade da amostra (sem repetição). 46

8.1 Fases do aprendizado por reforço em lote crescente usado para resolver o problema do vendedor de impressoras. 49

A.1 Tipos de aprendizado. 51
A.2 Cross Validation. Método K-fold 54

Lista de Tabelas

5.1 Lista de variáveis de estado consideradas relevantes e utilizadas no modelo do problema do vendedor de impressoras. 32
5.2 Recompensas definidas para o problema do vendedor de impressoras 35

Lista de Algoritmos

1 IteracaoDeValorHorizonteInfinito(M, ϵ) 10
2 IteracaoDeValorHorizonteFinito(M, ϵ, H) 11

3 Q-Learning (M, α, γ, s_0) 16
4 Dyna-Q (M, s_0 α, γ, N) 18

5 BRL(Q_0, m) adaptado de (Kalyanakrishnan e Stone, 2007) 22
6 BRLRepeticaoDeExperiencias(Q_0, D) adaptado de (Kalyanakrishnan e Stone, 2007) 23
7 BRLAjustado(Q_0, D) adaptado de (Kalyanakrishnan e Stone, 2007) 24
8 BRLCrescente(Q_0, m) adaptado de (Kalyanakrishnan e Stone, 2007) 25

9 BRLCrescenteSemExploracao(Q_0, m) 40
10 BRLCrescenteComRegressao(Q_0, m) 41

Capítulo 1

Introdução

Planejamento é um processo de escolha de ações através da previsão de seus efeitos com o intuito de satisfazer objetivos previamente definidos. Planejamento automático (Ghallab *et al.* , 2004) é a subárea da *Inteligência Artificial* que estuda métodos para automatizar esse processo através de algoritmos independentes de domínio para serem usados por agentes inteligentes (robóticos ou de software).

A maioria das abordagens de planejamento baseiam-se no modelo chamado de *sistema de transição de estado* (Ghallab *et al.*, 2004). Formalmente, um sistema de transição de estados pode ser descrito pela tupla $E = (\mathcal{S}, \mathcal{A}, T)$ em que:

- $\mathcal{S}$ é um conjunto finito de estados,
- $\mathcal{A}$ é o conjunto de ações sendo $\mathcal{A}(s)$ o conjunto de ações aplicáveis no estado $s \in S$,
- $T : \mathcal{S} \times \mathcal{A} \rightarrow 2^{\mathcal{S}}$ é a função de transição de estados sendo que para cada estado $s \in \mathcal{S}$ e ação $a \in \mathcal{A}(s)$, devolve um subconjunto $S_r \subseteq \mathcal{S}$ que representa os possíveis estados resultantes da aplicação de a.

A função de transição de estados T define 3 tipos possíveis de planejamento: (i) planejamento determinístico ($T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$) em que a execução de uma ação leva o agente para um único estado; (ii) planejamento não-determinístico ($T : \mathcal{S} \times \mathcal{A} \rightarrow 2^{\mathcal{S}}$) em que a execução de uma ação pode levar o agente para um dos estados de um subconjunto de S; (ii) planejamento probabilístico ($T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$) em que a execução de uma ação pode levar o agente para um dos estados de S de acordo com uma distribuição de probabilidade.

Planejamento Probabilístico é a área que trata problemas de tomada de decisão em que as ações possuem efeitos probabilísticos. Os problemas de planejamento probabilístico são normalmente modelados como um processo de decisão markoviano (Markov Decision Process - MDP) (Bellman, 1957). Um MDP modela a interação entre um agente e um ambiente como um processo de tempo discreto. A ação executada por um agente em um dado estado possui efeito probabilístico e leva o agente para outro estado com uma certa probabilidade. Para cada ação executada o agente vai para um novo estado e recebe uma recompensa. O objetivo do agente é maximizar a recompensa esperada sobre uma sequência de execução de ações. Assim, o objetivo de um planejador probabilístico é encontrar uma *política ótima*, isto é, uma função que mapeia para cada estado do sistema uma ação para o agente executar, de forma que a recompensa esperada acumulada seja máxima. Para o cálculo da política ótima o agente precisa levar em consideração o *horizonte*, isto é, o número de passos que o agente deve executar ações. O horizonte pode ser: *finito* quando o número de ações que o agente deve tomar é fixo e conhecido; *infinito* quando as ações devem ser feitas por um número infinito de passos ou *indefinido* semelhante ao finito porém, o número de ações a serem executadas é desconhecido.

No *planejamento probabilístico*, nem sempre é possível ter uma descrição completa do problema e construir uma política baseando-se no conhecimento completo do ambiente. Neste caso, o agente precisa aprender a melhor política através de tentativa e erro fazendo interações com

o ambiente (Kaelbling *et al.*, 1996). Dá-se a esse processo o nome de *aprendizado por reforço* (Sutton e Barto, 1998).

Algoritmos básicos de aprendizado por reforço, como Q-learning (Watkins e Dayan, 1992), geralmente precisam de muitas interações até a convergência para boas políticas, tornando muitas vezes sua aplicação para problemas reais impossível (Sascha Lange e Riedmiller, 2012). Nas situações críticas em que o agente não pode interagir com o ambiente durante o processo de aprendizado (por exemplo, aprender a dirigir um automóvel no trânsito de São Paulo ou aprender procedimentos cirúrgicos realizados em pacientes humanos), a solução é aprender baseando-se em um simulador ou utilizando um número fixo de amostras de interações passadas. Para estas situações são utilizadas adaptações das técnicas de aprendizado por reforço conhecidas como *aprendizado por reforço em lote* (Batch Reinforcement Learning - BRL) (Ernst *et al.*, 2005).

O termo *aprendizado por reforço em lote* é usado para descrever um cenário de aprendizado por reforço em que um conjunto de experiências de aprendizagem, geralmente um conjunto de transições, é dada *a priori*. A tarefa do sistema de aprendizado, então, deixa de ser o cálculo da solução ótima, e passa a ser o cálculo da melhor solução possível a partir dos dados fornecidos. Para isso, os algoritmos de aprendizado por reforço em lote precisam alcançar uma maior eficiência tirando o melhor proveito do conjunto de experiências.

No aprendizado por reforço devemos encontrar uma política ótima que é definida através da convergência de uma função valor. No aprendizado por reforço em lote, em que usamos um conjunto finito de experiências, devemos encontrar uma política ótima aproximada. O objetivo é encontrar a melhor política possível a partir dos dados fornecidos.

Devido à eficiência no uso dos dados para aprendizado, os algoritmos modernos de aprendizado por reforço em lote tratam problemas conhecidos do aprendizado por reforço como a sobrecarga do processo de exploração (Sascha Lange e Riedmiller, 2012). Também garantem estabilidade e aproximação da política ótima.

1.1 Motivação

Um exemplo interessante em que é possível aplicar técnicas de planejamento probabilístico e aprendizado por reforço em lote é o problema do **vendedor de impressoras**. Neste problema, uma sequência de ações são executadas por um vendedor com o objetivo de influenciar o cliente à compra. Exemplos de ações são: *ligar para o cliente*; *oferecer um determinado produto*; *fazer uma demonstração do produto*; *negociar preço*. O resultado das ações executadas é incerto e depende de inúmeros fatores, alguns deles desconhecidos pelo vendedor. Mesmo um vendedor experiente é incapaz de conhecer todos os aspectos que interferem no processo. Por exemplo, a empresa interessada na compra pode estar passando por um processo de reestruturação ou o responsável pela compra pode estar passando por problemas pessoais. Um banco de dados de experiências passadas de processos de venda pode ser usado para o aprendizado automático de políticas de venda. Para isso é possível aplicar técnicas de aprendizado por reforço em lote para a construção de um sistema de suporte à tomada de decisão num processo de vendas.

A Figura 1.1 mostra como seria um sistema de suporte à tomada de decisão que baseia-se num banco de dados de experiências passadas. O banco de dados de processos de venda é constantemente alimentado por cada vendedor. O sistema de aprendizado utiliza o conjunto de registros armazenados de um determinado período de tempo, e mantém um fator de aprendizado diante dos novos registros. No Problema do Vendedor de Impressoras as informações coletadas, e usadas no processo de aprendizado, são provenientes tanto de vendedores experientes quanto de vendedores inexperientes. O sistema precisa ser capaz de aprender com os casos de sucesso, mas também com os casos de fracasso.

Para aplicar métodos de aprendizado em problemas reais, como processos de venda, modelados como MDP, algumas questões precisam ser respondidas e alguns problemas precisam ser tratados:

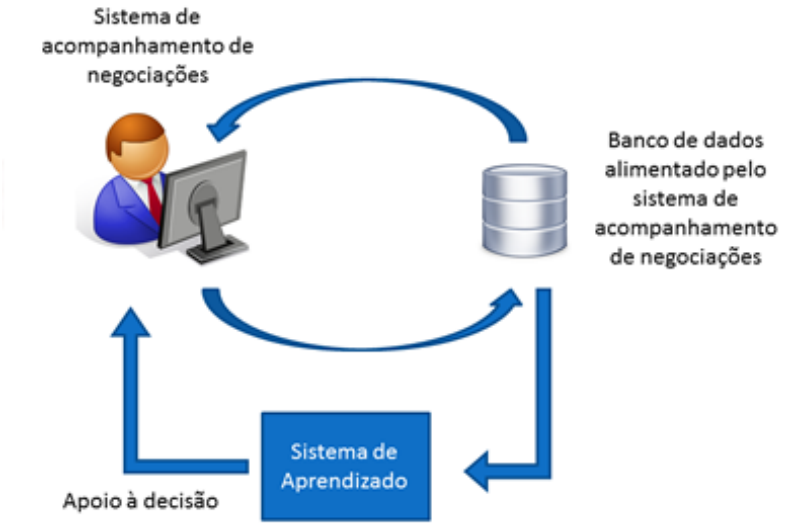


Figura 1.1: *Modelo de um sistema de suporte à tomada de decisão num processo de vendas*

- O problema possui uma função de transição *estacionária* ou *não-estacionária*? Por exemplo, os dados sofrem alterações com o tempo?
- Qual é a quantidade de experiências anteriores necessárias para aprender uma política ótima?
- Dado um banco de dados com n experiências, qual é o erro percentual gerado pela política aprendida com relação à política ótima?

1.2 Proposta

Neste trabalho propomos usar as técnicas de aprendizado por reforço em lote no problema de tomada de decisões em processos de venda utilizando um banco de dados com histórico de processos passados. As políticas aprendidas devem maximizar o sucesso das vendas enquanto minimizam o custo do processo. O banco de dados utilizado conta com um histórico de negociações de impressoras de grande formato (*plotters*), que pertence ao programa de vendas de impressoras HP no Brasil chamado *Big Impression* (Impression, 2013), que conta com um grupo de revendas e parceiros localizados em todo território brasileiro. Este banco de dados contém registros de mais de 20 mil negociações coletadas desde 2007.

A política ótima aprendida a partir deste banco de dados poderá ser usada de três diferentes formas pelo programa Big Impression:

- recomendação de ações para os vendedores do grupo,
- identificação de vendedores que necessitam de treinamento e
- detecção das causas de vendas sem sucesso.

Além disso, esse trabalho propõe duas novas extensões para o algoritmo de aprendizado por reforço em lote: aprendizado por reforço em lote crescente sem exploração e aprendizado por reforço em lote com regressão de processos.

Para resolver o problema com aprendizado por reforço em lote, o histórico de registro de negociações do banco de dados *Big Impression* foi convertido para uma base de tuplas que representam transições de estados de um MDP, e o conjunto de registro usados como entrada para o algoritmo de BRL.

Para nossas análises definimos uma medida de convergência de lote que também pode ser usada com a técnica de validação do aprendizado supervisionado, em particular a técnica de *Validação Cruzada* (Kohavi, 1995) (*K-Fold* do aprendizado supervisionado), em que verificamos a convergência da política devolvida pelos algoritmos em relação a um conjunto de testes após treiná-lo com um conjunto distinto de experiências passadas. Em nossas análises abordamos os seguintes temas:

- convergência para uma política ótima considerando um determinado lote de experiências com e sem repetição de experiências,
- comparação de desempenho dos diferentes algoritmos de aprendizado por reforço em lote propostos na literatura e das duas extensões propostas nesse trabalho,
- comparação entre as ações sugeridas pela política aprendida e as ações tomadas pelos vendedores, e como isso pode ser usado para identificar vendedores que precisam ser treinados.

1.3 Contribuições desse trabalho

Uma das principais contribuições desse trabalho está no estudo das diversas técnicas de aprendizado por reforço em lote, na proposta de melhorias e no estudo de caso envolvendo a aplicação de aprendizado por reforço em lote em um sistema real de aprendizado de política de venda. Propomos uma maneira de modelar um histórico de processos de venda em transições de processos de decisão markoviano permitindo aplicar aprendizado por reforço em lote para construir uma política ótima de ações de venda. Outra contribuição desse trabalho está na proposta de uma medida de erro de convergêcia que permite comparar o desempenho de algoritmos em que são utilizadas amostras para o aprendizado de políticas, como o aprendizado por reforço em lote.

1.4 Organização do texto

Este texto está organizado nos seguintes capítulos:

Capítulo 2 Fundamentos de planejamento probabilístico. Apresentamos o processo de decisão markoviano (o principal modelo para planejamento probabilístico), e alguns algoritmos para resolver problemas modelados como um processo de decisão markoviano como o algoritmo de iteração de valor.

Capítulo 3 Fundamentos de aprendizado por reforço. Apresentamos o problema de aprendizado por reforço que são problemas modelados como um MDP em que não é possível ter uma descrição completa do problema, e que a mesma precisa ser aprendida. Apresentamos alguns métodos que resolvem o problema, como o algoritmo *Q-Learning* e discutimos algumas deficiências que serão tratadas nos capítulos posteriores.

Capítulo 4 Aprendizado por reforço em lote. Apresentamos as técnicas de aprendizado por reforço em lote que resolvem deficiências conhecidas do aprendizado por reforço tradicional, como a sobrecarga do processo de exploração, e permitem que o aprendizado seja feito sem interação com ambiente através do uso de um banco de dados de experiências conhecidas *a priori*.

Capítulo 5 Estudo de caso. Apresentamos o problema do vendedor de impressoras, um problema real em que deseja-se construir um sistema que apoie vendedores na tomada de decisão de ações de venda. Mostramos neste capítulo como este problema pode ser modelado como um problema de aprendizado por reforço em lote.

Capítulo 6 Resolvendo o problema do vendedor de impressoras. Mostramos neste capítulo como o problema do vendedor de impressoras pode ser resolvido usando aprendizado por reforço em lote. Abordamos as dificuldades, e soluções encontradas, para transformar os dados do formato em que eles são disponibilizados para o formato necessário para aplicar aprendizado por reforço em lote. Propomos algumas melhorias nos algoritmos de aprendizado por reforço em lote e uma maneira a avaliar o aprendizado inspirada nos métodos de validação cruzada.

Capítulo 7 Resultados experimentais. Mostramos neste capítulo alguns resultados obtidos ao aplicar as diversas técnicas de aprendizado por reforço em lote para resolver o problema do vendedor de impressoras. Analisamos ainda as melhorias propostas para os algoritmos apresentadas neste trabalho.

Capítulo 2

Planejamento probabilístico

Um problema de planejamento probabilístico é um problema de planejamento em que a função de transição de estados é dada em termos de distribuições de probabilidades. Ou seja, para cada estado $s \in S$ e ação $a \in A$, é dada uma distribuição de probabilidade sobre o conjunto de estados sucessores $s' \in S$. Problemas de planejamento probabilístico são normalmente modelados como um processo de decisão markoviano.

2.1 Processo de decisão markoviano

Um processo de decisão markoviano (Markov Decision Process - MDP) (Bellman, 1957) (Russell e Norvig, 2003) é um modelo matemático usado para modelar tomada de decisões sequenciais em que uma ação tem de ser escolhida pelo agente em cada estado. A execução de uma ação levará, probabilisticamente, a um estado e uma recompensa será recebida. O objetivo do agente é maximizar a recompensa acumulada com o passar do tempo.

Num MDP os efeitos das ações são probabilísticos e obedece a propriedade de Markov: o efeito de uma ação depende apenas do estado atual do sistema e não de como o processo chegou a tal estado.

Definição 2.1 *Um MDP pode ser definido formalmente como uma tupla* $\mathcal{M} = (S, A, P, R, S_0)$ *em que:*

- *S é um conjunto finito de estados. Se o processo tem conhecimento total do estado atual, então o MDP é chamado de* totalmente observável*, caso contrário, é* parcialmente observável.
- *A é um conjunto finito de ações que podem ser executadas pelo agente e permitem que o sistema mude de estado.*
- *P define as probabilidades de transição sobre S, sendo $P(s'|s, a)$ a probabilidade condicional de ir para o estado $s' \in S$, dado que o agente está no estado $s \in S$ e executa a ação $a \in A$. Note que as transições são markovianas pois a probabilidade de alcançar s' a partir de s depende apenas de s, ou seja, independe dos estados anteriores,*
- *$R : S \times A \rightarrow \mathbb{R}$ é a função de recompensa, sendo que R(s,a) representa a recompensa obtida pelo agente dado que ele está no estado $s \in S$ e executa a ação $a \in A$.*
- *S_0 é o conjunto de estados iniciais com $S_0 \subseteq S$*

Definição 2.2 *O* horizonte *h de um MDP representa o número limite de passos (escolhas de ações) disponíveis para o agente. O horizonte pode ser* finito *quando o número de ações que o agente deve tomar é fixo e finito,* infinito *quando a tomada de ações deve ser feita repetidamente e sem possibilidade de parar ou* indefinido *semelhante ao infinito porém, com possibilidade de parar em estados meta (ou becos sem saída).*

Definição 2.3 *Uma* política *de um MDP é uma função $\pi : S \rightarrow A$ que mapeia para cada estado do sistema qual ação o agente deve escolher. Em geral, resolver um processo de decisão se resume em encontrar boas políticas que otimizam de alguma forma um critério de desempenho (por exemplo, maximização da recompensa acumulada esperada, minimização do custo acumulado esperado, etc.).*

Uma política pode ser classificada como:

- *Total, se é definida para todos os estados em S;*
- *Parcial, se é definida apenas para um subconjunto $S_r \subseteq S$.*

Uma política também pode ser classificada como:

- Estacionária*, se a ação recomendada para um dado estado $s \in S$ independe do momento de decisão,*
- Não-estacionária*, se a ação recomendada para um dado estado $s \in S$ pode variar durante o processo.*

Definição 2.4 *A função valor de uma política π para um MDP $\mathcal{M}$ é uma função $V^\pi : S \rightarrow \mathbb{R}$ em que $V^\pi(s)$ devolve o valor esperado da recompensa para esta política a partir do estado $s \in S$, isto é:*

$$V^\pi(s) = E[\sum_{i=0}^{\infty} r_i | s_0 = s], \tag{2.1}$$

Em problemas de horizonte infinito V^π pode não convergir para um valor finito tornando impossível comparar duas políticas. Para que V^π seja limitada, as recompensas recebidas são amortizadas por um fator de desconto γ ($0 \leq \gamma < 1$). Quanto menor o fator de desconto, mais valor será dado às recompensas imediatas em relação às recompensas futuras, isto é:

$$V^\pi(s) = E[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s], \tag{2.2}$$

que corresponde a:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V^\pi(s'), \forall s \in S. \tag{2.3}$$

Dados um estado $s \in S$, uma ação $a \in A$ e uma política π para um MDP, pode-se definir o valor da ação a no estado s seguindo a política π, pela recompensa imediata de a somada a recompensa esperada após a. Essa função denotada por Q^π é definida como:

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^\pi(s'), \forall s \in S, a \in A. \tag{2.4}$$

$$V^\pi(s) = \max_{a \in A} Q^\pi(s, a). \tag{2.5}$$

2.1.1 Política ótima

Dado um MDP $\mathcal{M}$, o processo de decisão é determinar π^* tal que $V^{\pi^*}(s) \geq V^\pi(s)$ para todo π e para todo s. Chamamos π^* de *política ótima* de $\mathcal{M}$.

A função valor ótima V^* é a função valor associada a qualquer política ótima que é calculada pela seguinte equação (Bellman, 1957):

$$V^*(s) = \max_{a \in A} \{R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s')\}. \tag{2.6}$$

A Equação 2.6 é chamada de *Equação de Bellman* e os principais algoritmos para encontrar uma política ótima baseiam-se nesta equação. Note que uma forma mais simples de escrever esta equação é fazer a maximização da função $Q^*(s,a)$

$$V^*(s) = \max_{a \in A}\{Q^*(s,a)\}, \tag{2.7}$$

em que

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V^*(s') \tag{2.8}$$

2.1.2 Algoritmos para resolver MDPs

O algoritmo de iteração de valor (Puterman, 1994) (Algoritmo 1) usa programação dinâmica para determinar o valor de $V^*(s)$ para cada estado s do MDP calculando de forma iterativa o valor estimado de $V^*(s)$ na iteração t denotado por $V_t(s)$, que é atualizada a cada iteração da seguinte forma:

$$V_t(s) \leftarrow \max_{a \in A}\{R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V_{t-1}(s')\}. \tag{2.9}$$

O algoritmo inicia com valores V_0 aleatórios. Em cada iteração i calcula-se V_i baseado na função valor V_{i-1}. O algoritmo também faz uso da função $Q(s,a)$ que representa o valor da ação a aplicada no estado s no passo, ou estágio, t, dada por:

$$Q_t(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V_{t-1}(s'). \tag{2.10}$$

E o valor do estado s no estágio de decisão t é igual ao valor da melhor ação:

$$V_t(s) = \max_{a \in A} Q_t(s,a). \tag{2.11}$$

Definição 2.5 *Chamamos de* ação gulosa *a ação $a \in A$ que maximiza $Q_t(s,a)$ para qualquer t, e* ação ótima *a ação $a \in A$ que maximiza $Q^*(s,a)$.*

Algoritmo 1: IteracaoDeValorHorizonteInfinito(M, ϵ)

Entrada: M: um MDP, ϵ: erro máximo permitido
Saída: π^*: uma política ótima

1 **para cada** $s \in S$ **faça**
2 | $V_0(s) \leftarrow$ *Valor Aleatório*;
3 **fim**
4 $t \leftarrow 1$
5 **para cada** $s \in S$ **faça**
6 | **para cada** $a \in A$ **faça**
7 | | $Q_t(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V_{t-1}(s')$;
8 | **fim**
9 **fim**
10 **repita**
11 | **para cada** $s \in S$ **faça**
12 | | **para cada** $a \in A$ **faça**
13 | | | $Q_t(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V_{t-1}(s')$
14 | | **fim**
15 | | $V_t(s) \leftarrow \max_{a \in A} Q_t(s,a)$
16 | | $\pi(s) \leftarrow arg \max_{a \in A} Q^t(s,a)$
17 | **fim**
18 | t ← t+1
19 **até** $|V_t(s) - V_{t-1}(s)| < \epsilon \ \forall s \in S$;
20 **retorna** π.

Para processos com horizonte infinito (ou indefinido) o algoritmo deve parar apenas após a convergência do valor de cada estado, ou seja, quando $\max_{s \in S}\{V_t(s) - V_{t-1}(s)\} < \epsilon$ para algum

ϵ suficientemente pequeno, neste caso, tem-se uma política estacionária.

Algoritmo 2: IteracaoDeValorHorizonteFinito(M, ϵ, H)

Entrada: M: um MDP, ϵ: erro máximo permitido, H: o horizonte
Saída: π^*: uma política ótima

1 **para cada** $s \in S$ **faça**
2 | $V(s,0) \leftarrow 0$;
3 **fim**
4 $h \leftarrow 1$
5 **para cada** $s \in S$ **faça**
6 | **para cada** $a \in A$ **faça**
7 | | $Q(s,a,h) \leftarrow R(s,a) + \sum_{s' \in S} P(s'|s,a)V(s',h-1)$;
8 | **fim**
9 **fim**
10 **repita**
11 | **para cada** $s \in S$ **faça**
12 | | **para cada** $a \in A$ **faça**
13 | | | $Q(s,a,h) \leftarrow R(s,a) + \sum_{s' \in S} P(s'|s,a)V(s',h-1)$
14 | | **fim**
15 | | $V(s,h) \leftarrow \max_{a \in A} Q(s,a,h)$
16 | | $\pi(s,h) \leftarrow arg\max_{a \in A} Q(s,a,h)$
17 | **fim**
18 | h $\leftarrow$ h+1
19 **até** $h \leq H$;
20 **retorna** π.

O critério de parada do algoritmo para horizonte finito é que o número de iterações seja igual ao horizonte do problema. No caso de horizonte finito, o algoritmo gera uma política para cada estágio de decisão (política não-estacionária). A complexidade do algoritmo é $O(h|A||S|^2)$.

A complexidade do algoritmo de Iteração de Valor é polinomial no número de estados, no entanto, o número de estados cresce exponencialmente com o número de variáveis de estado. Além disso, o algoritmo de Iteração de Valor é *off-line*, ou seja, calcula a política completa (para todos os estados do sistema) antes do agente executar suas ações. Isso faz com que os algoritmos apresentados sejam pouco úteis em sistemas grandes e que necessitem de tomadas de decisão em tempo real.

Existem diversos algoritmos para resolver MDPs de forma assíncrona e de maneira mais eficiente. Em geral estes algoritmos partem do mesmo princípio dos dois algoritmos apresentados porém resolvem o problema de forma relaxada. Isso faz com que o sistema perca a solução ótima mas permita encontrar uma política *anytime* com menor tempo de convergência. Exemplos de algoritmos eficientes para resolver MDPs são:

- **Programação Dinâmica em Tempo Real (RTDP)** (Barto *et al.*, 1993) Apresenta uma tática eficiente para atualização assíncrona dos valores dos estados. A ideia básica do algoritmo é simular inúmeras vezes a execução da política sorteando o próximo estado de acordo com sua probabilidade, e atualizar o valor dos estados ao mesmo tempo em que a política é atualizada. Esta maneira de escolher quais estados terão seu valor atualizado é muito poderosa porque os estados que influenciam mais no valor da esperança são atualizados mais vezes.

- **Programação Dinâmica em Tempo Real com Marcações (LRTDP)** (Bonet e Geffner , 2003). No RTDP, escolher qual estado s terá seu valor atualizado de acordo com a probabilidade dele ser alcançado faz com que $V^\pi(s)$ aproxime-se rápido de $V^*(s)$. No entanto, estados com pouca probabilidade de serem alcançados demoram para convergir. O LRTDP

aumenta a velocidade de convergência do RTDP marcando como resolvidos estados cujos valores já convergiram.

- **Upper Confidence Bound applied to trees (UCT)** (Kocsis e Szepesvári, 2006) Usa técnicas de amostragem de Monte-Carlo (Hastings, 1970) para resolver MDPs. Diferente de outras técnicas, o UCT não precisa conhecer as probabilidades de transição ou os valores de recompensa das ações, e funciona simulando a interação do agente com o ambiente utilizando um simulador. O planejador PROST (Keller e Eyerich, 2012), baseado no UCT, foi o campeão da competição internacional de planejamento probabilístico em 2011 (IPCC-2011).

Capítulo 3

Aprendizado por Reforço

Em problemas de tomada de decisão sequencial, nem sempre é possível ter uma descrição completa do ambiente e, consequentemente, construir uma política baseando-se nessa descrição. Neste caso, o agente precisa aprender a melhor política através de tentativa e erro, realizando interações com o ambiente (Kaelbling *et al.*, 1996). A esse processo dá-se o nome de *aprendizado por reforço* (Sutton e Barto, 1998).

Diferentemente do aprendizado supervisionado em que existe um professor que diz ao agente qual deveria ter sido a ação correta para cada estado, no aprendizado por reforço (*Reinforcement Learning - RL*) existe apenas uma medida que indica o quão boa, ou ruim, foi a escolha de uma ação, mas não diz exatamente qual teria sido a melhor ação. Esse tipo de aprendizado foi inspirado no processo de aprendizado infantil. Uma criança costuma realizar ações aleatórias e aprende quais ações são boas ou ruins de acordo com o *feedback* de seus pais.

Como num problema de planejamento probabilístico modelado como um MDP, no aprendizado por reforço temos um agente que se encontra no estado $s \in S$ e escolhe uma ação a ser executada que o leva para um outro estado s'. Para cada par estado-ação há um reforço $R(s, a)$ que é dado ao agente quando ele executa a ação a no estado s. Possíveis variações da função reforço são $R(s)$ em que o reforço é dado ao agente ao ser levado para o estado s, $R(s, a, s')$ em que o reforço é dado ao agente quando ele executa a ação a no estado s e é levado para o estado s', e $R(a)$ em que o reforço é dado ao agente após executar a ação a.

O relacionamento agente e ambiente é ilustrado na Figura 3.1. Note que o agente de aprendizado por reforço (agente AR) seleciona uma ação a_t no estado s_t e recebe do ambiente um reforço r_{t+1} e é levado para o estado s_{t+1}.

O objetivo do problema de RL consiste em escolher uma política de ações que maximize o total de recompensas recebidas pelo agente após n interações com o ambiente. Essa política é denominada *política ótima*.

Como é possível perceber, a descrição de um problema de *aprendizado por reforço* é muito parecida com a descrição de um *processo markoviano de decisão (MDP)* descrito no Capítulo 1. A

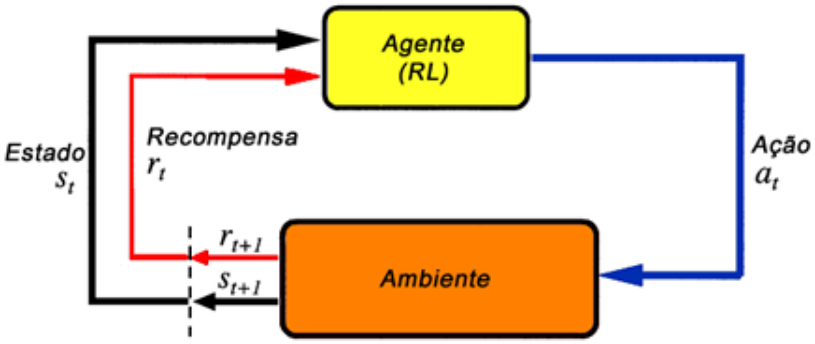


Figura 3.1: *Modelo teórico de aprendizado por reforço apresentado por (Sutton e Barto, 1998).*

	Fase A	Fase B
(PP) Planejamento Probabilístico	Cálculo da política ótima (*offline*)	Aplicação da política calculada (*online*)
(RL) Aprendizado por reforço	Aprendizado da política ótima (*online*)	Aplicação da política aprendida (*online*)

Figura 3.2: *Paralelo entre planejamento probabilístico e aprendizado por reforço*

única diferença é que no *aprendizado por reforço* as transições entre os estados são desconhecidas, e a recompensa é chamada de *sinal de reforço*. Em alguns casos, a função reforço R também pode ser desconhecida.

Definição 3.1 *Um problema de aprendizado por reforço pode ser definido formalmente como uma tupla* $\mathcal{RL} = (S, A, R, S_0, \alpha, \gamma)$ *em que:*

- S *é um conjunto finito de estados,*
- A *é um conjunto finito de ações que podem ser executadas pelo agente e permitem que o sistema mude de estado,*
- $R : S \times A \rightarrow \mathbb{R}$ *é a função de reforço, sendo que* $R(s,a)$ *representa a recompensa obtida pelo agente dado que ele está no estado* $s \in S$ *e executa a ação* $a \in A$*,*
- S_0 *é o conjunto de estados iniciais com* $S_0 \subseteq S$*,*
- α *é o fator de aprendizado,*
- γ *é o fator de desconto.*

A Figura 3.2 faz um paralelo entre planejamento probabilístico e aprendizado por reforço. No planejamento probabilístico o cálculo da política ótima é feito de maneira *offline* (Fase A), isto é, sem interação com o ambiente, com base no modelo completo do ambiente, em seguida a política calculada pode ser aplicada no ambiente (Fase B). No aprendizado por reforço o agente não possui o modelo completo do ambiente, ou seja, a função $P(s'|s, a)$ é desconhecida e portanto precisa aprender a política interagindo com o ambiente (Fase A) e em seguida a política aprendida pode ser aplicada com sucesso no ambiente (Fase B).

Para resolver um problema de aprendizado por reforço é preciso atentar-se a algumas características do:

- **Aprendizado por interação:** O agente age no ambiente e espera pelo valor do reforço que o ambiente devolve e pelo estado resultante da ação.
- **Retorno atrasado:** um valor de reforço alto recebido pelo agente ao escolher uma ação em um determinado estado não significa que a ação escolhida é recomendada. Uma ação é produto de uma decisão local no ambiente, mas o intuito do agente é alcançar objetivos globais. A qualidade das ações deve ser vista pelas interações a longo prazo.
- **Orientado pelo objetivo:** no aprendizado por reforço não é necessário conhecer detalhes da modelagem do ambiente. Simplesmente existe um agente que age neste ambiente desconhecido tentando maximizar seu sucesso ao longo de uma sequência de execução de ações.

- **Exploração versus Explotação:** consiste na decisão entre aprender mais sobre as partes desconhecidas do ambiente (*exploração*) ou aperfeiçoar o conhecimento do ambiente já conhecido (*explotação*). Há casos em que o agente precisa explorar o ambiente para encontrar recompensas globais maiores ou evitar becos sem saída. Mas também há casos onde é primordial fazer boas escolhas locais. Para que um sistema seja realmente autônomo, a decisão entre a exploração e explotação deve ser tomada pelo agente. Uma boa estratégia é mesclar os dois modos.

3.1 Q-Learning

Q-Learning é um método proposto por (Watkins e Dayan, 1992) para resolver problemas de aprendizado por reforço. Este método estima de forma iterativa, com interações com o ambiente real, a função $Q^*(s,a)$ que determina a esperança de recompensas futuras quando o agente executa a ação a no estado s e continua agindo de acordo com a política aprendida até o momento daí em diante. Como num MDP de horizonte infinito, a soma das recompensas futuras pode ser infinita, e portanto usamos um fator de desconto γ ($0 \leq \gamma < 1$). Quanto menor este fator maior valor será dado às recompensas próximas do estado atual, e assim, a função Q define a soma das recompensas futuras descontadas.

Dada uma estimativa da função Q^*, denotada por Q, a política a ser seguida pelo agente pode ser facilmente determinada. Para maximizar a recompensa esperada basta o agente escolher em cada estado s a ação a que tem maior valor de $Q(s,a)$.

Entretanto, seguindo essa abordagem, o agente perde uma parte de sua capacidade de aprender. Como em cada um dos estados é sempre executada a ação de maior valor de Q, apenas essa ação terá seu valor atualizado. Dessa forma torna-se necessário intercalar fases de exploração e explotação durante o processo de aprendizado.

É importante fazer uma ponderação entre o valor das ações que foi aprendido em experiências anteriores com o valor que será aprendido em experiências futuras. Em ambiente não determinístico isto evita que uma recompensa obtida ao acaso muito diferente da média real tenha um peso alto na atualização da política. Ao mesmo tempo evita que o resultado de experiências distantes interfira na política em ambientes dinâmicos. Para isso é usado um fator α que pode ser compreendido como um *fator de aprendizado* ou *fator de esquecimento*.

Considere Q_t a estimativa de $Q^*(s,a)$ no instante t. O aprendizado de $Q^*(s,a)$ dá-se através da seguinte fórmula:

$$Q_{t+1}(s_t,a_t) \leftarrow (1-\alpha)Q_t(s_t,a_t) + \alpha(r(s_t,a_t) + \gamma \max_{a'} Q_t(s_{t+1},a')), \tag{3.1}$$

em que:

- $s_t \in S$ é o estado no instante t do processo de aprendizado,
- a_t é a ação realizada em s_t,
- $r(s_t,a_t)$ é a recompensa recebida do ambiente ao executar a ação a_t no estado s_t,
- s_{t+1} é o novo estado após executar a ação a_t,
- γ é o fator de desconto e
- α é a fator de aprendizado, ou seja, o fator de ponderação entre a estimativa do valor de Q até o instante t e a nova recompensa recebida.

Note que a Equação 3.1 pode ser descrita como

$$Q_{t+1}(s_t,a_t) \leftarrow (1-\alpha)Q_t(s_t,a_t) + \alpha(\bar{Q}_t(s_t,a_t)), \tag{3.2}$$

sendo

$$\bar{Q}_t(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}). \tag{3.3}$$

Chamamos a Equação 3.2 de "passo de aproximação da função Q" e a Equação 3.3 de "passo de programação dinâmica".

O algoritmo Q-Learning aproxima iterativamente Q para a função Q^*. Esta convergência é garantida desde que se satisfaça as seguintes condições (Mitchell, 1997):

- o sistema pode ser modelado como um MDP estacionário,
- a função de recompensa é limitada, isto é, existe alguma constante positiva c tal que, para todos os estados s e ação a, $|r(s, a)| < c$,
- as ações devem ser escolhidas de maneira que cada par estado-ação seja visitado um número infinito de vezes.

O Algoritmo 3 mostra um exemplo de implementação do algoritmo Q-Learning. Note que o algoritmo não especifica como as ações são escolhidas pelo agente. As ações podem ser escolhidas usando qualquer estratégia de exploração (por exemplo, testar o efeito de uma ação nos estados em que a ação ainda não foi executada) ou explotação (testar novamente o efeito de uma ação num estado em que a ação já foi testada).

Algoritmo 3: Q-Learning (M, α, γ, s_0)

Entrada: M: um modelo, s_0: um estado inicial,
α: fator de aprendizado, γ: fator de desconto

1 **para cada** $s \in S$ **faça**
2 | **para cada** $a \in A$ **faça**
3 | | $Q_0(s, a) \leftarrow$ valor aleatório
4 | **fim**
5 **fim**
6 $t \leftarrow 0$
7 $a_0 \leftarrow arg \max_{a' \in A} Q_0(s, a')$
8 **enquanto** *critério de parada não é satisfeito* **faça**
9 | $[s_{t+1}, r] \leftarrow executaAcao(s_t, a_t)$
10 | $a_{t+1} \leftarrow arg \max_{a' \in A} Q_t(s_{t+1}, a')$
11 | $V_t(s_{t+1}) \leftarrow \max_{a' \in A} Q_t(s_{t+1}, a')$
12 | $Q_{t+1}(s_t, a_t) \leftarrow (1 - \alpha)Q_t(s_t, a_t) + \alpha[r + \gamma V_t(s_{t+1})]$
13 | $t \leftarrow t + 1$
14 **fim**

Uma estratégia bastante utilizada é a exploração aleatória denominada ϵ-Greedy em que o agente executa a ação gulosa com probabilidade $1 - \epsilon$ e escolhe uma ação aleatória com probabilidade ϵ. Quanto menor o valor de ϵ, menor a probabilidade de se fazer uma escolha de ação aleatória e menor a taxa de exploração do agente. Apesar da garantia de convergência do algoritmo *Q-Learning*, ela é extremamente lenta.

3.2 SARSA

O algoritmo *SARSA* (*State-Action-Reward-State-Action*) (Rummery e Niranjan, 1994) trata-se de uma modificação do algoritmo Q-Learning. Neste método a ação a_{t+1} escolhida não é necessariamente a ação mais valorizada, ou seja, que maximiza $Q_t(s_{t+1}, a_{t+1})$, e sim escolhida aleatoriamente. Somente após a execução de a_{t+1} em s_{t+1} é que $Q_t(s_t, a_t)$ é atualizada:

$$Q_{t+1}(s_t, a_t) \leftarrow (1-\alpha)Q_t(s_t, a_t) + \alpha(r(s_t, a_t) + \gamma Q_t(s_{t+1}, a_{t+1})) \tag{3.4}$$

A proposta surgiu da constatação de que o algoritmo Q-Learning, ao considerar as ações mais valorizadas desprezando as restantes, poderia apresentar alguns problemas. Por exemplo, uma política gulosa que seleciona um caminho mais curto, desprezando outros que poderiam ser mais seguros por manterem a distância de estados que devem ser evitados. Sendo assim, o algoritmo SARSA alcança um melhor rendimento em problemas em que o conjunto de ações é muito grande ou em problemas em que o ambiente apresenta apenas penalidades, isto é, reforços negativos (Sutton, 1996).

3.3 Dyna-Q

A idéia básica do *Dyna-Q* (Sutton, 1990) é acrescentar ao algoritmo *Q-Learning* a capacidade de planejamento, através da atualização contínua de um modelo do ambiente, simulando um número pré-determinado de execuções sobre este. Assim, cada experiência real pode ser acompanhada por inúmeras experiências simuladas realizadas sobre o modelo do mundo, cada uma destas contribuindo para o processo de aprendizagem. O Algoritmo 4 apresenta o algoritmo Dyna-Q proposto por Sutton (1990).

3.4 Conclusão

Os três algoritmos de aprendizado por reforço apresentados neste capítulo possuem as seguintes limitações:

1. necessitam de muitas interações com o ambiente real para encontrar boas políticas,
2. possuem comportamento instável em algumas situações,
3. são inviáveis em sistemas em que as ações possuem um custo muito elevado.

No Capítulo 4 mostramos como a técnica de aprendizado por reforço em lote pode superar estas limitações.

Algoritmo 4: Dyna-Q (M, s_0 α, γ, N)

Entrada: M: um modelo, s_0: um estado inicial,
α: fator de aprendizado, γ: fator de desconto
N: número de simulações com o modelo parcial

```
para cada s ∈ S faça
    para cada a ∈ A faça
        Q_0(s, a) ← valor aleatório
    fim
fim
t ← 0
enquanto critério de parada não é satisfeito faça
    //Experiência real
    a_t ← ε − Greed(s, Q)
    Executa ação a_t e observa estado resultante s_{t+1} e recompensa r_{t+1}
    Q_{t+1}(s_t, a_t) ← (1 − α)Q_t(s_t, a_t) + α[r + γV_t(s_{t+1})]
    modelo(s_t, a_t) ← (s_{t+1}, r_{t+1})
    t ← t + 1
    i ← 0
    //Simulação com o modelo parcial estimado
    enquanto i < N faça
        s_t ← estado aleatório previamente observado no modelo
        a_t ← ação aleatória executada previamente em s_t
        (s_{t+1}, r) ← modelo(s_t, a_t)
        Q_{t+1}(s_t, a_t) ← (1 − α)Q_t(s_t, a_t) + α[r + γV_t(s_{t+1})]
        t ← t + 1
        i ← i + 1
    fim
fim
```

Capítulo 4

Aprendizado por reforço em lote (Batch Reinforcement Learning - BRL)

Conforme discutimos no capítulo anterior, algoritmos básicos de aprendizado por reforço, como Q-learning, Sarsa e Dyna-Q, geralmente precisam de muitas interações até a convergência para boas políticas, tornando sua aplicação para diversos problemas reais impossível. Além disso, há situações críticas em que o agente não pode interagir com o ambiente durante o processo de aprendizado. Neste caso, é possível aprender baseando-se em um número fixo de amostras de interações conhecidas *a priori* (por exemplo, realizadas e armazenadas por especialistas humanos). Para estas situações são utilizadas adaptações das técnicas de aprendizado por reforço conhecidas como *aprendizado por reforço em lote* (Batch Reinforcement Learning - BRL) (Ernst *et al.*, 2005).

Aprendizado por reforço em lote é uma extensão do aprendizado por reforço, que usa um conjunto de experiências para o aprendizado de políticas sem realizar novas experiências durante o aprendizado. Este conjunto de experiências pode ser conhecido *a priori* ou ser adquirido numa fase independente da fase de aprendizado chamada de exploração (Ernst *et al.*, 2005).

Da mesma forma que um problema geral de aprendizado por reforço definido por Sutton e Barto (1998), o objetivo de um problema de aprendizado por reforço em lote é encontrar uma política que maximize a soma esperada de recompensas obtidas pelo agente após interagir no ambiente. No entanto, a diferença é que no problema de aprendizado em lote o agente não interage diretamente com o ambiente durante o aprendizado. Em vez de observar um estado s, tentar executar uma ação a e adaptar sua política de acordo com o próximo estado s' observado e a recompensa r recebida, o agente apenas seleciona uma experiência passada de um conjunto τ de p experiências obtidas de interações prévias com o ambiente, isto é, $\tau = \{e_1, e_2, ..., e_p\}$. Uma experiências e_i é dada pela tupla (s_i, a_i, r_i, s'_i) que representa a ocorrência da transição de s_i para s'_i através da ação a_i e recebendo a recompensa r_i (Sascha Lange e Riedmiller, 2012).

A Figura 4.1 mostra as fases de um aprendizado em lote: 1) *Exploração*, fase em que é feita a coleta de transições com uma estratégia de amostragem arbitrária (caso não esteja disponível

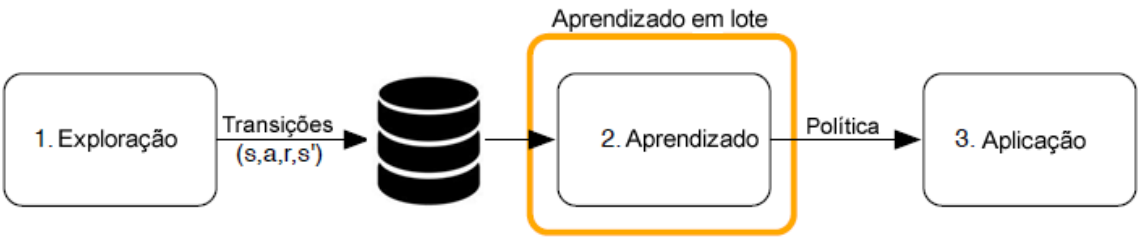


Figura 4.1: *Problema geral do aprendizado por reforço em lote adaptado de Sascha Lange e Riedmiller (2012).*

	Fase A	Fase B	Fase C
(PP) Planejamento Probabilístico		Cálculo da política ótima (*offline*)	Aplicação da política calculada (*online*)
(RL) Aprendizado por reforço		Aprendizado da política ótima (*online*)	Aplicação da política aprendida (*online*)
(BRL) Aprendizado por reforço em lote	Exploração do ambiente	Aprendizado da política ótima (*offline*)	Aplicação da política aprendida (*online*)

Figura 4.2: *Paralelo entre planejamento probabilístico, aprendizado por reforço e aprendizado por reforço em lote*

um conjunto de experiências prévias). 2) *Aprendizado em lote*, em que as amostras são utilizadas para determinar a política a ser seguida. 3) *Aplicação* da política aprendida. Note que, nesta abordagem a exploração não é parte da tarefa de aprendizado. Durante a fase de aplicação no ambiente, que também não faz parte da tarefa de aprendizado (assim como no aprendizado por reforço tradicional), as políticas não são melhoradas (embora os dados coletados possam ser usados posteriormente para um refinamento da política).

No caso mais geral de um problema de aprendizado por reforço em lote com a fase de exploração, não são feitas suposições sobre o procedimento de amostragem das transições. As transições podem ser obtidas usando estratégias arbitrárias e não são, necessariamente, amostras uniformes do espaço de transições. As amostras podem ou não ser coletadas ao longo de trajetórias ligadas, isto é, amostras obtidas a partir de uma sequência de estados sucessores.

Como o agente não interage infinitamente com o ambiente durante o aprendizado, e o conjunto de transições fornecido é finito, não se pode esperar que o planejador consiga encontrar uma política ótima. O objetivo do BRL, portanto, muda de aprender uma política ótima para aprender a melhor política possível a partir dos dados fornecidos (Sascha Lange e Riedmiller, 2012).

A Figura 4.2 faz um paralelo entre planejamento probabilístico, aprendizado por reforço e aprendizado por reforço em lote. No aprendizado por reforço em lote, assim como no aprendizado por reforço tradicional, o agente não possui o modelo completo do ambiente, ou seja, a função $P(s'|s,a)$ é desconhecida e portanto precisa aprender a política. O processo começa com a etapa de exploração do ambiente (Fase A) cujo objetivo é coletar experiências que serão utilizadas na etapa de aprendizado da política (Fase B) que será aplicada no ambiente (Fase C).

4.1 Fundamentos dos Algoritmos de BRL

Métodos de aprendizado *on-line* independentes do modelo, como o *Q-learning*, são atraentes do ponto de vista conceitual e são bem sucedidos quando aplicados a problemas com espaço de estados pequeno e discreto. No entanto, ao aplicá-los em sistemas reais com o espaço de estados maior, e possivelmente contínuo, estes algoritmos deparam-se com alguns fatores limitantes como por exemplo (Sascha Lange e Riedmiller, 2012):

1. A sobrecarga do processo de exploração do ambiente durante o aprendizado torna o aprendizado lento,

2. Problemas de convergência (estabilidade) ao utilizar uma função valor estimada.

Os algoritmos modernos de BRL são capazes de resolver as limitações do AR, mencionadas anteriormente, como segue:

1. **Repetição das experiências para resolver o problema de sobrecarga do processo de exploração.** No aprendizado por reforço puro o agente alterna entre o aprendizado e a exploração em praticamente todo passo. No estado s o agente seleciona e executa a ação a, e então observa o estado subsequente s' e recompensa r, e imediatamente atualiza (aprende) o valor da função Q para o par (s, a). Então o agente volta a explorar com a política atualizada *esquecendo a experiência passada*, representada pela tupla (s, a, r, s'). Este método garante a convergência, no entanto há um grave problema de desempenho com essas atualizações locais. Quando há uma atualização do valor de Q para o par (s_t, a_t) no passo t, isto pode influenciar os valores de Q para os pares (s_{t-1}, a) para todo $a \in A$. No entanto esta mudança não é propagada imediatamente para todos os estados visitados anteriormente, que só serão atualizados na próxima vez que forem visitados (sorteados). Na prática, muitas das interações do Q-Learning são necessárias apenas para propagar a informação. Para superar este problema de desempenho, ji Lin (1992) introduziu a ideia de repetição de experiências nos algoritmos de BRL. A ideia por trás da repetição de experiências é acelerar a convergência do BRL não usando apenas uma vez as transições armazenadas, mas usando-as repetidamente como se fossem novas experiências. Isto permite a propagação da informação sem iteração adicional com o ambiente acelerando assim a convergência do BRL. Na Seção 4.2.1 apresentamos este método com detalhes.

2. **Ajuste para tratar o problema de convergência.** No RL a função valor é atualizada localmente para um estado particular depois de cada experiência, deixando o valor de todos os outros estados inalterados. Atualizações subsequentes podem usar este valor atualizado para sua própria atualização. O RL primeiro realiza uma atualização de programação dinâmica através do passo de *programação dinâmica*:

$$\bar{q}_{s,a} = r + \gamma \max_{a'} Q(s', a'), \tag{4.1}$$

calculando uma estimativa para o valor do par (s, a) atual, $\bar{q}_{s,a}$. Em seguida realiza o passo de aproximação da função $Q(s, a)$ obtendo uma nova estimativa dada por:

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha \bar{q}_{s,a} \tag{4.2}$$

Baird (1995), entre outros, mostrou exemplos em que combinações particulares de *Q-Learning* possuem comportamento instável ou até mesmo conduz a certeza de divergência. Comportamento estável pode ser comprovado apenas para circunstâncias especiais e fazendo-se suposições sobre o sistema e estrutura de recompensa (Sascha Lange e Riedmiller , 2012). Os problemas de estabilidade observados estão relacionados com a interdependência de erros cometidos na Equação 4.2 e a divergência entre o valor estimado da função para o valor ótimo. Enquanto a atualização de programação dinâmica (4.1) tenta diminuir gradualmente a diferença entre $Q(s, a)$ do valor ótimo $Q^*(s, a)$, armazenar o valor atualizado no passo (4.2) pode (re-)introduzir um erro ainda maior (Sascha Lange e Riedmiller, 2012). Este erro de aproximação influencia todas as atualizações subsequentes e pode trabalhar contra a convergência ou até mesmo impedi-la. Gordon (1995) surgiu com a ideia de estender o método de atualização no BRL, a fim de separar a etapa de programação dinâmica (4.1) da etapa de aproximação da função (4.2). A ideia é primeiro aplicar uma atualização de programação dinâmica para todos os pares (s, a), calculando novos valores $\bar{q}_{s,a}$, e então usar aprendizado supervisionado para treinar (*fit*), de forma síncrona, $Q(s, a)$ (ao invés de fazer atualizações locais da Equação 4.2). Este método será explicado com detalhes na Seção 4.2.2.

4.2 Algoritmos

Conforme abordado anteriormente, algoritmos de aprendizado por reforço em lote são projetados para reduzir a quantidade de dados experimentais necessários para o aprendizado, e para situações em que não é possível interagir com o ambiente durante a fase de aprendizado. (nesse caso usa-se experiências passadas). Algoritmos de aprendizado por reforço em lote conseguem extrair mais informações de uma determinada sequência de experiências do que algoritmos *on-line*, já que eles não precisam usar cada experiência para fazer apenas uma atualização da função $Q(s, a)$. Isso torna o aprendizado mais rápido com um número menor de amostras necessárias para obter boas políticas.

O Algoritmo 5 apresenta um modo geral de aprendizado por reforço em lote seguindo o esquema da Figura 4.1. O agente começa com um valor inicial Q_0 para a função Q. A fase de exploração é realizada por m passos com as ações sendo escolhidas seguindo algum critério, por exemplo $\epsilon - Greedy$. As experiências são guardadas, e a sequência de experiências, D, são então utilizadas como um lote para o aprendizado da nova política. O laço que vai da linha 6 até a linha 14 implementa a etapa de exploração do ambiente para obtenção das experiências que serão usadas na etapa de aprendizado. Para os problemas em que o agente não pode interagir com o ambiente e o conjunto de transições que serão usadas para o aprendizado já existe, esta etapa não é executada. É na linha 16 que ocorre o aprendizado ao executar um algoritmo de aprendizado por reforço em lote passando como parâmetro o banco de dados (lote) de experiências.

Algoritmo 5: BRL(Q_0, m) adaptado de (Kalyanakrishnan e Stone, 2007)

Entrada: Q_0: uma política inicial,
m: o tamanho do lote

Saída: Q: a melhor política

1 $Q \leftarrow Q_0$;
2 $D \leftarrow \{\}$
3 $experiencias \leftarrow 0$
4 // Exploração do ambiente
5 **enquanto** $experiencias < m$ **faça**
6 $\quad experiencias \leftarrow experiencias + 1$
7 $\quad i \leftarrow experiencias$
8 $\quad s_i \leftarrow estadoDoAmbiente()$
9 $\quad a_i \leftarrow \epsilon - Greedy(s_i)$
10 $\quad (s_{i+1,r_i}) \leftarrow$ executaAcao(a_i)
11 $\quad d_i \leftarrow (s_i, a_i, r_i, s_{i+1})$
12 $\quad D \leftarrow D \cup \{d_i\}$
13 **fim**
14 // Aprendizado
15 $Q \leftarrow$ "*Executa um algoritmo de aprendizado em lote*"
16 Devolve Q

Dois métodos diferentes de implementar a função de aprendizado por reforço em lote (linha 16 do Algoritmo 5) são respectivamente **BRL Com Repetição de Experiências** (*Experience Replay*) (ji Lin, 1992)) e **BRL Ajustado** (*Fitted Q-Iteration*) (Ernst *et al.*, 2005)) apresentados nas seções a seguir.

4.2.1 BRL Com Repetição de Experiências

No aprendizado por reforço em lote com repetições de experiências (*Experience Replay*) (ji Lin, 1992)), ao invés de utilizar cada experiência para uma única atualização, usamos o lote de experiências repetidamente. Conforme mostrado no Algoritmo 6, o lote D de experiências armazenadas é utilizado k vezes (laço entre as linhas 3 e 9 do algoritmo), e dentro de cada passo uma atualização idêntica a do *Q-learning* é feita para cada experiência (linha 6 do algoritmo).

Algoritmo 6: BRLRepeticaoDeExperiencias(Q_0, D) adaptado de (Kalyanakrishnan e Stone, 2007)

Entrada: Q_0: uma política inicial,
D: o conjunto de experiências (lote)

Saída: Q: a melhor política possível com base nas experiências coletadas

1 $Q \leftarrow Q_0$;
2 $i \leftarrow 0$;
3 **enquanto** $i < k$ **faça**
4 **para cada** $t \in [1, ...|D|]$ **faça**
5   // $d_t = (s_t, a_t, r_t, s_{t+1})$
6   $Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q_t(s_{t+1}, a))$
7 **fim**
8 $i \leftarrow i + 1$;
9 **fim**
10 Devolve Q

4.2.2 BRL Ajustado

Proposto por Damien Ernst, o algoritmo BRL Ajustado *(Fitted Q-Iteration)* (Ernst *et al.* , 2005) é o algoritmo mais popular de aprendizado por reforço em lote. Ele pode ser visto como o 'Q-Learning do BRL'. Dado um conjunto fixo $D = \{(s_t, a_t, r_{t+1}, s_{t+1}|t = 1, ..., p)\}$ de p transições (s, a, r, s'), o algoritmo começa com uma aproximação inicial da função Q , $Q_0(s, a) = q_0$ (Ernst *et al.* (2005) usou $q_0 = 0$) para todo $(s, a) \in S \times A$. BRL Ajustado itera sobre os seguintes passos:

1. Para cada transição $(s, a, r, s') \in D$, calcula-se:

$$T_{s,a} \leftarrow r + \gamma \max_{a'} Q_{t-1}(s', a') \tag{4.3}$$

   e armazena a tupla $(s, a, T_{s,a})$ num conjunto P.

2. Usa aprendizado supervisionado para calcular Q_t

O Algoritmo **BRL Ajustado** é mostrado no Algoritmo 7. Note que no laço compreendido entre as linhas 4 e 7 o algoritmo calcula novas estimativas para o valor da função Q usando o passo de programação dinâmica, porém o valor da função Q não é atualizado e as estimativas são armazenadas. Posteriormente no laço compreendido entre as linhas 11 e 14 o valor de Q é atualizado através do passo de aproximação da função (linha 13). Isto é feito passando por todas

as experiências do lote por k vezes (laço entre as linhas 10 e 15).

Algoritmo 7: BRLAjustado(Q_0, D) adaptado de (Kalyanakrishnan e Stone, 2007)

Entrada: Q_0: uma política inicial,
D: o conjunto de experiências (lote)

Saída: Q: a melhor política possível com base nas experiências coletadas

1 $t \leftarrow 1$;
2 **enquanto** *tleqT* **faça**
3 //Encontra uma estimativa para $Q(s_i, a_i)$ para cada d_i baseado em Q_{t-i}
4 **para cada** $i \in [1, ...|D|]$ **faça**
5 // $d_i = (s_i, a_i, r_i, s_{i+1})$
6 $T_i \leftarrow r_i + \gamma \max_a Q_{t-1}(s_{i+1}, a)$ //Passo de programação dinâmica
7 **fim**
8 //Ajusta os valores de Q usando aprendizado supervisionado
9 $Q_t \leftarrow Q_0$
10 **para cada** *iteracao = 1 até k* **faça**
11 **para cada** $i \in [1, ...|D|]$ **faça**
12 // $d_i = (s_i, a_i, r_i, s_{i+1})$
13 $Q_t(s_i, a_i) \leftarrow (1 - \alpha_{supervisionado})Q_t(s_i, a_i) + \alpha_{supervisionado}(T_i)$ //Passo de aproximação
14 **fim**
15 **fim**
16 $t \leftarrow t + 1$;
17 **fim**
18 Devolve Q

4.3 Aprendizado por reforço em lote crescente

No aprendizado por reforço em lote, regiões importantes do espaço de estados, por exemplo, regiões próximas do estado meta, podem não estar cobertas pelo conjunto de amostras. Neste caso, não é possível aprender boas políticas. Na prática, a fase de exploração tem um impacto importante na qualidade da política aprendida. Obviamente a distribuição das transições usadas no processo de aprendizado precisa ser próxima da probabilidade de transição real do ambiente para que o agente possa calcular boas políticas, isto é, para que que o aprendizado por reforço em lote convirja. Caso não seja possível convergir para úma política ótima, é possível melhorar a política encontrada obtendo-se um novo lote de amostras do ambiente.

Por esse motivo, é comum usar uma variação do aprendizado por reforço em lote que alterna entre o aprendizado por reforço puro e o aprendizado em lote. A ideia principal desse tipo de aprendizado é alternar entre a fase de aprendizado e a exploração em que o conjunto de amostras cresce com a interação do agente no ambiente até a convergência da função Q. Esse Algoritmo é chamado de *aprendizado por reforço em lote crescente*.

A Figura 4.3 mostra as fases do aprendizado por reforço em lote crescente. O processo de aprendizado por reforço em lote crescente tem as mesmas três fases, como o processo de aprendizado por reforço em lote puro apresentado na Figura 4.1. A diferença é que o processo de aprendizado por reforço em lote crescente pode retornar à fase de exploração incrementando o conjunto de amostras, seguindo uma política de exploração. Assim como no aprendizado em lote tradicional, o agente começa com uma estimativa inicial da função $Q = Q_0$, executa m ações escolhidas seguindo algum critério, por exemplo $\epsilon - Greedy$. As experiências são guardadas, e

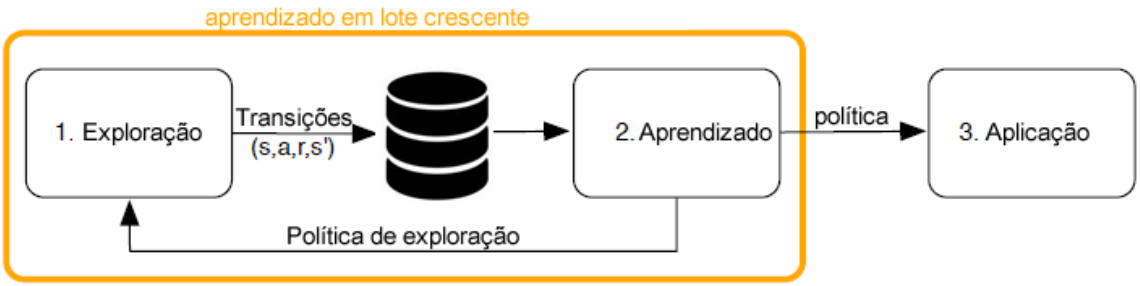


Figura 4.3: *Fases do aprendizado por reforço em lote crescente adaptado de Sascha Lange e Riedmiller (2012).*

a sequência D de experiências, são então utilizadas no cálculo da nova política. Esta política é novamente utilizada para gerar um novo lote de experiências, e o ciclo continua até que Q tenha convergido.

Algoritmo 8: BRLCrescente(Q_0, m) adaptado de (Kalyanakrishnan e Stone, 2007)

Entrada: Q_0: uma política inicial,
m: o tamanho do lote

Saída: Q: a política ótima

1 $Q \leftarrow Q_0$;
2 $D \leftarrow \{\}$
3 **enquanto** *Q não convergiu* **faça**
4 $\quad i \leftarrow 0$
5 $\quad experiencias \leftarrow 0$
6 $\quad$ // Exploração do ambiente para incrementar o lote
7 $\quad$ **enquanto** $experiencias < m$ **faça**
8 $\quad\quad i \leftarrow i + 1$
9 $\quad\quad s_i \leftarrow estadoDoAmbiente()$
10 $\quad\quad experiencias \leftarrow experiencias + 1$
11 $\quad\quad a_i \leftarrow \epsilon - Greedy(s_i)$
12 $\quad\quad (s_{i+1,r_i}) \leftarrow$ executaAcao(a_i)
13 $\quad\quad d_i \leftarrow (s_i, a_i, r_i, s_{i+1})$
14 $\quad\quad D \leftarrow D \cup \{d_i\}$
15 $\quad$ **fim**
16 $\quad$ // Aprendizado
17 $\quad Q \leftarrow$ "*Executa um algoritmo de aprendizado em lote*"
18 **fim**
19 Devolve Q

O Algoritmo 8 mostra como seria uma adaptação do Algoritmo 5 para o aprendizado por reforço em lote crescente. Diferente do Algoritmo 5, o Algoritmo 8 alterna entre o passo de exploração e o passo de aprendizado até que Q tenha convergido para uma política ótima (laço entre as linhas 3 e 18). Um possível critério de parada para o algoritmo é checar se para cada par (s, a) a alteração de $Q(s, a)$ foi inferior a um ϵ pequeno.

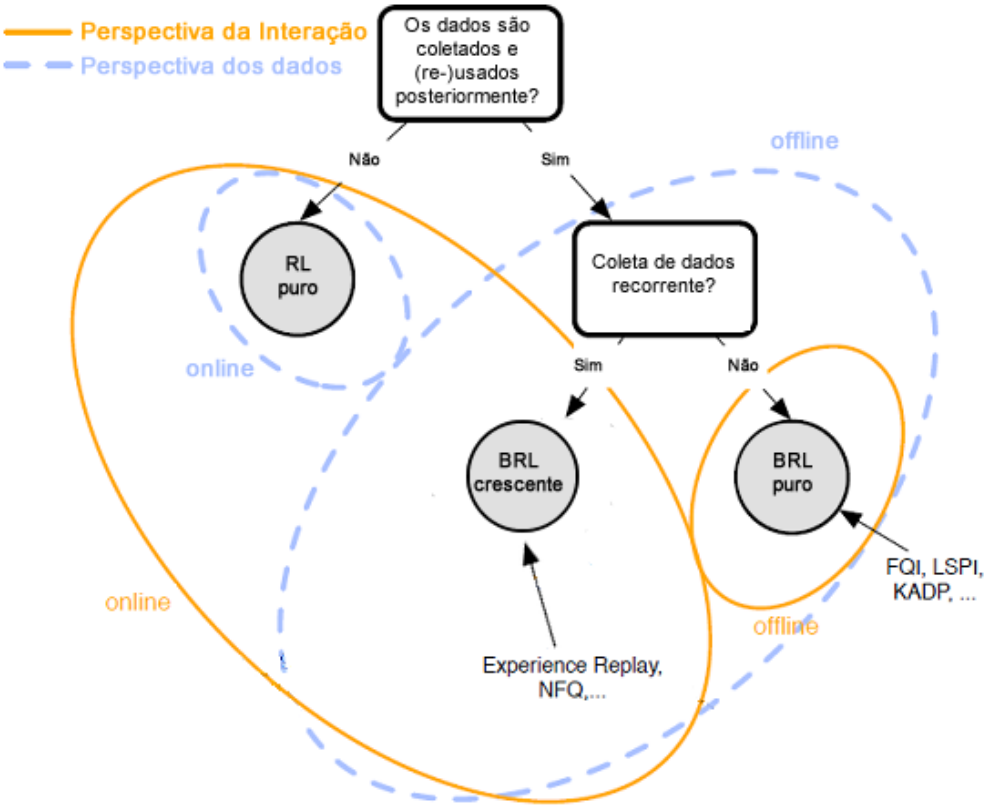


Figura 4.4: *Classificação dos algoritmos de aprendizado por reforço em lote vs. aprendizado por reforço tradicional de acordo com a perspectiva de uso dos dados e interação ((Sascha Lange e Riedmiller, 2012)).*

4.4 Classificação dos algoritmos

A Figura 4.4 propõe uma classificação dos algoritmos de aprendizado por reforço entre algoritmos *online* e *offline*. De um lado temos algoritmos *online* puros de RL como o *Q-learning* clássico. No lado oposto temos algoritmos de aprendizado em lote puros que funcionam completamente *offline* através de um conjunto fixo de transições. No meio destes extremos há uma série de outros algoritmos que, dependendo da perspectiva, podem ser classificadas como algoritmos em lote, *online* (linha tracejada da Figura 4.4) ou *offline* (linha cheia da Figura 4.4). Por exemplo, a abordagem de aprendizado em lote crescente poderia ser classificada como um método *online*, pois interage com o ambiente como um método *online* e melhora sua política de forma incremental com as novas experiências disponíveis, mas também pode ser visto como um método *offline* pois armazena toda a experiência e usa métodos de aprendizado em lote para aprender com as novas observações.

Capítulo 5

Estudo de caso: Modelando o Problema do Vendedor de Impressoras

A oferta de soluções ou produtos sofisticados dependem de um processo de venda complexo em que uma série de ações (que chamaremos de *ações de venda*) devem ser executadas pelos vendedores com o objetivo de influenciar o cliente à compra. Como o resultado das ações pode ser incerto, a escolha da melhor sequência de ações de venda pode ser vista como um problema de processo estocástico de decisão sequencial.

5.1 Processos de Venda

Um processo de venda pode variar de acordo com o vendedor, com o produto a ser comercializado e com o cliente, porém, em geral um processo de venda passa por algumas etapas básicas descritas a seguir:

1. **Prospecção**: O objetivo dessa etapa é buscar clientes em potencial (*prospects*), isto é, clientes que possuam características que os apontem como possíveis consumidores do produto que está sendo comercializado.

2. **Qualificação do cliente**: A etapa de prospecção tende a ser pouco criteriosa. Ter uma lista de clientes com características de potenciais compradores não é suficiente. Por exemplo, se o produto a ser comercializado for impressoras de *outdoors*, é natural procurar na etapa de prospecção por empresas de propaganda e marketing. Porém, nem toda empresa de propaganda e marketing trabalha com esse tipo de mídia. A etapa de qualificação tem como objetivo fazer o primeiro contato com o cliente afim de obter mais informações e descobrir se ele tem real necessidade de adquirir os produtos comercializados.

3. **Definição das necessidades do cliente**: Nessa etapa o vendedor deve procurar entender com detalhes as necessidades de um cliente qualificado para oferecer o produto mais adequado, garantindo assim a satisfação do cliente após a venda.

4. **Elaboração de propostas**: É uma das etapas mais críticas no processo de venda. Uma proposta precisa resolver a necessidade do cliente de forma satisfatória e precisa ter um custo compatível com sua situação. Nem sempre a necessidade do cliente será satisfeita pelo produto mais barato, e é comum ter soluções parecidas de produtos concorrentes a preços inferiores. A proposta precisa defender que está sendo oferecido um produto ideal e melhor que o produto concorrente.

5. **Negociação**: Depois de convencer o cliente de que o produto oferecido é o que melhor atende às suas necessidades, o processo entra na etapa de negociações. Esta etapa fica muito centralizada na questão financeira. O vendedor nessa etapa terá que oferecer descontos ou

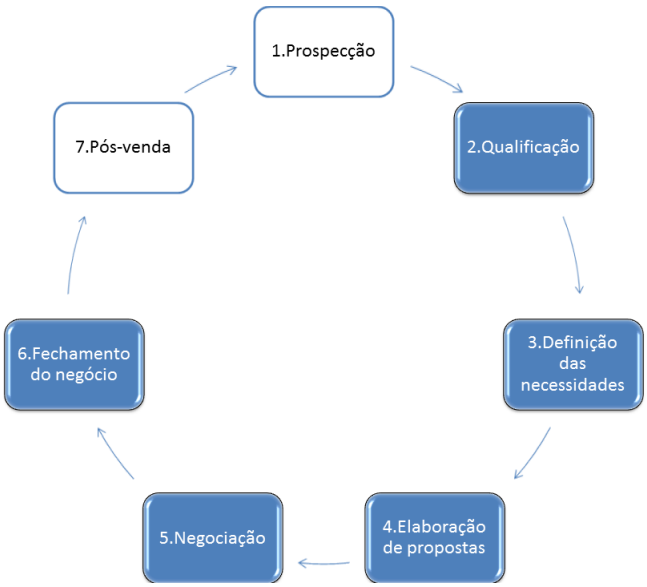


Figura 5.1: *Ciclo de Vendas: etapas básicas de um processo de venda.*

condições favoráveis ao comprador, mas também precisa garantir a rentabilidade de seus produtos.

6. **Fechamento do negócio**: Essa é a etapa em que o cliente decide pela compra do produto. Embora pareça uma etapa simples onde a venda está totalmente concluída, na verdade há muitas razões para o cliente desistir ou adiar a compra nessa etapa. O cliente pode desconfiar de que as condições apresentadas não são as melhores possíveis, pode desconfiar das afirmações do vendedor, pode analisar a possibilidade de encontrar condições melhores nos concorrentes ou simplesmente pode ter entraves financeiros que inviabilizam a compra. É normal nessa etapa retroceder para outras etapas do processo ou até mesmo perder definitivamente a venda.

7. **Pós-venda**: Nessa etapa a venda foi concluída com sucesso, porém é importante que o vendedor mantenha contato com o cliente para verificar se ele está satisfeito e se surgiram novas necessidades.

A Figura 5.1 mostra as dependências entre as etapas básicas de um processo de venda. É importante notar que em qualquer uma dessas etapas o processo pode ser interrompido com a desistência de compra do cliente. Cada uma dessas etapas envolve uma ou mais *ações de venda*. As caixas escuras indicam estágios do processo de venda em que o vendedor deve tomar decisões críticas e que serão tratadas nesse trabalho.

5.2 Venda de impressoras Plotters e Sistema de Acompanhamento de Negociações (Forecast)

Plotters de Impressão são impressoras destinadas a impressões em grandes dimensões e são muito usadas em empresas de arquitetura para imprimir plantas e projetos de engenharia. Também são muito usadas pelo mercado de comunicação visual na impressão de posters, banners e painéis, e na indústria têxtil para impressão em tecidos.

Existem modelos específicos de impressoras plotters para cada necessidade. De modo geral os modelos distinguem-se pelo tipo de aplicação (impressão Digital, geoprocessamento, ...), tipo de mídia usada na impressão (lona, tecido, papel fotográfico, ...), largura de impressão, tecnologia de impressão (tinta a base de água, tinta a base de látex, ...) e quantidades de cores.

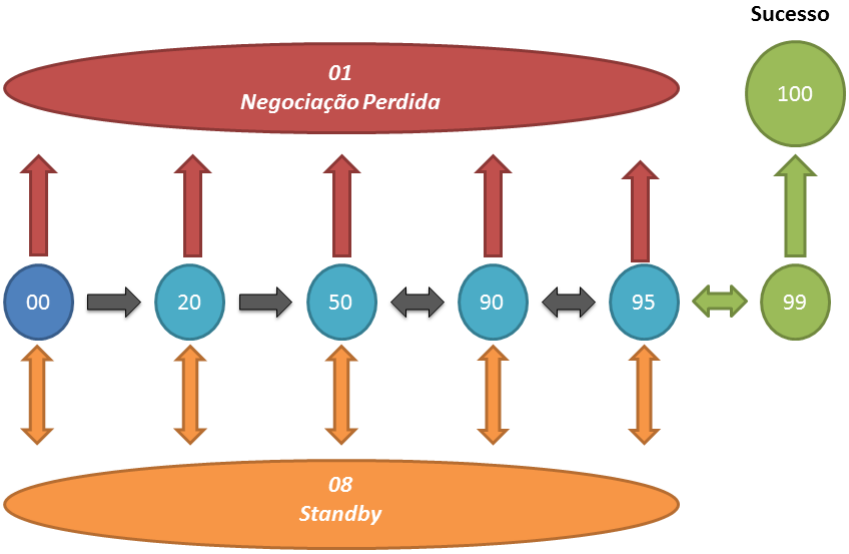


Figura 5.2: *Ciclo de Vendas do Programa Big Impression. Na etapa 08 (Standby) o vendedor aguarda por um tempo antes de retomar o processo de negociação.*

Cada tipo de impressão exige características específicas da impressora plotter. Por exemplo, um engenheiro ao imprimir uma planta de construção não precisa de fidelidade nas cores, mas precisa de precisão nos traços. Impressão de fotografia já exige fidelidade nas cores. Impressão de adesivos para carros exigem mídias específicas e tinta que não dissolve em água. Cada característica influencia diretamente o custo do equipamento, o tamanho, as condições necessárias para instalação e os conhecimentos técnicos exigidos para o manuseio.

De uma forma geral, podemos dividir os tipos de impressoras plotters em 3 grandes grupos de acordo com o mercado que ela atende:

- **TÉCNICO**: plotters para arquitetos e engenheiros,
- **CRIATIVO**: plotters para fotógrafos e designers,
- **PSP**: plotters para prestadores de serviço de impressão,

Com tudo isso, é possível deduzir que o processo de venda de uma impressora plotter é bastante complexo, possui um ciclo de vendas característico e tempo de negociação extenso.

O *Big Impression* é um programa especializado no mercado de impressão em grandes formatos, que atende o público SMB (pequenas e médias empresas) e profissionais liberais. Surgiu em 2003 após a percepção da necessidade de mercado de ter especialistas no segmento de impressão em grandes formatos. Hoje, o programa *Big Impression* é um canal mais especializado, representando o maior número de vendas de impressoras Plotters HP no território brasileiro através de revendedores espalhados por todo território nacional.

Com o objetivo de controlar as negociações e profissionalizar o relacionamento comercial entre as revendas parceiras e os clientes, foi criado um sistema de controle de negócios denominado *Forecast*. O *Forecast* tornou-se uma importante ferramenta de apoio ao processo de vendas de impressoras *plotters* pois permitiu ter controle e previsão da vendas, porém ela não dá suporte à tomada de decisão do vendedor, que é uma das propostas desse trabalho de mestrado.

O processo de venda de *ploters*, conforme está modelado no sistema *Forecast*, é feito em etapas denominadas *estágios de negociação*. Em cada estágio a revenda deve executar algumas ações e reportar ao sistema o resultado da execução das ações avançando, retrocedendo, adiando ou terminando o processo de negociação. A aplicabilidade das ações no sistema em geral depende de pré-requisitos e o resultado das ações em geral depende de fatores externos ao sistema e não são determinísticos. A Figura 5.2 mostra o ciclo de vendas implementado pelo sistema Forecast.

Estágio 00 - Primeiro contato / Qualificação

O estágio *00* corresponde à etapa de qualificação do cliente (Etapa 2 da Figura 5.1). Neste estágio a revenda precisa realizar o contato inicial com o cliente para conhecer suas necessidades e definir quais produtos serão oferecidos ao cliente. Convidar o cliente para uma demonstração do produto pode ser importante se o cliente não conhece o produto oferecido ou tiver dúvidas se o produto atenderá suas necessidades.

Estágio 20 - Conhecer melhor o Cliente

Este estágio corresponde à etapa de Definição das necessidades do cliente (Etapa 3 da Figura 5.1). Neste estágio espera-se que o vendedor mantenha contato e conheça melhor o cliente, suas necessidades mais detalhadamente, suas instalações e quem serão os responsáveis por aprovar a compra do produto (por exemplo o diretor financeiro ou o dono da empresa).

Estágio 50 - Proposta Comercial

Após conhecer o cliente, suas necessidades e suas condições, chega o momento de elaborar uma *proposta comercial* (Etapa 4 da Figura 5.1). É natural que o cliente esteja negociando com outros fornecedores e por isso é essencial tentar conhecer quais são os concorrentes no processo, quais produtos estão sendo oferecidos e com quais condições. Fazer demonstrações para mostrar a superioridade do produto oferecido em relação ao concorrente pode ser necessário caso o produto oferecido tenha um preço superior.

Estágio 90 - Aceite da proposta comercial

O objetivo deste estágio (Etapa 5 da Figura 5.1) é aguardar o aceite do cliente em relação a proposta elaborada no estágio anterior. É comum neste estágio o cliente recusar a proposta oferecida e o vendedor deve decidir entre voltar para o estágio anterior para elaborar uma nova proposta ou desistir da venda.

Oferecer descontos, condições de pagamento e serviços agregados podem ser necessários nessa fase, se por algum motivo o cliente perder o interesse pela compra por causa de preço. Para oferecer descontos o vendedor precisa solicitar autorização via o sistema Forecast, justificar o pedido de desconto e aguardar o deferimento da solicitação realizado pelo gerente do programa

Estágio 95 - Pedido

Neste estágio o cliente optou pela compra (Etapa 6 da Figura 5.1). Caso necessário o vendedor pode oferecer ao cliente o financiamento do produto. Neste caso, cabe ao vendedor aguardar a aprovação do financiamento. Como algumas *plotters* possuem algumas pré-condições de instalação, também é importante que o vendedor cheque com o cliente se as pré-condições são atendidas.

Estágio 99 - Entrega

Este estágio corresponde a etapa 7 da Figura 5.1. Neste estágio cabe ao vendedor apenas acompanhar o processo de faturamento de entrega. Não há nenhuma ação para ser realizada.

Estágio 100 - Negociação concluída com sucesso

Este estágio também pode ser visto como parte da etapa 7 da Figura 5.1. Chegar neste estágio significa que a venda foi realizada com sucesso. Para o sistema o processo acaba por aqui, mas é muito importante que o vendedor mantenha contato constante com o cliente para detectar novas oportunidades.

Estágio 01 - Negócio perdido

Existem muitas razões para que o negócio seja perdido, por exemplo: o cliente pode não ser um cliente potencial para os produtos oferecidos, o cliente pode não ter condições para efetuar a compra ou ainda o negócio pode ter sido perdido para um concorrente. Para todos estes casos o vendedor precisa finalizar o processo.

Estágio 08 - Standby

Em alguns casos o vendedor detecta que o cliente não tem potencial para compra no momento mas possivelmente terá no futuro. Às vezes o próprio cliente sinaliza a possibilidade de compra futura. Para estes casos, o vendedor pode deixar a negociação inativa por um determinado período de tempo e reativá-la com todo o histórico no futuro.

5.3 Modelando o problema do vendedor de impressoras como um problema de aprendizado por reforço em lote

Um problema enfrentado pelo programa Big Impression é a rotatividade de revendas e vendedores. Frequentemente revendas são credenciadas ou descredenciadas do programa. Dentro das revendas é ainda mais comum a rotatividades de vendedores. Quando ocorre o desligamento de uma boa revenda ou um bom vendedor, este conhecimento e experiência é perdido.

Apesar do *programa Big Impression* responder por mais de 50% das plotters HP vendidas no Brasil, menos de 10% das negociações terminam em vendas. Considerando que somente os clientes qualificados são inseridos no sistema *Forecast*, isto é, após passar pela etapa de *qualificação*, espera-se que o volume de vendas possa aumentar caso ações corretas sejam aplicadas para cada cliente. Dado o grande número de revendas e vendedores que utilizam o sistema Forecast, uma maneira de oferecer suporte à escolha correta de ações de venda é através de um sistema automático que aprende a melhor política com base no histórico de negociações armazenadas no banco de dados do *Forecast*. Para isso usaremos a técnica descrita no Capítulo 4 e nas Seções 5.3.1, 5.3.2 e 5.3.3 mostramos como, a partir do sistema *Forecast*, modelamos estados, ações e a função recompensa do MDP para o qual queremos aprender a política ótima.

5.3.1 Identificando Estados

Há muitos fatores que podem interferir no processo de vendas de impressoras. Alguns são óbvios, como por exemplo a situação financeira do cliente. Porém, há muitos fatores desconhecidos que influenciam esse processo e que precisam de alguma forma ser descobertos e mapeados no modelo.

O sistema Forecast possui entre 20 e 30 variáveis mapeadas no processo de venda. Portanto é preciso identificar quais destas variáveis são relevantes para a descrição de estados e, consequentemente, para tomada de decisão. Numa análise prévia da base de dados do sistema Forecast com o histórico de negociações, descobrimos algumas das variáveis que certamente influenciam no processo, e que serão utilizadas na descrição de estados (i.e., definem variáveis de estado). Entre elas:

- **Segmento**: como explicado anteriormente, *plotters* HP são divididas em 3 grupos chamados de *segmentos*, técnico, criativo e PSP.Existe uma significativa diferença de preços e procura entre os segmentos, O segmento PSP possui as máquinas maiores e mais caras e o segmento CRIATIVO possui as máquinas menos procuradas.

- **PF/PJ**: define se o cliente é pessoa física (PF) ou pessoa jurídica (PJ). Pessoas jurídicas possuem taxas de crédito diferenciadas, além disso, em negociações com pessoas físicas o

estagio_atual	Variável que representa o estágio em que a negociação se encontra, e pode assumir os valores {00, 01, 08, 20, 50, 90, 95, 99}
estagio_anterior	Esta variável é usada somente para a situação de *standby* (Estágio 08), e faz com que a negociação, ao ser reativada, volte automaticamente para o estágio de onde saiu.
segmento	Segmento do produto oferecido ao cliente (técnico, criativo ou PSP)
pj	Variável booleana. Indica se o cliente é uma empresa (1) ou pessoa física (0).
regiao	Região onde o cliente está localizado (NORTE, NORDESTE, CENTRO-OESTE, SUDESTE, SUL, NAO_IDENTIFICADA)
demonstracao	Variável booleana. Indica se o cliente foi chamado para uma demonstração
standby	Variável booleana. Indica se a negociação ficou em *standby*.
proposta	Variável booleana. Indica se foi apresentada uma proposta ao cliente
desconto	Variável booleana. Indica se foi oferecido desconto ao cliente.
desconto_aprovado	Variável booleana. Indica se o desconto foi aprovado pelo gerente do programa.
aceitou_proposta	Variável booleana. Indica se o cliente aceitou alguma proposta apresentada.
financiamento	Variável booleana. Indica se foi solicitado financiamento do produto.
financiamento_aprovado	Variável booleana. Indica se o financiamento foi aprovado.

Tabela 5.1: *Lista de variáveis de estado consideradas relevantes e utilizadas no modelo do problema do vendedor de impressoras.*

vendedor negocia diretamente com o único responsável pela compra, enquanto em negociações com pessoas jurídicas o vendedor nem sempre consegue negociar diretamente com todos os responsáveis.

- **Região**: define a região de localização do cliente, de acordo com a divisão regional do Brasil estabelecida pelo IBGE. Numa análise prévia da base de históricos percebemos que o processo varia dependendo da região onde reside o cliente (PF) ou onde se localiza a empresa (PJ). Diferenças no mercado de *plotter* e diferenças tributárias podem ser explicações para isto.

- **Estágio da negociação**: conforme abordado anteriormente, uma negociação é dividida em estágios. Algumas ações só podem ser aplicadas dentro de determinados estágios, por exemplo, uma proposta comercial só pode ser oferecida ao cliente no Estágio 50 da negociação.

A Tabela 5.1 descreve o conjunto completo de variáveis de estado consideradas na modelagem de estado do problema do vendedor de impressoras. Numa análise superficial, esse conjunto de variáveis corresponde a 2^{33} estados possíveis. No entanto, na base de dados do Forecast foram observadas apenas 1636 estados.

5.3.2 Identificando Ações

A seguir, apresentamos a lista das ações de venda consideradas no modelo.

1. **oferecer_produto_criativo**: Ação de oferecer ao cliente um produto do segmento *Criativo* que só é aplicada nos estados em que a negociação está no 00 (variável de estado *estado_atual* é 00).

2. **oferecer_produto_tecnico**: Ação de oferecer ao cliente um produto do segmento *Técnico* que só é aplicada nos estados em que a negociação está no 00 (variável de estado *estado_atual* é 00).

3. **oferecer_produto_PSP**: Ação de oferecer ao cliente um produto do segmento *PSP* que só é aplicada nos estados em que a negociação está no 00 (variável de estado *estado_atual* é 00).

4. **conhecer_cliente**: Ação de alterar o processo para o estágio de negociação 20 em que o objetivo do vendedor é conhecer melhor o cliente. Tem efeito determinístico, altera apenas a variável de estado *estagio_atual* e pode ser executada nos estados em que *estagio_atual* é *00* e *segmento* não é vazio (ou seja, o vendedor já escolheu qual produto oferecer ao cliente).

5. **elaborar_proposta**: Ação de alterar o processo para o estágio 50 em que o objetivo do vendedor é elaborar uma proposta comercial para o cliente. Tem efeito determinístico, altera apenas a variável de estado *estagio_atual* e pode ser executada nos estados em que *estagio_atual* é *20* ou *90*.

6. **negociar**: Ação de alterar o processo para o estágio 90 em que o objetivo do vendedor é negociar com o cliente. Tem efeito determinístico, altera apenas a variável de estado *estagio_atual* e pode ser executada nos estados em que *estagio_atual* é *50* ou *95* e que *proposta* é *VERDADEIRO*.

7. **fechar_pedido**: Ação de alterar o processo para o estágio 95 em que o objetivo do vendedor é fechar o pedido. Tem efeito determinístico, altera apenas a variável de estado *estagio_atual* e pode ser executada nos estados em que *estagio_atual* é *90* ou *99*.

8. **entregar_produto**: Ação de alterar o processo para o estágio 99. Ao chegar nesse estágio considera-se que a venda foi efetuada com sucesso. Tem efeito determinístico, altera apenas a variável de estado *estagio_atual* e pode ser executada nos estados em que *proposta* é *VERDADEIRO* e *aceitou_proposta* é *VERDADEIRO*.

9. **standby**: Ação de alterar a negociação para o estágio 01 (*Standby*) em que a negociação fica parada. Tem efeito determinístico, altera as variáves de estado *estagio_anterior* para o valor de *estagio_atual*, altera o valor de *estagio_atual* para 08 e a variável *standby* para *VERDADEIRO*. Pode ser executada em qualquer estado não terminal.

10. **reativar_negociacao**: Ação de reativar uma negociação que está em *standby*. Tem efeito determinístico e altera a variável de estado *estagio_atual* para o valor de *estagio_anterior*. Pode ser executada apenas nos estados em que *estagio_atual* é 08.

11. **agendar.demonstracao**: Ação de convidar o cliente para uma demonstração do produto. Pode ser executada em qualquer estado do sistema. Tem efeito determinístico e altera o valor da variável de estado *demonstracao* para *VERDADEIRO*.

12. **desistir**: Ação de desistir de uma negociação, necessária para os casos em que o cliente não tem interesse ou não tem potencial para a compra do produto. Pode ser executada em qualquer estado não terminal, ou seja, quando a variavel de estado *estagio_atual* não é 01 nem 99. Esta ação tem efeito determinístico e altera o valor da variável *estagio_atual* para 01 .

13. **proposta.sem.descontro**: Ação de oferecer ao cliente uma proposta sem qualquer desconto ou benefício. Pode ser executada nos estados em que *estagio_ atual* é *50* e tem efeito probabilístico. Com probabilidade 1 altera a variável de estado *proposta* para *VERDADEIRO* e com uma probabilidade p desconhecida altera a variável de estado *aceitou_ proposta* para *VERDADEIRO* ou *FALSO*.

14. **oferecer.desconto**: Ação de montar uma proposta comercial com desconto para apresentar ao cliente. O desconto pode ser reprovado pelo gerente do projeto. Pode ser executada nos estados em que *estagio_ atual* é *50* e tem efeito probabilístico. Com probabilidade 1 altera as variáveis de estado *proposta* e *desconto* para *VERDADEIRO*, com uma probabilidade $p1$ desconhecida altera a variável de estado *desconto_ aprovado* para *VERDADEIRO* ou *FALSO*, e com uma probabilidade $p2$ também desconhecida altera a variável de estado *aceitou_ proposta* para *VERDADEIRO* ou *FALSO*.

15. **oferecer.financiamento**: Ação de oferecer opções de financiamentos para o cliente. Necessária apenas em casos em que o cliente não tem condições de pagar o produto a vista. O financiamento pode ser reprovado pela instituição financeira. Pode ser executada nos estados em que *estagio_ atual* é *50* e tem efeito probabilístico. Com probabilidade 1 altera as variáveis de estado *proposta* e *financiamento* para *VERDADEIRO*, com uma probabilidade $p1$ desconhecida altera a variável de estado *financiamento_ aprovado* para *VERDADEIRO* ou *FALSO*, e com uma probabilidade $p2$ também desconhecida altera a variável de estado *aceitou_ proposta* para *VERDADEIRO* ou *FALSO*.

5.3.3 Definição da função recompensa

As recompensas não são dadas pelo sistema *Forecast* e precisam ser definidas. No entanto isso não pode ser feito de maneira arbitrária. Uma ação que gera uma recompensa muito baixa tem tendência a ser evitada, enquanto uma ação que gera uma recompensa alta tende a ser considerada como a melhor. A definição das recompensas precisa ser fiel à realidade do problema pois irá influenciar diretamente no processo de aprendizado da melhor política.

A Tabela 5.2 mostra as recompensas definidas para o problema do vendedor de impressoras tratado neste trabalho. Os valores das recompensas foram definidos com a ajuda de um especialista conhecedor do problema. Note que a única recompensa positiva, de valor alto, é dada à ação entregar_produto que indica sucesso da venda.

Açao	Recompensa
oferecer_produto_criativo	-1
oferecer_produto_técnico	-1
oferecer_produto_PSP	-1
conhecer_cliente	-1
elaborar_proposta	-1
negociar	-1
fechar_pedido	-1
entregar_produto	100
standby	-10
reativar_negociacao	-1
agendar_demonstracao	-5
desistir	-1000
proposta_sem_descontro	-1
oferecer_desconto	-10
oferecer_financiamento	-1

Tabela 5.2: *Recompensas definidas para o problema do vendedor de impressoras*

Capítulo 6

Aprendizado por reforço em lote para o Problema do Vendedor de Impressoras

Para usar aprendizado por reforço em lote no problema do vendedor de impressoras usando experiências extraídas do banco de dados do *Forecast*, foi preciso fazer a reconstrução do processo de negociação e a limpeza dos dados (por exemplo, eliminar informações desnecessárias para o processo).

No sistema *Forecast* os dados estão modelados como *Workflow* (van Der Aalst *et al.*, 2003). A primeira etapa para resolver o problema é extrair os dados e convertê-los para um conjunto de tuplas $P = \{< p, s, a, s', r, h >\}$, em que p é o identificador do processo, s e $s' \in S$, estados do sistema, h o passo, ou horizonte, do processo em que a ação foi executada, a é a ação executada pelo vendedor no processo p no passo h e r a recompensa recebida na transição.

A Figura 6.1 mostra o modelo de workflow implementado pelo *Forecast* em que é possível identificar quais os valores das variáveis de estado e quais foram as ações executadas pelo vendedor em cada instante da negociação. No então não é possível saber qual era o estado completo da negociação no momento em que a ação foi executada. Portanto, o primeiro passo para converter os dados em um conjunto de transições (experiências) é reconstruir o processo de negociação ao longo do tempo de maneira que seja possível identificar o estado completo da negociação no momento em que cada ação foi executada.

6.1 Reconstruindo o processo: geração do conjunto de experiências

O primeiro passo para reconstruir o processo foi criar uma tabela com a identificação da negociação, a descrição da ação, a data em que a ação foi executada e as variáveis de estado alteradas pela ação. A Figura 6.3 mostra um exemplo dessa tabela auxiliar usada para reconstruir o processo. Com esta tabela criada, o segundo passo é separar cada negociação e ordenar as ações pela data de execução (para identificar o horizonte de cada ação do processo).

Com as negociações separadas e as ações ordenadas, o próximo passo é identificar o estado inicial de cada negociação. O estado inicial é dado pelas características do cliente: a região e se o cliente é pessoa física ou jurídica. São 12 os possíveis estados iniciais dados pela combinação das 6 regiões possíveis, e se o cliente é PF ou PJ.

Com o estado inicial de cada negociação identificado e com a tabela contendo as ações executadas e seus efeitos ordenadas pela data de execução, é possível reconstruir todo o processo e criar uma tabela com todas as transições identificadas isto é, o conjunto de tuplas $P = \{< p, s, a, s', r, h >\}$.

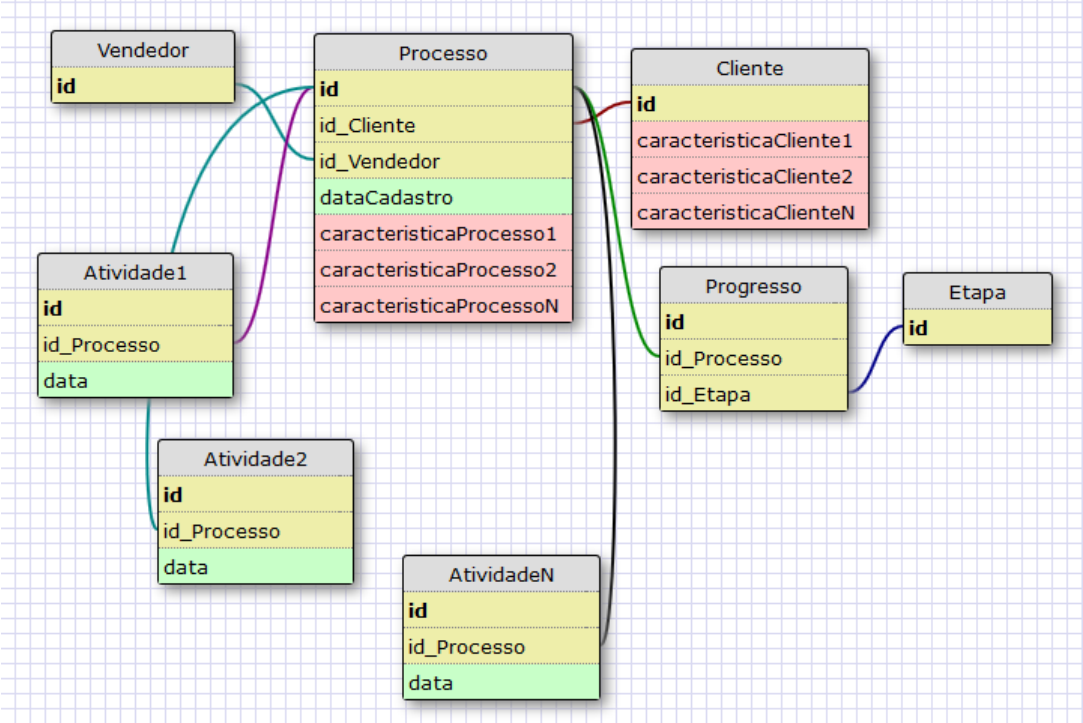


Figura 6.1: *Diagrama de banco de dados modelado como um workflow*

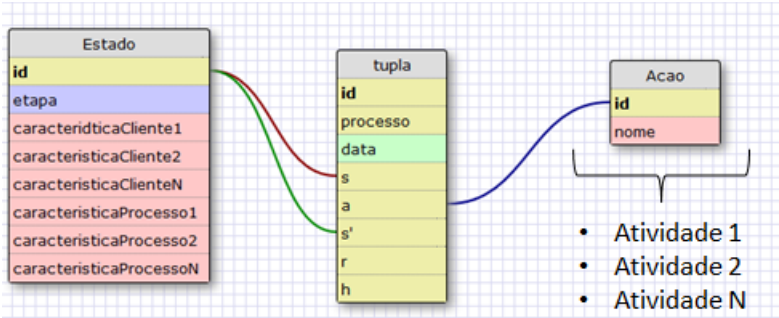
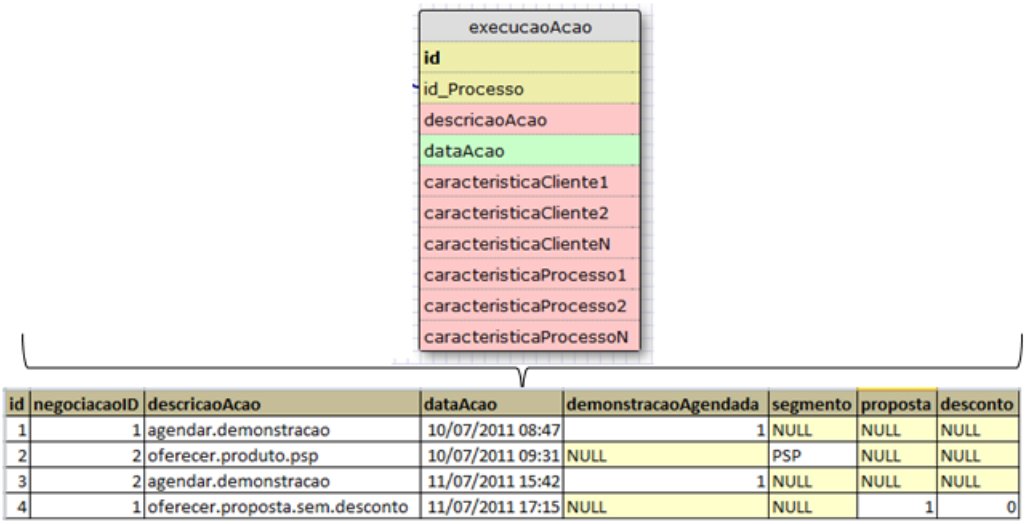


Figura 6.2: *Diagrama de banco de dados modelado na forma de transições (tuplas* $< p, s, a, s', r, h >$*)*



id	negociacaoID	descricaoAcao	dataAcao	demonstracaoAgendada	segmento	proposta	desconto
1	1	agendar.demonstracao	10/07/2011 08:47	1	NULL	NULL	NULL
2	2	oferecer.produto.psp	10/07/2011 09:31	NULL	PSP	NULL	NULL
3	2	agendar.demonstracao	11/07/2011 15:42	1	NULL	NULL	NULL
4	1	oferecer.proposta.sem.desconto	11/07/2011 17:15	NULL	NULL	1	0

Figura 6.3: *Tabela auxiliar de execução de ações usada para a construção do banco de dados da Figura 6.2*

6.2 Limpeza dos dados

Para o aprendizado de boas políticas é necessário garantir que os dados estejam completos, isto é, que não falte nenhuma informação relevante para a identificação completa do estado, e que não haja nenhuma inconsistência nos dados disponibilizados. Alteração manual pode ser uma das causas de registros incompletos ou de inconsistência no histórico de registros. Por exemplo, o sistema de venda pode obrigar o vendedor, para chegar num determinado estágio de negociação, a passar por diversas etapas e preencher inúmeras informações, no entanto, alterações manuais no banco de dados podem evitar que o vendedor tenha que respeitar o fluxo normal do sistema. Ainda que isso seja feito em caráter de exceção, estes registros podem atrapalhar o aprendizado da política ótima correta e, portanto, precisam ser retirados do conjunto de dados fornecidos para o aprendizado. Essa limpeza também foi realizada com a ajuda de um especialista.

6.3 Dinâmica do ambiente

Um dos desafios ao aplicar métodos de aprendizado em sistemas de venda é a dinâmica do mercado que pode ter sido alterada durante a fase de exploração (coleta das experiências). Muitos fatores provenientes do mercado influenciam na negociação de produtos, em especial produtos de alto valor. Por exemplo, se o mercado passa por um período de maior oferta de crédito, ou se a taxa de juros está baixa, é mais fácil ter um pedido de financiamento aprovado. Por outro lado, os pedidos de financiamento são negados com mais frequência quando diminui a oferta de crédito ou quando a taxa de juros sobe.

Conforme abordado no Capítulo 4, métodos de aprendizado por reforço em lote são bons pois permitem o aprendizado sem interação com o ambiente e conseguem acelerar o aprendizado reduzindo o número de experiências necessárias. No entanto, ao usar como amostra experiências de um determinado período, a política calculada refletirá a dinâmica de transições do período considerado e pode não ser útil caso a dinâmica do sistema atual seja diferente.

Para contornar esta dificuldade, propomos as seguintes soluções para resolver o problema do vendedor de impressoras:

1. **Descartar as negociações muito antigas.** Apesar do sistema *Forecast* contar com um histórico de mais de 7 anos de negociações, utilizamos para o cálculo da política ótima apenas os últimos 2 anos. Com isso evitamos usar para o aprendizado negociações muito antigas em que as diferenças do mercado em relação ao atual são relevantes. Usar os últimos dois anos como amostra não foi uma escolha arbitrária, e sim sugerida pelo gerente do projeto com base em seu conhecimento do mercado de impressoras *plotters*. O período considerado contém registros de 9.747 negociações somando 51.203 transições de estados distintas.

2. **Usar o lote de experiências em ordem cronológica**. Uma outra ideia adotada nesse trabalho, para tratar o problema de mudanças na dinâmica do modelo, foi usar os lotes de experiências em ordem cronológica, ou seja, na ordem em que as transições aconteceram no ambiente real. Na Equação 4.2 o fator de aprendizado (ou de esquecimento) α, assim como o fator de desconto γ podem fazer com que as experiências mais recentes tenham mais ou menos importância no aprendizado da função Q.

6.4 Resolvendo o problema: melhorias propostas no algoritmo de aprendizado por reforço em lote

Para resolver o problema foram utilizadas algumas técnicas de aprendizado por reforço em lote como aprendizado por reforço em lote puro (4.2.1) e aprendizado por reforço em lote crescente (4.3).

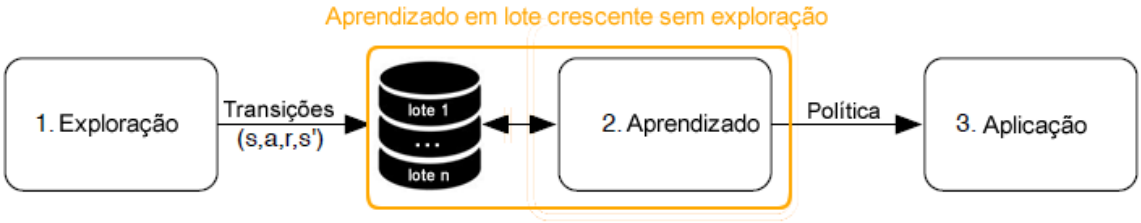


Figura 6.4: *Fases do aprendizado por reforço em lote crescente usado para resolver o problema do vendedor de impressoras*

6.4.1 Aprendizado por reforço em lote crescente sem exploração

Algoritmo 9: BRLCrescenteSemExploracao(Q_0, m)

Entrada: Q_0: uma política inicial,
m: o tamanho do lote

Saída: Q: a política ótima

1 $Q \leftarrow Q_0$;
2 $tupla \leftarrow 0$;
3 $D \leftarrow \{\}$
4 **enquanto** *Q não convergiu e tupla* $<$ *TAMANHO_BD* **faça**
5 $\quad i \leftarrow 0$
6 $\quad experiencias \leftarrow 0$
7 $\quad$ // incrementar o lote
8 $\quad$ **enquanto** $experiencias < m$ *e* $tupla <$ *TAMANHO_BD* **faça**
9 $\quad\quad i \leftarrow i + 1$
10 $\quad\quad experiencias \leftarrow experiencias + 1$
11 $\quad\quad tupla \leftarrow tupla + 1$
12 $\quad\quad d_i \leftarrow selecionaTupla(tupla)$
13 $\quad\quad D \leftarrow D \cup \{d_i\}$
14 $\quad$ **fim**
15 $\quad$ // Aprendizado
16 $\quad Q \leftarrow$ "*Executa um algoritmo de aprendizado em lote*"
17 **fim**
18 Devolve Q

Nem sempre é preciso usar todas as experiências do banco de dados, especialmente quando se trata de um número muito grande de dados. Em determinados problemas é possível que com uma fração do número de experiências do banco de dados, seja possível convergir para uma política ótima. Nesse caso propomos uma nova forma de calcular a política ótima, resolvendo o problema com lotes crescentes sem retornar para a fase de exploração, mas extraindo mais experiências do próprio banco de dados. A figura 6.4 ilustra este processo e o Algoritmo 9 mostra como seria uma implementação do aprendizado do aprendizado por reforço em lote crescente sem exploração. Como é possível notar, o algoritmo seleciona lotes de tamanho m no banco de dados (laço entre as linhas 8 e 14 do algoritmo) e atualiza a política usando todos os lotes coletados (linha 16) até que a política tenha convergido ou até que não tenha mais experiências disponíveis no banco de dados (laço entre as linhas 4 e 17).

6.4.2 Aprendizado por reforço em lote com regressão por processo

Algoritmo 10: BRLCrescenteComRegressao(Q_0, m)

Entrada: Q_0: uma política inicial,
m: o tamanho do lote

Saída: Q: a política ótima

```
1  Q ← Q_0;
2  tupla ← 0;
3  D ← {}
4  enquanto Q não convergiu e tupla < TAMANHO_BD faça
5      i ← 0
6      experiencias ← 0
7      // incrementar o lote
8      enquanto experiencias < m e tupla < TAMANHO_BD faça
9          i ← i + 1
10         experiencias ← experiencias + 1
11         tupla ← tupla + 1
12         d_i ← selecionaTerminal()
13         D ← D ∪ {d_i}
14         enquanto temPredecessor(d_i) faça
15             i ← i + 1
16             experiencias ← experiencias + 1
17             tupla ← tupla + 1
18             d_i ← selecionaPredecessor(d_{i-1})
19             D ← D ∪ {d_i}
20         fim
21     fim
22     // Aprendizado
23     Q ← "Executa um algoritmo de aprendizado em lote"
24 fim
25 Devolve Q
```

Conforme abordado no Capítulo 4, para resolver um problema geral de aprendizado por reforço em lote não são feitas suposições sobre o procedimento de amostragem das transições e as amostras não precisam ser coletadas ao longo de trajetórias ligadas (obtidas a partir de uma sequência de estados sucessores). No entanto, caso as amostras sejam coletadas ao longo de trajetórias ligadas, o algoritmo de aprendizado por reforço em lote pode ser melhorado utilizando as transições como *trials* do RTTP (transições ligadas) na ordem inversa: de um estado terminal (venda ou desistência) até encontrar um estado inicial (um dos doze estados iniciais do problema). A melhoria na convergência deve-se ao fato de que o valor da função Q para um determinado par estado-ação depende dos valores de Q dos possíveis estados sucessores. Ao executar o algoritmo de aprendizado em lote utilizando as transições ao longo de trajetória ligadas em ordem inversa, as atualizações dos valores de Q usarão uma estimativa mais atualizada de Q dos estados sucessores acelerando a convergência.

O Algoritmo 10 mostra como seria uma implementação de aprendizado por reforço em lote

com regressão por processo. A diferença em relação ao Algoritmo 9 é que neste algoritmo sempre que o lote precisa ser incrementado primeiro é selecionado a última transição de um processo (linha 12), mais precisamente a que leva para um estado terminal, e então todas as transições do mesmo processo são adicionadas ao lote na ordem inversa da ocorrência da transição (laço entre as linhas 14 e 20).

6.4.3 Teste de convergência

Considere:

- $\hat{V}(s)$ o valor aprendido para $V^*(s)$,
- $\hat{q}(s,a)$ o valor aprendido para $q^*(s,a)$,
- $T = \{t_1, t_2, ..., t_n\}$ um conjunto de teste com n tuplas onde $t_i = (s_i, a_i, s'_i, r_i)$, $s_i \in S$, $a_i \in A$, $s'_i \in S$, $r_i \in R$ indica em que a ação a_i foi executada no estado s_i, gerando uma recompensa r_i e indo para o estado s'_i.

Considere a equação:

$$E(i) = \hat{Q}(s_i, a_i) - (r_i + \gamma \hat{V}(s'_i)) \tag{6.1}$$

em que $E(i)$ é o erro calculado de $\hat{Q}(s_i, a_i)$ usando a experiência t_i. Considere $T_{s,a}$ um subconjunto de T em que:

$$T_{s,a} = \{t_{k1}, t_{k2}, ..., t_{kj}\} \text{ e } t_{ki} = (s, a, s'_{ki}, r_{ki}) \tag{6.2}$$

$$E(s,a) = \sum_k E(k), \text{em que } t_k \in T_{s,a} \tag{6.3}$$

A Equação 6.3 é uma estimativa do quanto $\hat{Q}(s,a)$ convergiu em relação ao conjunto de testes T. É possível notar que se $\hat{Q}(s,a) = Q^*(s,a)$, então para uma amostra de testes T não viesada e suficientemente grande, $E(s,a)$ converge para 0.

Para demonstrar, considere $\mathbb{E}_{s,a}(E(i))$ a esperança de $E(i)$ para $t_i \in T_{s,a}$ considerando $T_{s,a}$ uma amostra não viesada (com as mesmas frequências do ambiente) e suficientemente grande.

$$\begin{aligned} \mathbb{E}_{s,a}(E(i)) = \mathbb{E}(Q^*(s,a) - (R(s,a) + \gamma V^*(s'_i))) = \\ Q^*(s,a) - (R(s,a) + \gamma \mathbb{E}(V^*(s'_i))) = \\ Q^*(s,a) - (R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s')) = \\ Q^*(s,a) - Q^*(s,a) = 0 \end{aligned} \tag{6.4}$$

Sendo assim a equação 6.5 é uma boa estimativa do quanto a política aprendida convergiu para a política ótima. Chamaremos $E(T)$ de erro de convergência do lote T.

$$E(T) = \frac{\sum_{s \in S, a \in A} N(s,a) E(s,a)}{n}, \text{onde } N(s,a) = \text{número de tuplas } \{s, a, s'_i, r_i\} \tag{6.5}$$

, e n o número de experiências

Esta medida pode ser usada para comparar os de BRL. Com isso, uma forma de validar os algoritmos usados para resolver o *Problema do Vendedor de Impressoras* é adaptando a técnica *K-Fold* de *Validação Cruzada* do aprendizado supervisionado, apresentada no Apêndice A.4, usando a Equação 6.5 para validar o aprendizado em cada um dos K subconjuntos de teste.

Capítulo 7

Resultados Experimentais

No *Problema do Vendedor de Impressoras*, o objetivo é encontrar uma política ótima π^*, ou seja, uma função que mapeia a melhor ação de venda a ser executada em cada estado (Seção 5.3.1) de forma a maximizar a esperança das recompensas acumuladas ao longo de um processo de venda, considerando que o resultado da execução das ações é estocástico conforme modelado no Capítulo 5. Assim, queremos aprender a política ótima de venda de impressoras *plotters*, a partir do banco de dados de experiências de vendas do *Forecast*. Para isso, foram implementados 4 algoritmos de aprendizado por reforço em lote, a saber:

1. BRL sem repetição (Seção 4.2)
2. BRL com repetição (Seção 4.2.1)
3. BRL crescente sem exploração (Seção 6.4.1)
4. BRL com regressão (Seção 6.4.2)

Na Seção 6.4.3 discutimos uma maneira de calcular o erro de convergência da função valor computado pelo aprendizado por reforço em lote que será usada nesse capítulo e que também será adaptada para realizar uma *Validação Cruzada*, conhecida como *K-Fold* na área de aprendizado supervisionado e detalhada no Apêndice A.4.

As seções a seguir descrevem diferentes análises realizadas neste trabalho dos algoritmos de aprendizado por reforço em lote implementados para resolver o problema do vendedor de impressoras.

7.1 BRL crescente (sem exploração) sem repetição

A Figura 7.1 mostra o erro de convergência do lote (usando a definição de convergência de lotes descrita na Seção 6.4.3), do algoritmo de aprendizado por reforço em lote crescente, conforme foi proposto na Seção 6.4.1, isto é, que considera lotes cada vez maiores do banco de experiências do *Forecast*. Foi utilizado o algoritmo básico de BRL, sem repetição, para cada lote de experiências considerado, com $\alpha = 0.2$ e $\gamma = 0.9$. Como era esperado, quanto maior o tamanho da amostra t, menor é o erro $E(t)$ do lote.

7.2 BRL sem repetição vs BRL com Repetição

A Figura 7.2 compara o aprendizado por reforço em lote sem repetição e com repetição de experiências. Note que além do erro de convergência diminuir com o aumento do tamanho do lote, como mostrado no experiento anterior, o erro também diminui com o aumento do número de repetições. Os resultados mostram que quanto maior o lote, repetir experiências provoca menor impacto na diminuição do erro de convergência. Por exemplo, usando 25% da base de

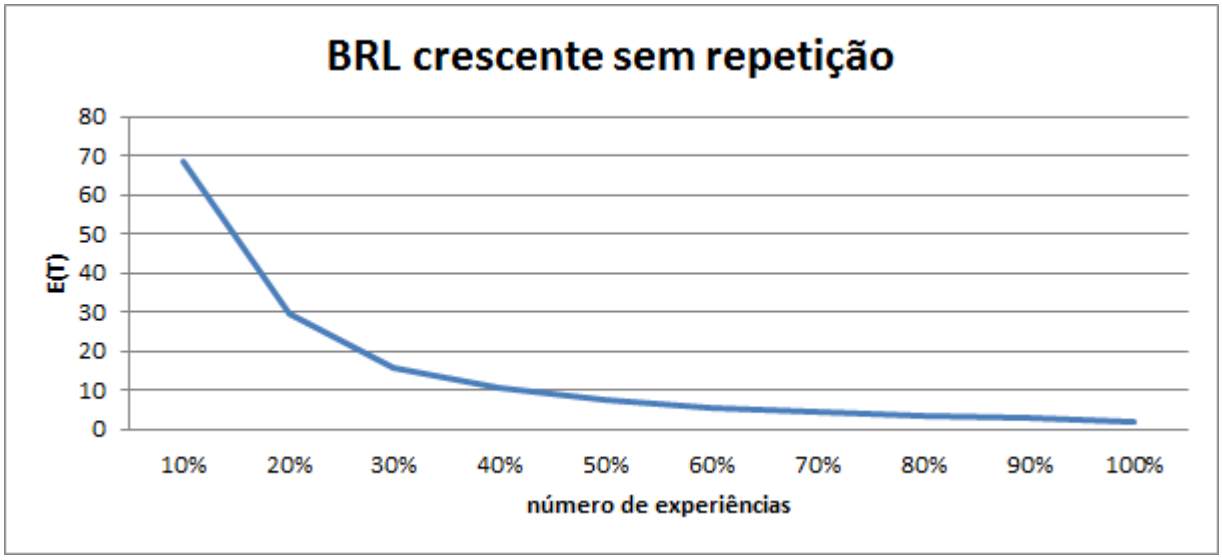


Figura 7.1: *Aprendizado por reforço em lote crescente sem repetição*

experiências, é possível diminuir o erro de convergência do lote de 65.5 (sem repetições) para 41.4 (com 10 repetições). Por outro lado, usando 100% da base de experiências, o erro diminui de 9 (sem repetições) para 5.15 (com 10 repetições).

Note ainda que a versão do algoritmo com repetição de experiências precisou de apenas 50% da amostra com 10 repetições para atingir um erro de convergência do lote aproximadamente igual a 12.5, enquanto que o uso de 100% da amostra sem repetição resulta num erro de convergência igual a 8.9. Isso mostra que o aprendizado por reforço em lote pode de fato alcançar bons resultados com um número menor de amostras, diminuindo assim o custo das interações com o ambiente. Foram usados para esta experiência os valores $\gamma = 0,9$ e $\alpha = 0.2$.

7.3 Ordenação do lote de experiências

A princípio, a política aprendida pelo aprendizado por reforço em lote independe da ordem em que as experiências são consideradas, porém, adotar uma ordem adequada para atualizar os estados pode interferir no tempo de convergência. Usar as transições de um dado processo de negociação na ordem inversa do tempo, isto é, a partir do final do processo de venda regredimos até o estado inicial, pode acelerar a convergência como visto na Seção 6.4.2. A Figura mostra que usando 50% das experiências e variando o número de repetições, o BRL com regressão converge mais rapidamente que o BRL sem regressão com 1, 2 e 3 repetições. A regressão parece não ser suficiente no caso em que o número de repetições é zero, uma explicação para isso é que no *Forecast* 90% das negociações terminam em fracasso de venda, e usando um fator de aprendizado $\alpha = 0.2$ o algoritmo demora para esquecer o reforço inicial negativo obtidos pela propagação dos processos de fracasso e passar a considerar os reforços positivos das negociações de sucesso.

7.4 K-Fold: Verificando a qualidade da amostra

Como foi dito anteriormente, o erro de convergência da função valor computada pelo BRL também pode ser usado para fazer uma avaliação do erro usando como lote de teste 10% da amostra e aprendendo a política com 90% de experiências da amostra, como é feito no aprendizado supervisionado. A Figura 7.4 mostra o desempenho do algoritmo BRL com regressão usando diferentes lotes de teste contendo 10% das experiências considerando uma partição do conjunto completo de experiências dividido em 10 lotes. Os resultados mostram que o erro de convergência do lote de teste considerado pode variar entre 13.5 e 17.7, o que pode ser considerado baixo.

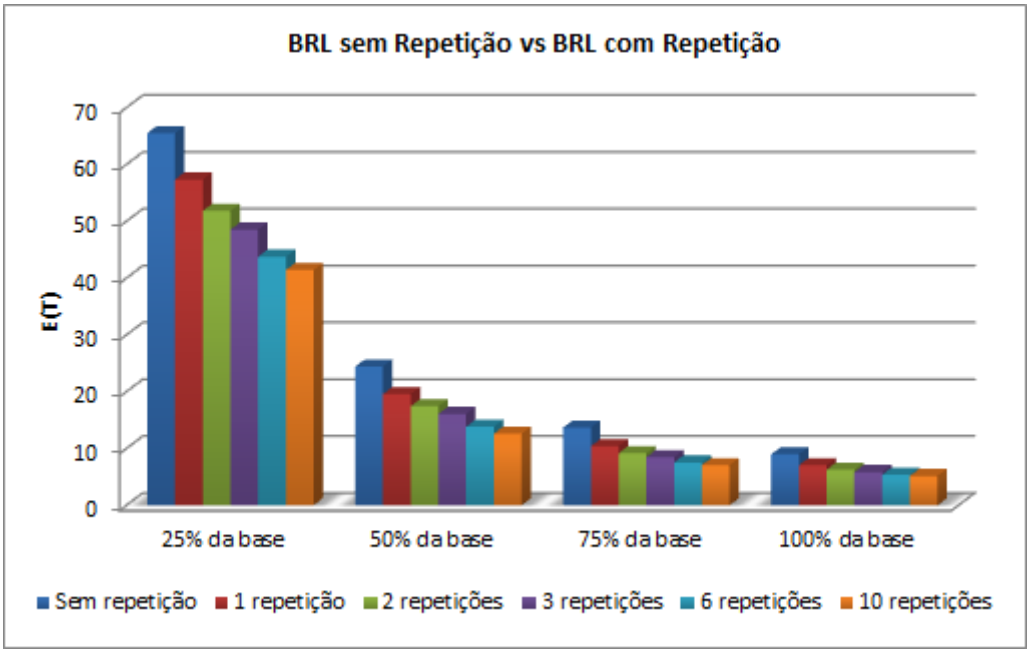


Figura 7.2: *Aprendizado por reforço em lote para o problema do vendedor de impressoras variando a quantidade de experiências da base (25%, 50%, 75% e 100%) e o número de repetições (de 0, 1, 2, 3, 6 e 10).*

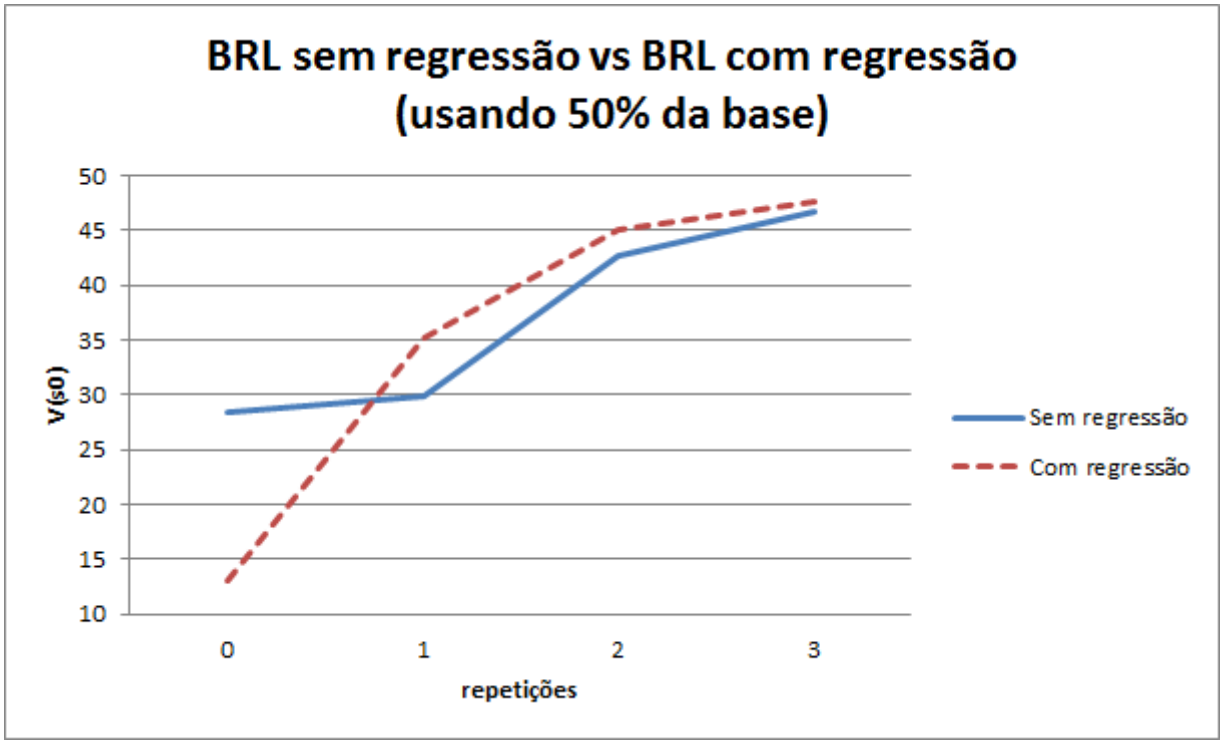


Figura 7.3: *Aprendizado por reforço em lote com regressão vs aprendizado por reforço em sem regressão*

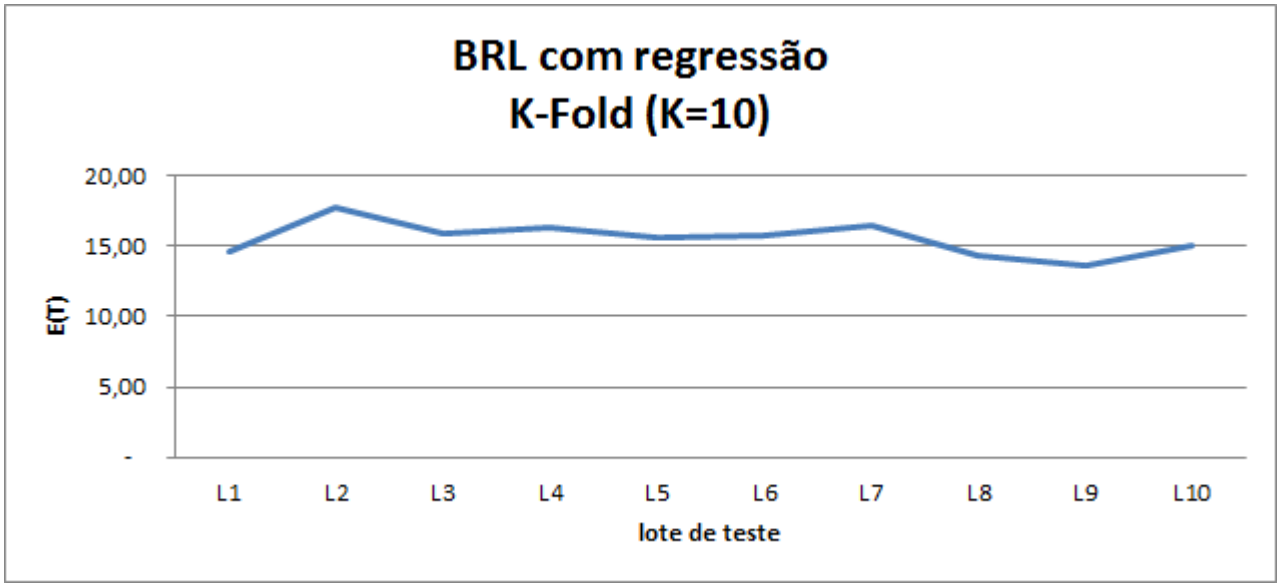


Figura 7.4: *K-Fold para verificar a qualidade da amostra (sem repetição).*

7.5 Política dos vendedores versus política aprendida

A próxima análise compara as ações tomadas pelos vendedores no lote de 100% das experiências consideradas (contendo 51203 transições), com a política ótima gerada pelo algoritmo BRL sem repetição (também com 100% das experiências) obtendo os seguintes resultados:

- Em 52.72% das transições, a ação escolhida pelos *vendedores* é a mesma que a ação indicada pela política aprendida;
- Em 61,83% das transições, a ação escolhida pelos *melhores vendedores* (os 11 vendedores responsáveis por 80% das negociações de sucesso de venda) é a mesma que a ação indicada pela política aprendida;
- Em 71,43% das transições das negociações que terminaram em sucesso, a ação escolhida pelos *vendedores* seguiram a política aprendida;
- Em 71,99% das transições das negociações que terminaram em sucesso, a ação escolhida pelos *melhores vendedores* seguiram a política aprendida;

Com isso é possível perceber que a política adotada pelos vendedores nos casos de sucesso é ainda mais próxima da política aprendida e a política adotada pelos melhores vendedores é mais próxima da política aprendida.

Capítulo 8

Conclusão

Este trabalho mostrou como modelar processos de venda como um processo de decisão markoviano e como aprender políticas a partir de um banco de dados de negociação de venda usando um algoritmo de aprendizado por reforço em lote, e suas variações.

Usamos o banco de dados do sistema *Forecast* selecioando processos de venda dos últimos 2 anos que correspondem a cerca de 50000 transições e 10000 processos de venda. Os resultados empíricos mostraram que é possível aprender boas políticas usando apenas 50% das experiências com 10 repetições.

Propomos uma medida de erro de convergêcia que permite comparar o desempenho de algoritmos em que são utilizadas amostras para o aprendizado de políticas, como o aprendizado por reforço em lote.

Identificamos que é possível melhorar o Aprendizado por Reforço em Lote nos casos em que as transições vieram de trajetórias completas apenas aprendendo com cada processo na ordem contrária em que as transições aconteceram. Isso ocorre porque o valor de um estado depende dos valores dos possíveis estados futuros.

Por fim, quando comparamos a política aprendida com as ações executadas nos processos de venda, foi possível identificar que em cerca de 72% dos processos de sucesso, os melhores vendedores escolheram as mesmas ações indicadas pela política aprendida.

A política ótima aprendida a partir do banco de dados do *Forecast* poderá ser usada de três diferentes formas pelo programa Big Impression:

- recomendação de ações para os vendedores do grupo,
- identificação de vendedores que necessitam de treinamento e
- identificação dos estados mais cíticos e detecção das causas de vendas sem sucesso.

Trabalhos futuros

Utilizando aprendizado por reforço em lote com regressão e repetição de experiências e com base nas medidas de desempenho propostas nesse trabalho, foi possível calcular políticas com um erro de convergência muito baixo para o banco de dados de 2 anos de processos de venda do *Forecast*. A política ótima aprendida pode ser usada para identificar os vendedores menos experientes, que precisam ainda passar por um periodo de treinamento. Com isso é possível criar um sistema de apoio à tomada de decisão que recomenda ações de venda para os vendedores menos experientes.

Com a hipótese de que o ambiente (mercado) varia, mesmo que lentamente, é excencial usar as técnicas de *aprendizado em lote estendido* para que a política seja sempre atualizada na medida em que o mercado sofra variações. Porém, diferente da técnica descrita na Seção 4.3, no problema do vendedor de impressoras, não é possível interagir no ambiente com o objetivo apenas de coletar novas experiências. Neste caso, pode-se usar as mesmas transições reais coletadas na fase de aplicação para incrementar o lote de experiências. A Figura 8.1 mostra as fases de um sistema de aprendizado que pode ser usado para resolver o problema descrito neste trabalho mantendo uma taxa de aprendizado constante na medida em que o mercado varie.

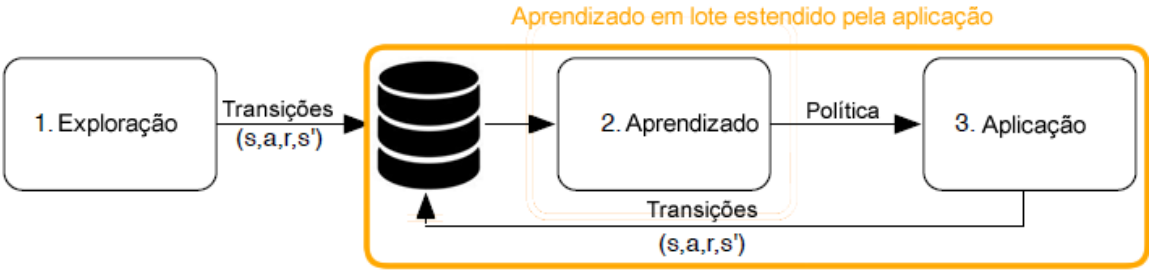


Figura 8.1: *Fases do aprendizado por reforço em lote crescente usado para resolver o problema do vendedor de impressoras.*

Apêndice A

Aprendizado Supervisionado

Aprendizado de máquina é uma subárea da Inteligência Artificial que dedica-se ao desenvolvimento de técnicas computacionais que permitam que uma máquina (agente) aprenda um novo comportamento com base em dados empíricos provenientes de sensores ou de banco de dados.

A.1 Introdução

Pesquisas na área de *aprendizado de máquina* lidam com a construção de agentes que possam *aprender* com suas experiências (Mitchell, 1997), ou seja, agentes que melhoram seu desempenho em uma deteminada tarefa através de uma dada *medida de desempenho*. Formalmente, um agente aprende a partir da experiência E, em relação a uma classe de tarefas T, com medida de desempenho P, se seu desempenho P em T melhora com E (Mitchell, 1997).

Usando como exemplo prático a tarefa de fazer *análise de crédito* teríamos:

- **Tarefa T:** classificar novos clientes como bons ou maus pagadores,
- **Medida de desempenho P:** porcentagem de clientes classificados corretamente,
- **Experiência de treinamento E:** uma base de dados em que clientes conhecidos são classificados como bons ou maus pagadores.

Podemos dividir as técnicas de apendizado em 3 grupos mostrados na Figura A.1. Dizemos que o aprendizado é *supervisionado* quando é dado um conjunto de exemplos do comportamento desejado e *não supervisionado* quando é dado um conjunto de comportamentos quaisquer. Neste último caso, o aprendizado dá-se através de uma medida que pode ser usada para identificar

Figura A.1: *Tipos de aprendizado.*

grupos de comportamentos, entre eles, o comportamento desejado. O *aprendizado por reforço* será detalhado no Capítulo 3.

A.2 Aprendizado supervisionado

No aprendizado supervisionado é fornecida uma referência do objetivo a ser alcançado, ou seja, o algoritmo de aprendizado recebe o valor de saída desejado para cada conjunto de dados de entrada apresentado. Neste caso, o processo de aprendizado significa aprender uma função a partir de exemplos de sua entrada e saída, e usar a função aprendida para outras entradas não previstas no treinamento.

Etapas do aprendizado supervisionado:

1. Coletar um grande conjunto de exemplos,
2. Dividir este conjunto em dois subconjuntos distintos: conjunto de treinamento e conjunto de teste,
3. Treinar o algoritmo de aprendizado utilizando o conjunto de treinamento,
4. Simular o algoritmo de aprendizado treinado no conjunto de testes e medir a porcentagem de exemplos corretamente classificados,
5. Repetir os passos de 1 a 5 para diferentes tamanhos de conjuntos de treinamento e diferentes conjuntos de treinamento

A.3 Aprendizado não-supervisionado

No aprendizado não-supervisionado é fornecido apenas o conjunto de dados de entrada sem a saída desejada. Em geral é utilizado como classificador e funciona encontrando aglomerados de conjuntos de dados semelhantes entre si (*clusters*). O processo não supervisionado é obviamente mais difícil que o supervisionado pois o sistema precisa definir sozinho o número de classes. Técnicas de aprendizado não supervisionado são muito utilizadas em *mineração de dados* e funcionam melhor quando as classes estão bem separadas no espaço de atributos.

As etapas do processo de aprendizagem não supervisionada envolvem:

1. **Seleção de atributos**: os atributos devem ser selecionados de forma que codifiquem a maior quantidade possível de informações evitando redundância entre eles,
2. **Medida de proximidade**: uma medida para quantificar a similaridade entre dois vetores de atributos,
3. **Critério de agrupamento**: pode ser compreendido como a sensibilidade que o algoritmo usado deve adotar,
4. **Algoritmo de clusterização**: após adotar uma medida de proximidade e um critério de agrupamento, o algoritmo de crusterização agrupa os dados em classes de objetos similares
5. **Validação dos resultados**: testes apropriados são usados para verificar o resultado.
6. **Interpretação dos resultados**

A.4 Validação dos sistemas de aprendizado

O principal objetivo de um sistema de Aprendizado é predizer com sucesso casos que ainda não foram vistos. Isto se torna bastante problemático quando um conjunto pequeno de dados de exemplo está disponível para treinamento. Nestes casos não se sabe dizer se tal número reduzido de dados para treinamento é suficiente para extrair o conhecimento necessário.

Existem inúmeros métodos utilizados para estimar o desempenho de um sistema de aprendizado. Alguns desses métodos podem ser tendenciosos ao apresentar estimativas muito otimistas ou muito pessimistas. Encontrar índices capazes de estimar de forma não tendenciosa o desempenho de um sistema de aprendizado é a base uma avaliação adequada dos sistemas de aprendizado.

A.5 Taxa de erro

Quando fornecemos uma entrada para um algoritmo de aprendizado de máquina este responde com modelo aprendido. O modelo aprendido pode ser o correto mas também pode conter erros. A taxa de erro é uma maneira bastante utilizada para medir a performance de algoritmos de aprendizado. Pode-se calcular a taxa de erro do algoritmo através da equação A.1.

$$E = \frac{\text{número de erros}}{\text{número de casos}} \tag{A.1}$$

Consideramos duas definições para a taxa de erro:

- **Taxa de erro amostral**: é a taxa de erro obtida utilizando somente os exemplos de treinamento, ou seja, o sistema aprende com um conjunto de exemplos e posteriormente é verificado quanto erro cometeu utilizando os mesmos exemplos com que foi treinado,
- **Taxa de erro real**: é a taxa de erro obtida sobre um conjunto muito grande de novos casos independentes dos casos usados para treinamento.

A taxa de erro real obviamente fornece uma excelente medida de acurácia de um algoritmo de aprendizado, no entanto, em casos reais conjuntos grandes de exemplo são raros e é muito comum a taxa de erro amostral ser muito inferior a taxa de erro real. Sendo assim, se a taxa de erro amostral fosse utilizada a estimativa de acurácia seria otimista e pouco confiável. Uma alternativa é tentar se aproximar da taxa de erro real através de taxas de erros empíricas calculadas com pequenos conjuntos de exemplo.

Existem métodos estatísticos em que os algoritmos de aprendizado são executados com diferentes instâncias dos dados de amostra permitindo obter aproximações da taxa de erro real bem mais confiáveis que a taxa de erro amostral. Estes métodos serão apresentados na sessão A.6.

A.6 Estimativa da taxa de erro real

Como o cálculo da taxa de erro real é, na maioria das vezes, inviável e a taxa de erro amostral não é uma boa estimativa, uma alternativa é encontrar estimativas mais precisas para a taxa de erro real.

Um dos requisitos para estimar a taxa de erro real é manter a amostra em ordem aleatória, ou seja, a amostra não deve ser pré-selecionada evitando que qualquer cálculo seja feita sobre exemplo mais ou menos representativos.

A.6.1 Treinar e testar

O princípio dos métodos *Treinar e Testar* está em dividir a amostra em dois grupos mutuamente exclusivos e independentes. O primeiro, chamado de *conjunto de treinamento* é usado para treinar o sistema de aprendizado. O segundo, chamado de *conjunto de testes*, é usado apenas para medir a taxa de erro do sistema de aprendizado.

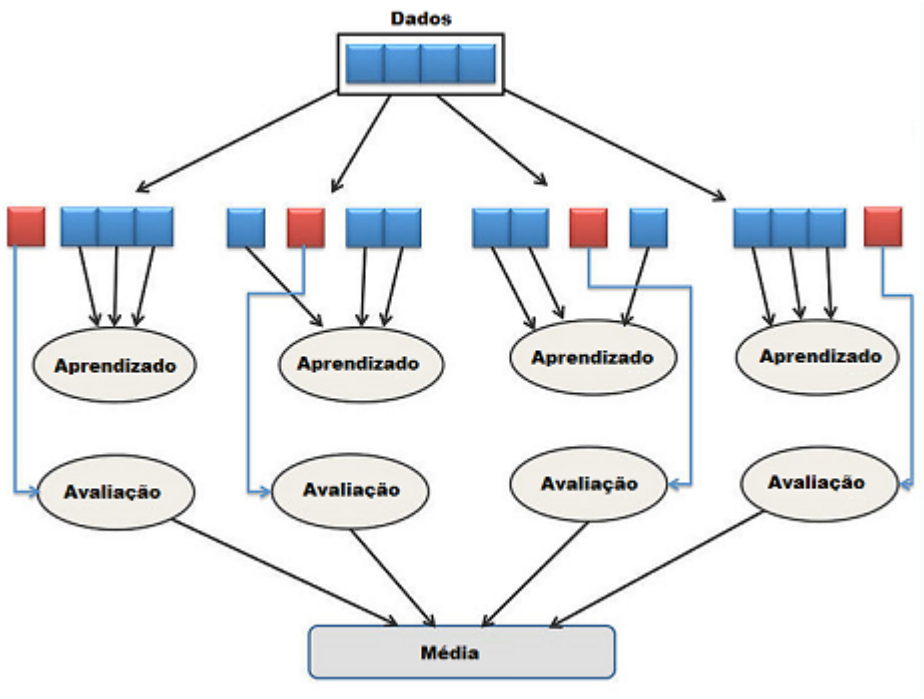


Figura A.2: *Cross Validation. Método K-fold*

A.6.2 Reamostragem

Métodos de reamostragem são variações dos métodos *Treinar e Testar* nos quais são realizados inúmeros experimentos de Treinar e Testar com diferentes conjuntos de treinamento e teste. Em cada experimento é obtida uma taxa de erro, e a média dessas taxas de erro é a estimativa para a taxa de erro real. Para isso procura-se encontrar um estimador que não seja tendencioso, ou seja, que não estime a taxa de erro de forma otimista nem pessimista. Um estimador não tendencioso precisa convergir para o valor correto quando aplicado para uma amostra suficientemente grande. O estimador pode variar de amostra para amostra, mas a média sobre todas as amostras é correta.

A.6.3 Validação cruzada (Cross Validation)

Um dos métodos de estimativa de erro por reamostragem é o Validação Cruzada (Cross Validation). Também é conhecido como *K-fold Cross Validation* onde o K representa o número de partições da amostra geradas aleatoriamente.

Neste método a amostra é dividida em K partições mutuamente exclusivas e independentes. A cada iteração uma iteração diferente é usada para testar o sistema enquanto todas as K-1 partições são usadas para treinar o sistema. A taxa de erro é a média das taxas de erros calculadas nas K iterações.

Referências Bibliográficas

Baird (1995) Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. Em *anais do Twelfth International Conference on Machine Learning*, páginas 30–37. Morgan Kaufmann. Citado na pág. 21

Barto *et al.* (1993) Andrew G. Barto, Steven J. Bradtke e Satinder P. Singh. Learning to act using real-time dynamic programming, 1993. Citado na pág. 11

Bellman (1957) R. E. Bellman. *Dynamic Programming*. Princeton University Press, USA. Citado na pág. 1, 7, 8

Bonet e Geffner (2003) Blai Bonet e Héctor Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. Em *ICAPS*, páginas 12–21. AAAI Press. Citado na pág. 11

Ernst *et al.* (2005) Damien Ernst, Pierre Geurts, Louis Wehenkel e L. Littman. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556. Citado na pág. 2, 19, 22, 23

Ghallab *et al.* (2004) M. Ghallab, D. Nau e P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers Inc., USA. Citado na pág. 1

Gordon (1995) Geoffrey J. Gordon. Stable function approximation in dynamic programming. Relatório técnico, Carnegie Mellon University, Pittsburgh, PA, USA. Citado na pág. 21

Hastings (1970) W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109. doi: 10.1093/biomet/57.1.97. URL http://biomet.oxfordjournals.org/cgi/content/abstract/57/1/97. Citado na pág. 12

Impression (2013) Programa Big Impression. Shopping do plotter, 2013. URL http://www.shoppingdoplotter.com.br. Citado na pág. 3

ji Lin (1992) Long ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, Vol. 8:293–321. Citado na pág. 21, 22, 23

Kaelbling *et al.* (1996) Leslie Pack Kaelbling, Michael L. Littman e Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103. URL http://dblp.uni-trier.de/db/journals/corr/corr9605.html. Citado na pág. 2, 13

Kalyanakrishnan e Stone (2007) Shivaram Kalyanakrishnan e Peter Stone. Batch reinforcement learning in a complex domain. Em *The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, páginas 650–657, New York, NY, USA. ACM. ISBN 978-81-904262-7-5. Citado na pág. xv, 22, 23, 24, 25

Keller e Eyerich (2012) Thomas Keller e Patrick Eyerich. PROST: Probabilistic planning based on UCT. Em *ICAPS*, páginas 119–127. AAAI Press. Citado na pág. 12

Kocsis e Szepesvári (2006) Levente Kocsis e Csaba Szepesvári. Bandit based monte-carlo planning. Em *ECML-06*. Citado na pág. 12

Kohavi (1995) Ron Kohavi. A study of cross-validation and Bootstrap for accuracy estimation and model selection. Em *International Joint Conference on Artificial Intelligence (IJCAI)*, páginas 1137–1143. Morgan Kaufmann. Citado na pág. 4

Mitchell (1997) T.M. Mitchell. *Machine Learning*. McGraw-Hill Series in Computer Science. McGraw-Hill. ISBN 9780070428072. URL http://books.google.com.br/books?id=nlmVQgAACAAJ. Citado na pág. 16, 51

Puterman (1994) M. L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc. Citado na pág. 9

Rummery e Niranjan (1994) G. A. Rummery e M. Niranjan. On-line Q-learning using connectionist systems. Relatório Técnico TR 166, Cambridge University Engineering Department, Cambridge, England. Citado na pág. 16

Russell e Norvig (2003) Stuart J. Russell e Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education. ISBN 0137903952. Citado na pág. 7

Sascha Lange e Riedmiller (2012) Thomas Gabel Sascha Lange e Martin Riedmiller. Batch reinforcement learning. Em M. Wiering e M. van Otterlo, editors, *Reinforcement Learning: State-Of-The-Art*, volume 12 of *Adaptation, Learning, and Optimization*, páginas 45–73. Springer, Heidelberg, Germany. URL http://www.springer.com/978-3-642-27644-6. Citado na pág. xi, 2, 19, 20, 21, 25, 26

Sutton (1990) Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. Em *Seventh International Conference on Machine Learning*, páginas 216–224. Morgan Kaufmann. Citado na pág. 17

Sutton (1996) Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. Em *Advances in Neural Information Processing Systems 8*, páginas 1038–1044. MIT Press. Citado na pág. 17

Sutton e Barto (1998) R.S. Sutton e A.G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. Mit Press. ISBN 9780262193986. URL http://books.google.com.br/books?id=CAFR6IBF4xYC. Citado na pág. xi, 2, 13, 19

van Der Aalst *et al.* (2003) Wil MP van Der Aalst, Arthur HM Ter Hofstede, Bartek Kiepuszewski e Alistair P Barros. Workflow patterns. *Distributed and parallel databases*, 14(1):5–51. Citado na pág. 37

Watkins e Dayan (1992) Christopher J. C. H. Watkins e Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292. URL http://jmvidal.cse.sc.edu/library/watkins92a.pdf. Citado na pág. 2, 15

Índice Remissivo

Aprendizado de Máquina, 51
- Aprendizado não-supervisionado, 52
- Aprendizado supervisionado, 52
- Definição, 51

Aprendizado por reforço, 13
- Definição, 13
- Dyna-Q, 17
- Limitações, 20
- Métodos, 14
- Q-Learning, 15
  - $\epsilon - Greedy$, 16
  - Algoritmo, 16
  - Condições de convergência, 16
  - Fator de aprendizado, 15
  - Passo de aproximação da função Q, 15
  - Passo de programação dinâmica, 16
- SARSA, 16

Aprendizado por reforço em lote, 19
- Algoritmos, 22
  - BRL ajustado, 23
  - Repetição de experiência, 23
- Aprendizado em lote crescente, 24
- Classificação dos algoritmos, 26
- Definição, 19
- Fundamentos, 20
- Problema geral, 20

Impressoras Plotters, 28
- Segmento criativo, 29
- Segmento PSP, 29
- Segmento técnico, 29

O Problema do Vendedor de Impressoras, 27
- Estágios do sistema, 29
  - Aceite da proposta comercial, 30
  - Conhecer melhor o Cliente, 30
  - Entrega, 30
  - Negócio perdido, 31
  - Pedido, 30
  - Pré Qualificação, 30
  - Proposta Comercial, 30
  - Secesso, 30
  - Standby, 31
- Sistema de tomada de decisão, 31
- Variáveis de estado, 31

O Processo de Venda, 27
- Ciclo de venda, 28
- Definição das necessidades do cliente, 27
- Elaboração de propostas, 27
- Fechamento do negócio, 28
- Negociação, 28
- Pós-venda, 28
- Prospecção, 27
- Qualificação do cliente, 27

Planejamento, 1
- Ação ótima, 10
- Ação gulosa, 9
- Definição, 1
- MDP, 7
  - Equação de Bellman, 8
  - Horizonte, 7
  - Iteração de Valor, 9
  - Política, 8
  - Política Ótima, 8
  - Soluções Online, 11
- Planejamento Probabilístico, 7
- Planejamento Probabilistico, 1
- sistema de transição de estado, 1

Validação dos sistemas de aprendizado, 53
- Erro, 53
  - Erro amostral, 53
  - Erro real, 53
- Estimativa do erro real, 53
- Reamostragem, 54
- Treinar e testar, 53
- Validação cruzada, 54