

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ  
PRÓ-REITORIA DE PESQUISA, PÓS-GRADUAÇÃO E INOVAÇÃO  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Victor Ribeiro Prata

## **Sistema de Recomendação de Próxima Funcionalidade em Software**

Fortaleza – CE

20 de abril de 2021



Victor Ribeiro Prata

## **Sistema de Recomendação de Próxima Funcionalidade em Software**

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Linha de Pesquisa: Inteligência Artificial

Orientador: Prof. Dr. Ajalmar Rêgo da Rocha Neto

Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação (PRPI)

Programa de Pós-Graduação em Ciência da Computação (PPGCC)

Fortaleza – CE

20 de abril de 2021

Dados Internacionais de Catalogação na Publicação  
Instituto Federal do Ceará - IFCE  
Sistema de Bibliotecas - SIBI  
Ficha catalográfica elaborada pelo SIBI/IFCE, com os dados fornecidos pelo(a) autor(a)

---

- P912s Prata, Victor Ribeiro.  
Sistema de Recomendação de Próxima Funcionalidade em Software / Victor Ribeiro Prata. - 2021.  
110 f. : il. color.
- Dissertação (Mestrado) - Instituto Federal do Ceará, Mestrado em Ciência da Computação, Campus Fortaleza, 2021.  
Orientação: Prof. Dr. Ajalmar Rêgo da Rocha Neto.
1. Sistema de recomendação. 2. Redes Neurais Recorrentes. 3. Cadeias de Markov. 4. Aprendizado de Máquina. I. Título.

CDD 004

---



Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)  
Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação (PRPI)  
Programa de Pós-Graduação em Ciência da Computação (PPGCC)

Victor Ribeiro Prata

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, sendo aprovada pela Coordenação do Programa de Pós-Graduação em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará e pela banca examinadora:

---

**Orientador: Prof. Dr. Ajalmar Rêgo da Rocha Neto**

Instituto Federal de Educação, Ciência e Tecnologia do  
Ceará (IFCE)

---

**Prof. Dr. Jose Maria Pires De Menezes Junior**  
Universidade Federal do Piauí (UFPI)

---

**Prof. Dr. Amauri Holanda de Souza Júnior**  
Instituto Federal de Educação, Ciência e Tecnologia do  
Ceará (IFCE)

Fortaleza – CE  
20 de abril de 2021



*Este trabalho é dedicado aos meus pais e esposa, que me ajudaram durante toda essa caminhada. Obrigado!*



# Agradecimentos

Em primeiro lugar agradeço a Deus, pois até aqui me ajudou o Senhor.

Agradeço de todo meu coração ao meu pai e minha mãe, Mirton de Paiva Prata e Damiana Ribeiro Prata, que desde cedo me incentivaram a buscar na educação um futuro honesto, não medindo esforços para me ajudar nos momentos mais difíceis. Agradeço à minha esposa, Raissa Leandro de Sousa Prata, por ser minha conselheira e meu suporte.

Ao professor Ajalmar Rêgo da Rocha Neto por me orientar neste trabalho e ter me proporcionado a oportunidade de fazer parte de outros trabalhos.

Aos professores Amauri Holanda Souza Júnior e Henrique Leitão por suas contribuições neste trabalho e pelo tempo disponibilizado para tirar muitas das minhas dúvidas.

Aos colegas que de alguma forma contribuíram com a pesquisa realizada neste trabalho: Luan Sousa Cordeiro, Alan Rabelo, Átila Negreiros e Davi Leão.

Ao Ronaldo Moreira, CTO da Fortes Tecnologia, por suas contribuições, parceria e disponibilização da base de dados utilizada neste trabalho.

Ao Polo de Inovação IFCE/Embrapii, por possibilitar que este projeto fosse realizado.

E por fim, a todos que ajudaram de alguma forma neste trabalho.



# Resumo

Neste trabalho foi proposto e avaliado a implementação de um sistema de recomendação de próxima funcionalidade em softwares. O objetivo principal desse sistema de recomendação é de gerar recomendações inteligentes de funcionalidades de softwares para seus usuários com base no histórico de interação desses usuários com as funcionalidades. Cinco algoritmos de aprendizado de máquina foram analisados para esse sistema de recomendação e seus resultados comparados. Para tanto, os experimentos foram realizados em uma base de dados inédita que representa o uso real de seis softwares de contabilidade por milhares de usuários. Para escolha dos algoritmos, buscou-se na literatura os algoritmos mais utilizados em sistemas de recomendação semelhantes, dentre os quais foram selecionados os algoritmos de Cadeias de Markov de primeira e segunda ordem, redes neurais recorrentes LSTM, IndRNN e a Rede Neural Recorrente padrão (RNR). Foram ainda analisados três cenários de treinamento diferentes para os modelos, nos quais os históricos dos usuários foram compartilhados em três níveis diferentes, a saber: *(i)* treinamento a nível de usuário, em que para cada usuário foi treinado um modelo apenas com o histórico pessoal do usuário; *(ii)* treinamento a nível de empresa, no qual, para cada empresa, é treinado um modelo com o histórico de todos os usuários dessa empresa, sendo esse modelo compartilhado entre esses usuários; *(iii)* treinamento a nível de sistema, em que é gerado apenas um único modelo para todos os usuários do software com base no histórico de todos os usuários. Os resultados mostraram que os algoritmos de redes neurais conseguiram obter uma ligeira vantagem na capacidade de predição de próxima funcionalidade nos três cenários de teste, porém, as cadeias de Markov apresentaram um tempo de treinamento extremamente mais rápido. A partir dos resultados foi possível concluir que o desempenho de todos os modelos conseguiu ser satisfatório quando, ao invés de recomendar apenas a próxima funcionalidade, é recomendada uma lista de próximas funcionalidades mais prováveis, com os melhores resultados apresentando uma acurácia de 73% para as redes neurais recorrentes IndRNN e RNR padrão. As contribuições deste trabalho vão desde o fato de um sistema de recomendação de próxima funcionalidade de software ser um assunto pouco explorado, com os trabalhos relacionados se limitando a recomendação de páginas de sites, até o uso de uma base de dado inédita, que representa interações reais entre usuários de 6 softwares e as funcionalidades desses softwares. Além disso, a proposta de um sistema de recomendação de próxima funcionalidade é útil comercialmente, dada a capacidade de auxiliar a produtividade das empresas, tendo em vista a economia de tempo que se tem ao recomendar a próxima funcionalidade bem como auxiliar usuários novatos em determinados fluxos de trabalhos.

**Palavras-chave:** Sistema de recomendação. Redes Neurais Recorrentes. Cadeias de Markov. Aprendizado de Máquina.



# Abstract

In this work, it was proposed and evaluated an implementation of a recommendation system for the recommendation of next functionality in software. The main objective of this recommendation system is to generate intelligent recommendations of features of certain software for its users based on the history of access of these users to the features. Five machine learning algorithms were analyzed for this recommendation system and their results compared. The experiments were carried out on an unprecedented database that represents the real use of six accounting software with thousands of users. To choose the algorithms, we searched in the literature for the most used algorithms in recommendation systems, among which were selected the first and second order Markov chains, LSTM, IndRNN and the standard Recurrent Neural Network (RNN). Three different training scenarios for the models were also analyzed, in which the user histories were shared at three different levels, namely: *(i)* user level training, where for each user a model was trained only with your personal history; *(ii)* company-level training, in which, for each company, a model is trained with the history of all users of the company and this model is shared among its users; *(iii)* system level training, in which a model is generated for all users of the software based on the history of all users. The results showed that the neural network algorithms managed to obtain a slight advantage in the ability to predict next functionality in the three test scenarios, however, the Markov chains presented an extremely faster training time. From the results it was possible to conclude that the performance of all models managed to be satisfactory when instead of recommending only the next feature, a list of the most likely next features is recommended, with the best results presenting an accuracy of 73 % for IndRNN and the standard RNN networks. The contributions of this work are from this being a little explored subject, with the related works being limited to the recommendation of website pages until the use of an unprecedented database that represents real interactions of users of 6 software with its functionalities. In addition, the proposal for a recommendation system for the next functionality is commercially useful, with the ability to assist the productivity of companies in view of the time savings that they have when recommending the next functionality as well as assisting novice users in certain workflows.

**Keywords:** Recommendation system. Recurring Neural Networks. Markov chains. Machine Learning.



# Lista de ilustrações

|                                                                                                                                                                                                                                                                                                                                                       |    |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| Figura 2.1 – Aprendizado de uma função . . . . .                                                                                                                                                                                                                                                                                                      | 32 |
| Figura 2.2 – (a) Matriz de transição Q com 3 estados; (b) A soma de cada linha da matriz de transição é igual a 1. . . . .                                                                                                                                                                                                                            | 37 |
| Figura 2.3 – (a) Cadeia de Markov de primeira ordem; (b) Cadeia de Markov de segunda ordem. . . . .                                                                                                                                                                                                                                                   | 38 |
| Figura 2.4 – Modelo matemático simples de um neurônio . . . . .                                                                                                                                                                                                                                                                                       | 39 |
| Figura 2.5 – Rede perceptron multicamadas . . . . .                                                                                                                                                                                                                                                                                                   | 41 |
| Figura 2.6 – (a) Rede neural <i>feed-forward</i> ; (b) Rede neural recorrente. . . . .                                                                                                                                                                                                                                                                | 45 |
| Figura 2.7 – RNR desdobrada . . . . .                                                                                                                                                                                                                                                                                                                 | 45 |
| Figura 2.8 – Rede <i>feed-forward</i> - Um para um . . . . .                                                                                                                                                                                                                                                                                          | 46 |
| Figura 2.9 – RNR - Um para muitos . . . . .                                                                                                                                                                                                                                                                                                           | 46 |
| Figura 2.10–RNR - Muitos para um . . . . .                                                                                                                                                                                                                                                                                                            | 46 |
| Figura 2.11–RNR - Muitos para muitos . . . . .                                                                                                                                                                                                                                                                                                        | 47 |
| Figura 2.12–RNR - Muitos para muitos . . . . .                                                                                                                                                                                                                                                                                                        | 47 |
| Figura 2.13–RNR expandida . . . . .                                                                                                                                                                                                                                                                                                                   | 48 |
| Figura 2.14–Célula LSTM . . . . .                                                                                                                                                                                                                                                                                                                     | 51 |
| Figura 2.15–(a) RNR padrão; (b) IndRNN. . . . .                                                                                                                                                                                                                                                                                                       | 53 |
| Figura 3.1 – Visão geral dos problemas de recomendação sensíveis à sequência . . .                                                                                                                                                                                                                                                                    | 64 |
| Figura 4.1 – Representação de uso de funcionalidades recorrentes . . . . .                                                                                                                                                                                                                                                                            | 76 |
| Figura 4.2 – Descrição do problema de recomendação de próxima funcionalidade.<br>Blocos cinzas são o histórico de funcionalidades acessadas pelo usuário no passado. Blocos azuis representam as funcionalidades acessadas mais recentemente, enquanto que o bloco vermelho representa a próxima funcionalidade que o usuário deseja acessar. . . . . | 77 |
| Figura 4.3 – sequência de uso de funcionalidades . . . . .                                                                                                                                                                                                                                                                                            | 79 |
| Figura 4.4 – Predição sem <i>ranking</i> , apenas a uma funcionalidade é recomendada . .                                                                                                                                                                                                                                                              | 80 |
| Figura 4.5 – Predição com <i>ranking</i> , uma lista <i>top-5</i> é recomendada . . . . .                                                                                                                                                                                                                                                             | 80 |
| Figura 5.1 – (a) Distribuição da quantidade de sequências por usuários; (b) Distribuição da quantidade de sequências por empresas. . . . .                                                                                                                                                                                                            | 87 |
| Figura 5.2 – Quantidade de sequências por sistemas . . . . .                                                                                                                                                                                                                                                                                          | 87 |
| Figura 5.3 – Quantidade de itens por sequências. Média em vermelho . . . . .                                                                                                                                                                                                                                                                          | 88 |
| Figura 5.4 – Variação na quantidade de funcionalidades relevantes . . . . .                                                                                                                                                                                                                                                                           | 90 |
| Figura 5.5 – Processo de geração das subsequências . . . . .                                                                                                                                                                                                                                                                                          | 91 |
| Figura 5.6 – Acurácia por tamanho do <i>rank</i> da lista de recomendações . . . . .                                                                                                                                                                                                                                                                  | 92 |
| Figura 5.7 – Redes neurais recorrentes com tamanhos diferentes de <i>batch</i> . . . . .                                                                                                                                                                                                                                                              | 93 |

|                                                                                                                                                                                        |     |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Figura 5.8 – (a) Quantidade de ocorrências por transições de <i>batch</i> 1 na base de dados; (b) Quantidade de ocorrências por transições de <i>batch</i> 2 na base de dados. . . . . | 94  |
| Figura 5.9 – Acurácia por tamanho do <i>rank</i> da lista de recomendações . . . . .                                                                                                   | 95  |
| Figura 5.10–Redes neurais recorrentes com tamanhos diferentes de <i>batch</i> . . . . .                                                                                                | 96  |
| Figura 5.11–Acurácia por tamanho do <i>rank</i> da lista de recomendações . . . . .                                                                                                    | 97  |
| Figura 5.12–Acurácia por quantidade de sequências para cada sistema . . . . .                                                                                                          | 98  |
| Figura 5.13–Redes neurais recorrentes com tamanhos diferentes de <i>batch</i> . . . . .                                                                                                | 98  |
| Figura 5.14–Comparativo da acurácia para todos os cenários com <i>rank</i> 5 . . . . .                                                                                                 | 99  |
| Figura 5.15–(a) ARHR para <i>rank</i> 5; (b) NDCG para <i>rank</i> 5. . . . .                                                                                                          | 100 |

# Lista de tabelas

|                                                                                                                                       |     |
|---------------------------------------------------------------------------------------------------------------------------------------|-----|
| Tabela 3.1 – Possíveis resultados de uma recomendação de um item para um usuário.                                                     | 71  |
| Tabela 5.1 – Quantidade de sequências por usuários, empresas e sistemas na base de dados. . . . .                                     | 86  |
| Tabela 5.2 – Acurácia média e tempo de treinamento para as <i>top-5</i> recomendações. O desvio padrão é denotado por $\pm$ . . . . . | 101 |



# Lista de abreviaturas e siglas

|        |                                                                                        |
|--------|----------------------------------------------------------------------------------------|
| ARHR   | <i>Average Reciprocal Hit-Rate</i> – Taxa média de acerto recíproco                    |
| IndRNN | <i>Independently Recurrent Neural Network</i> – Rede Neural Recorrente Independente    |
| LSTM   | <i>Long Short-Term Memory</i> – Memória Longa de Curto Prazo                           |
| MC     | <i>Markov Chain</i> – Cadeia de Markov                                                 |
| NDCG   | <i>Normalized Discounted Cumulative Gain</i> – Ganho Cumulativo Descontado Normalizado |
| RNA    | Redes Neurais Artificiais                                                              |
| RNR    | Rede Neural Recorrente                                                                 |
| VA     | Variável Aleatória                                                                     |



# Lista de símbolos

|              |                                                          |
|--------------|----------------------------------------------------------|
| $\eta$       | Taxa de aprendizagem                                     |
| $\nabla$     | Vetor Gradiente                                          |
| $\mathbf{W}$ | Pesos da camada oculta de uma rede neural                |
| $\mathbf{U}$ | Pesos da camada de entrada de uma rede neural recorrente |
| $\mathbf{V}$ | Pesos da camada de saída de uma rede neural recorrente   |
| $diag$       | Diagonal da matriz                                       |
| $tanh$       | Função tangente hiperbólica                              |
| $\sigma$     | Função sigmóide                                          |
| $\odot$      | Produto Hadamard                                         |



# Sumário

|            |                                                                 |           |
|------------|-----------------------------------------------------------------|-----------|
| <b>1</b>   | <b>INTRODUÇÃO</b>                                               | <b>25</b> |
| <b>1.1</b> | <b>Motivações da Dissertação</b>                                | <b>26</b> |
| <b>1.2</b> | <b>Objetivos Geral e Específicos</b>                            | <b>28</b> |
| <b>1.3</b> | <b>Estrutura da Dissertação</b>                                 | <b>28</b> |
| <b>1.4</b> | <b>Produção Científica</b>                                      | <b>29</b> |
| <b>2</b>   | <b>APRENDIZADO DE MÁQUINA</b>                                   | <b>31</b> |
| <b>2.1</b> | <b>Visão Geral</b>                                              | <b>31</b> |
| <b>2.2</b> | <b>Modelos probabilísticos</b>                                  | <b>34</b> |
| 2.2.1      | Uma breve revisão de teoria da probabilidade                    | 35        |
| 2.2.1.1    | Variáveis aleatórias discretas                                  | 35        |
| 2.2.1.2    | Probabilidade conjunta                                          | 35        |
| 2.2.1.3    | Probabilidade condicional                                       | 35        |
| 2.2.1.4    | Distribuição marginal                                           | 35        |
| 2.2.1.5    | Regra de Bayes                                                  | 36        |
| 2.2.1.6    | Independência e independência condicional                       | 36        |
| 2.2.2      | <i>Markov Chain</i> (Cadeias de Markov)                         | 36        |
| <b>2.3</b> | <b>Redes Neurais Artificiais (RNA)</b>                          | <b>39</b> |
| 2.3.1      | Modelo de McCulloch e Pitts                                     | 39        |
| 2.3.2      | Perceptron simples                                              | 40        |
| 2.3.3      | Algoritmo de retropropagação do erro ( <i>Backpropagation</i> ) | 41        |
| 2.3.4      | Redes Neurais Recorrentes (RNR)                                 | 44        |
| 2.3.4.1    | Visão geral                                                     | 44        |
| 2.3.4.2    | Alguns exemplos de padrões de design importantes para RNR       | 46        |
| 2.3.4.3    | Cálculo do gradiente                                            | 47        |
| 2.3.5      | Long Short-Term Memory (LSTM)                                   | 50        |
| 2.3.6      | Independently Recurrent Neural Network (IndRNN)                 | 52        |
| <b>2.4</b> | <b>Conclusão</b>                                                | <b>54</b> |
| <b>3</b>   | <b>SISTEMAS DE RECOMENDAÇÃO</b>                                 | <b>55</b> |
| <b>3.1</b> | <b>Uma breve história</b>                                       | <b>55</b> |
| <b>3.2</b> | <b>Classificação dos Sistemas de Recomendações</b>              | <b>56</b> |
| 3.2.1      | Modelos Básicos de Sistemas de Recomendação                     | 57        |
| 3.2.1.1    | Filtragem colaborativa                                          | 57        |
| 3.2.1.2    | Filtragem baseada em conteúdo                                   | 59        |
| 3.2.1.3    | Sistemas de recomendação baseados no conhecimento               | 59        |

|            |                                                                                       |           |
|------------|---------------------------------------------------------------------------------------|-----------|
| 3.2.1.4    | Sistemas de recomendação demográfica . . . . .                                        | 60        |
| 3.2.1.5    | Sistema de recomendação baseado na comunidade . . . . .                               | 60        |
| 3.2.1.6    | Sistemas híbridos de recomendação . . . . .                                           | 60        |
| 3.2.2      | Sistemas de recomendação sensíveis ao contexto . . . . .                              | 61        |
| 3.2.2.1    | Sistemas de recomendação baseados em localização . . . . .                            | 61        |
| 3.2.2.2    | Sistemas de recomendação social . . . . .                                             | 62        |
| 3.2.2.3    | Sistemas de recomendação sensíveis ao tempo . . . . .                                 | 62        |
| 3.2.3      | Sistemas de Recomendações Sensíveis à Sequência . . . . .                             | 63        |
| 3.2.3.1    | Caracterização dos sistemas de recomendação sensíveis à sequências . . . . .          | 64        |
| 3.2.3.2    | Categorização das tarefas em sistemas de recomendação sensíveis à sequência . . . . . | 65        |
| <b>3.3</b> | <b>Avaliando sistemas de recomendação . . . . .</b>                                   | <b>68</b> |
| 3.3.1      | Avaliando sistemas de recomendação sensíveis à sequência . . . . .                    | 69        |
| 3.3.1.1    | Particionamento de conjunto de dados . . . . .                                        | 69        |
| 3.3.1.2    | Definição dos itens de alvos . . . . .                                                | 70        |
| 3.3.2      | Métricas de Avaliação . . . . .                                                       | 71        |
| 3.3.2.1    | Acurácia . . . . .                                                                    | 71        |
| 3.3.2.2    | <i>Hit Rate</i> (HR) . . . . .                                                        | 72        |
| 3.3.2.3    | <i>Average Reciprocal Hit-Rate</i> (ARHR) . . . . .                                   | 72        |
| 3.3.2.4    | <i>Normalized Discounted Cumulative Gain</i> (NDCG) . . . . .                         | 73        |
| <b>3.4</b> | <b>Conclusão . . . . .</b>                                                            | <b>74</b> |
| <b>4</b>   | <b>SISTEMA DE RECOMENDAÇÃO DE PRÓXIMA FUNCIONALIDADE . . . . .</b>                    | <b>75</b> |
| <b>4.1</b> | <b>Apresentação do problema . . . . .</b>                                             | <b>75</b> |
| 4.1.1      | Generalização das sequências . . . . .                                                | 77        |
| 4.1.2      | <i>Ranking</i> de próximas funcionalidades . . . . .                                  | 79        |
| 4.1.3      | Categorização do sistema de recomendação de próxima funcionalidade . . . . .          | 81        |
| <b>4.2</b> | <b>Definição formal . . . . .</b>                                                     | <b>82</b> |
| <b>4.3</b> | <b>Conclusão . . . . .</b>                                                            | <b>83</b> |
| <b>5</b>   | <b>EXPERIMENTOS E RESULTADOS . . . . .</b>                                            | <b>85</b> |
| <b>5.1</b> | <b>Base de dados . . . . .</b>                                                        | <b>85</b> |
| <b>5.2</b> | <b>Metodologia . . . . .</b>                                                          | <b>88</b> |
| <b>5.3</b> | <b>Métricas . . . . .</b>                                                             | <b>91</b> |
| <b>5.4</b> | <b>Resultados . . . . .</b>                                                           | <b>92</b> |
| 5.4.1      | Usuário . . . . .                                                                     | 92        |
| 5.4.2      | Empresa . . . . .                                                                     | 95        |
| 5.4.3      | Sistemas . . . . .                                                                    | 96        |
| 5.4.4      | Comparativo geral . . . . .                                                           | 99        |

|            |                                                |                |
|------------|------------------------------------------------|----------------|
| <b>6</b>   | <b>CONCLUSÃO E TRABALHOS FUTUROS . . . . .</b> | <b>103</b>     |
| <b>6.1</b> | <b>Conclusão . . . . .</b>                     | <b>103</b>     |
| <b>6.2</b> | <b>Trabalhos Futuros . . . . .</b>             | <b>104</b>     |
|            | <br><b>REFERÊNCIAS . . . . .</b>               | <br><b>105</b> |



# 1 Introdução

Graças ao aumento da capacidade de processamento dos computadores e aos preços acessíveis, tornou-se cada vez mais comum a sua utilização para realizar tarefas que antes exigiam grandes quantidades de papel, tais como preencher formulários, escrever relatórios, artigos e etc. Empresas que necessitavam lidar com bastante informação, tanto a respeito de seus clientes quanto da própria empresa em si, passaram então a utilizar os computadores no auxílio dessas tarefas através de softwares que tinham como objetivo cadastrar, visualizar e manipular informações. O aumento contínuo da capacidade de processamento permitiu o desenvolvimento de softwares cada vez mais completos em termos de funcionalidades e mais integrados a outros softwares, passando assim de softwares isolados à grandes sistemas computacionais.

Tendo em vista isso, sistemas com diversas funcionalidades que precisam atender a muitas necessidades requerem, conseqüentemente, uma grande quantidade de telas e menus para acomodar essas funcionalidades. Para os usuários desses sistemas que desejam acessar alguma funcionalidade ou seguir um certo fluxo de trabalho, é necessário, muitas vezes, gastar um certo tempo em menus e telas de funcionalidades intermediárias até encontrar a funcionalidade desejada. Dessa forma, muitas vezes esse processo de encontrar a tela desejada entre os diversos menus e telas intermediárias tornam custosas tarefas simples, que acabam necessitando de muitos cliques. É nesse contexto que a necessidade de recomendação de funcionalidades surge, podendo ser descrita de forma simplificada pela necessidade de recomendar a próxima funcionalidade com maior probabilidade de ser acessada pelo usuário. Esta capacidade de recomendação é especialmente relevante em sistemas com uma quantidade elevada de telas, podendo tais recomendações oferecerem um aumento considerável na produtividade e aprendizagem do sistema por parte dos seus usuários. Diversas estratégias e algoritmos podem ser adotados para se gerar recomendações ao usuário, sendo essa uma abrangente área de estudo denominada de Sistemas de Recomendação (*Recommender Systems*).

Sistemas de recomendação são aplicações que têm como objetivo auxiliar o usuário a selecionar itens de seu interesse. Vários serviços *online* usam sistemas de recomendação para indicar aos usuários ou visitantes do site itens adicionais que podem ser interessantes para eles. Exemplos mais frequentes de uso de sistemas de recomendação podem ser observados em sites de comércio eletrônico e redes sociais, que recomendam sempre produtos semelhantes ao que o usuário está procurando ou perfis de usuário na rede social que o usuário queira adicionar à sua rede. Além disso, os sistemas de recomendação podem otimizar o tempo de trabalho do usuário em determinadas tarefas através da recomendação de funcionalidades relevantes a essas atividades.

Determinar qual o item mais adequado para um determinado usuário não é um problema trivial, visto que isso exige um conhecimento prévio das preferências do usuário. Além disso, é natural que as preferências do usuário mudem com o passar do tempo, sendo necessário então atualizar o conhecimento sobre tais preferências. Algoritmos de aprendizado de máquina possuem tal capacidade de aprendizado e adaptação, de modo que estão presentes entre os principais trabalhos considerados o estado da arte em sistemas de recomendação.

## 1.1 Motivações da Dissertação

A utilização de técnicas aprendizado de máquina em sistemas de recomendação vêm se tornando algo cada vez mais comum, principalmente devido a grande quantidade de dados de usuários coletada. Esse aumento tem proporcionado grandes avanços no desenvolvimento de recomendadores cada vez mais precisos e capazes de se adaptar a mudanças. De maneira geral, esses avanços contribuem muito para a experiência do usuário com determinado sistema, tanto através de dicas como de sugestões do que fazer.

Apesar de já ser quase de praxe receber recomendações personalizadas de produtos ou serviços enquanto se navega pela internet, em softwares empresariais o que se tem normalmente são apenas tutoriais, treinamentos por parte das empresas e/ou algumas dicas por parte do próprio software quando se abre determinada funcionalidade. Uma camada de inteligência capaz de aprender com o comportamento do usuário e lhe auxiliar em suas tarefas, de modo a otimizar seu trabalho, ainda não é algo comum. Trabalhos correlatos encontrados foram sistema de recomendação de comandos para aplicações de software (MATEJKA et al., 2009) e sistemas de recomendação de páginas dentro de sites (FU; BUDZIK; HAMMOND, 2000; MOBASHER; COOLEY; SRIVASTAVA, 2000; MOBASHER et al., 2002), que possuem tal correlação pelo fato de que é comum em alguns softwares que cada funcionalidade fique em uma página ou tela própria.

Embora a área sistemas de recomendação seja relativamente nova, já existe uma série de sub-áreas que podem auxiliar o desenvolvimento de novos sistemas de recomendação de acordo com cada necessidade. Aggarwal (2016b), Ricci, Rokach e Shapira (2011b) apresentam em detalhes a história dos sistemas de recomendação bem como a evolução dos diferentes tipos de sistemas de recomendação. Além disso, ainda que encontrar trabalhos de sistemas de recomendação de funcionalidades de softwares na literatura seja difícil, é possível se fazer algumas correlações com outros problemas, tendo em vista que, no geral, a maioria dos sistemas de recomendação têm um objetivo em comum: recomendar itens relevantes para determinado usuário com base no conhecimento que se tem sobre ele. De maneira geral, ao invés de pensar no problema que se está lidando em si, pode-se pensar na forma como as recomendações são feitas para esse determinado problema. A exemplo

disso tem-se que sistemas de recomendação de produtos em *e-commerce* em sua maioria utilizam-se de **filtragem colaborativa**, recomendando os mesmos itens a usuário com gostos semelhantes. Já sistemas de recomendação de músicas podem tanto recomendar músicas baseado em filtragem colaborativa como na sequência de determinado usuário gosta de ouvir suas músicas, além de poder utiliza **filtragem baseada em conteúdo** para recomendar músicas semelhantes as que o usuário gosta. Dos trabalhos que se propõem a classificar em grupos os sistemas de recomendação, foram de grande relevância para este trabalho (RICCI; ROKACH; SHAPIRA, 2011a; AGGARWAL, 2016b; PARK et al., 2012).

O problema de recomendação de próxima funcionalidade de softwares se difere dos sistemas de recomendações clássicos principalmente pela componente temporal/sequencial, pois baseia-se nas sequências de acesso as funcionalidades. Nesse contexto, Quadrana, Cremonesi e Jannach (2018) apresentam uma ótima revisão de modelos e arquiteturas propostas para sistemas de recomendação baseados em sequências (*sequence-aware recommender systems*). Dentre as abordagens mais comuns, pode-se destacar modelos probabilísticos baseados nas cadeias de Markov, tais como MDP (Markov Decision Processes) (SHANI; HECKERMAN; BRAFMAN, 2005); Árvores de Contexto (HE et al., 2009), que correspondem a cadeias de Markov de ordem variável; e Modelos de Markov Ocultos (AGHDAM et al., 2015), em que variáveis latentes são introduzidas para capturar dependências de mais longa duração. Além disso, Redes Neurais Recorrentes (RNR) têm também sido aplicadas a esse tipo de sistema de recomendação. Dentre os trabalhos mais relevantes, vale ressaltar os recentes trabalhos de Hidasi et al. (2015), Jannach e Ludewig (2017), que utilizam RNR e vizinhos mais próximos para recomendações baseadas em sessão, bem como o trabalho de Zhang et al. (2014) para predição de cliques em mecanismos de busca.

Do exposto, tem-se que a tarefa de recomendação de funcionalidades de um software pode ser implementada como um sistema de recomendação baseado em sequências, utilizando-se de algoritmos de recomendação e métricas já estabelecidas nesta área de pesquisa para validar os resultados. Com isso, este trabalho apresenta o estudo de um sistema de recomendação de próxima funcionalidade em softwares comparando a utilização de 5 algoritmos de aprendizado de máquina comumente aplicados à problemas semelhantes ao de recomendar próxima funcionalidade, sendo estes: Cadeias de Markov de primeira e segunda ordem, Long short-term memory (LSTM), Independently Recurrent Neural Network (IndRNN) e a Rede Neural Recorrente (RNR) padrão. Para tanto, foram conduzidos experimentos com dados reais que representam o histórico de uso de milhares de usuário de todo o Brasil em seis sistemas de contabilidade desenvolvido pela empresa Fortes Tecnologia <sup>1</sup>. Além de estudar o uso de sistema de recomendação para recomendar próximas funcionalidades, este trabalho teve como objetivo desenvolver uma ferramenta

---

<sup>1</sup> <https://www.fortestecnologia.com.br>

capaz de ajudar os usuário na utilização desses sistemas a partir das recomendações.

As contribuições deste trabalho são variadas, indo desde o fato de abordar um problema pouco explorado, que é a recomendação de próxima funcionalidade em softwares; a implementação um sistema de recomendação, utilizando algoritmos de recomendação mais recentes no estado da arte; até a utilização de dados reais, possibilitando um estudo prático do uso de tais algoritmos. Até onde se tem conhecimento, a combinação dessas três contribuições torna este trabalho inovador na área de sistemas de recomendação de próxima funcionalidade, servindo como fonte de consulta para softwares com problemas semelhantes que buscam melhorar a interação e o fluxo de trabalho de seus usuários.

## 1.2 Objetivos Geral e Específicos

O objetivo geral da presente dissertação consiste em avaliar a performance de algoritmos comumente utilizados em sistemas de recomendação e classificação de sequências no problema de recomendação de próxima funcionalidade em softwares.

O escopo geral pode ainda ser agrupado em três objetivos específicos principais:

- OE1** Apresentar o problema de recomendação de próxima funcionalidade em softwares como um sistema de recomendação para o auxílio de usuários.
- OE2** Avaliar a performance dos algoritmos comumente usados em sistemas de recomendações semelhantes na base de dados.
- OE3** Destacar as melhores metodologia para implementação do sistema de recomendação de próxima funcionalidade em softwares.

O foco dos objetivos **OE2** e **OE3** são no aspecto algorítmico e metodológico das soluções a serem desenvolvidas. Enquanto que no objetivo **OE2** o interesse principal está na implementação e avaliação dos algoritmos a serem utilizados, o objetivo **OE3** busca destacar a metodologia que melhor se adéque ao problema estudado.

## 1.3 Estrutura da Dissertação

O conteúdo desta dissertação está organizado com a seguinte estrutura:

- Capítulo 2, trás uma revisão teórica dos algoritmos de aprendizado de máquina utilizados.
- No capítulo 3, é abordado, do ponto de vista histórico e funcional, os diversos tipos de sistemas de recomendação e como o problema de recomendação de próxima funcionalidade em softwares se enquadra no que é relatado na literatura.
- O capítulo 4 trás uma apresentação conceitual do problema como um sistema de recomendação e sua formulação matemática.

- Capítulo 5, é apresentado a base de dados em detalhes bem como metodologia aplicada para realização dos experimentos. É também, neste capítulo, apresentado os resultados e discussão dos experimentos realizados.
- E por fim, o capítulo Conclusão (Capítulo 6), apresenta a conclusão do trabalho e os trabalhos futuros.

## 1.4 Produção Científica

Durante o período de desenvolvimento deste trabalho de mestrado, foi escrito e publicados o seguinte trabalho científico:

**Prata V.R. et al.** “*A Novel Recommendation System for Next Feature in Software*”. In: *Intelligent Data Engineering and Automated Learning – IDEAL 2019*. IDEAL 2019. Lecture Notes in Computer Science, vol 11871. Springer, Cham.



## 2 Aprendizado de Máquina

Visto como um subcampo de inteligência artificial, o aprendizado de máquina (do inglês *machine learning*, ML), é o estudo de algoritmos que buscam melhorar seu desempenho automaticamente com a experiência (MITCHELL, 1997). As técnicas de aprendizado de máquina são as bases fundamentais para a maioria das aplicações inteligentes mais recentes e presentes no nosso dia-a-dia, tais como reconhecimento facial, mecanismos de pesquisa, filtros anti-spam, reconhecimento de voz em celulares, assistentes virtuais inteligentes e diversos sistemas de recomendações, que vão desde recomendações de produtos e serviços à sugestões de amigos em redes sociais.

O objetivo deste capítulo é fornecer o conhecimento básico das técnicas de aprendizado de máquina empregadas tanto neste trabalho como nos trabalhos correlatos aqui citados. Além disso, este capítulo oferece uma descrição aprofundada dos algoritmos e metodologias avaliadas neste estudo, sendo de fundamental importância para compreender os resultados obtidos nos experimentos.

### 2.1 Visão Geral

A ideia geral por trás de todo algoritmo de aprendizado de máquina é aprender a resolver determinados problemas automaticamente, de modo que não haja interferência humana no processo de aprendizado. Ou seja, desejamos programar computadores para que eles possam “aprender” a partir das informações disponíveis em formas de dados.

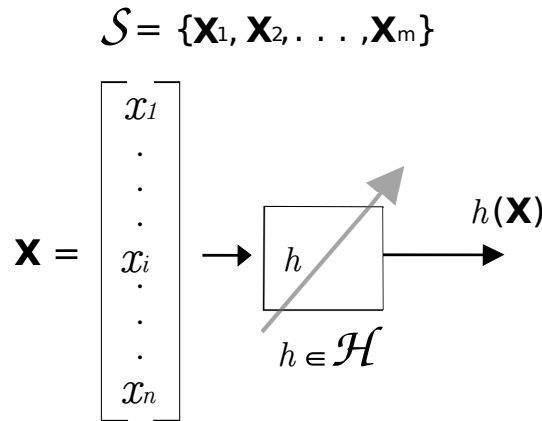
De maneira geral, o aprendizado é o processo de conversão da experiência em habilidades ou conhecimento. O processo de um algoritmo de aprendizado de máquina é resumido de modo que os dados de entrada representam a experiência enquanto os de saída algum conhecimento ou habilidade aprendida. Geralmente, essa saída assume a forma de outro programa de computador com poder de executar alguma tarefa, sendo também chamado de modelo de aprendizado de máquina. A forma como esse modelo gerado consegue executar tarefas se deve ao fato de o modelo em si ser uma função matemática que descreve o conjunto de dados aprendido.

Com isso, o processo de aprendizado de uma função pode ser expresso da seguinte maneira: considerando a existência de uma função  $f$  que descreve um determinado problema, o objetivo do algoritmo é aprender essa função a partir de amostras do problema. Como essa função é desconhecida, o que tem-se é uma hipótese, denotada por  $h$ , sobre a função  $f$ , obtida com base em um conjunto de treinamento  $\mathcal{S}$  de  $m$  exemplos de vetores de entrada  $\mathbf{x} = (x_1, x_2, \dots, x_n)$  de  $n$  componentes. Assim,  $f$  e  $h$  são funções dos vetores de entrada

$\mathbf{x}$ , de modo que  $h(\mathbf{x})$  é a saída. Além disso, assume-se a priori que a função hipotética  $h$  é selecionada de uma classe de funções  $\mathcal{H}$ , podendo  $f$  pertencer ou não a essa classe ou a um subconjunto desta classe. Quando sabe-se, mesmo que aproximadamente, o valor de  $f$  para cada entrada  $m$  do conjunto de treinamento  $\mathcal{S}$ , diz-se que esse aprendizado é supervisionado. Assim, quando encontra-se uma hipótese  $h$  que se aproxima de  $f$  para os valores de  $\mathcal{S}$ , então tem-se que essa hipótese é uma boa estimativa de  $f$ . Já no aprendizado não supervisionado, simplesmente tem-se um conjunto de vetores de treinamento sem valores de função conhecidos para eles.

Na Figura 2.1 é descrito esse processo de aprendizado, na qual, a cada entrada  $\mathbf{x}$ , a hipótese  $h$  é atualizada de acordo com a busca no espaço das funções  $\mathcal{H}$  (bloco com a seta).

Figura 2.1 – Aprendizado de uma função



Fonte: NILSSON, 1996.

Da mesma forma como o processo de aprendizado ocorre de diferentes maneiras entre os animais, principalmente entre humanos, um algoritmo de aprendizado de máquina pode aprender de diferentes formas. Shalev-Shwartz e Ben-David (2014) descrevem quatro parâmetros nos quais os paradigmas de aprendizagem podem ser classificados, sendo estes:

- **Supervisionado versus Não Supervisionado:** Como já descrito, a aprendizagem pode ser vista como um processo de usar experiência para ganhar especialização, nisso a diferença do aprendizado supervisionado é que são utilizados dados rotulados no processo de aprendizagem para guiar o processo. Ou seja, os valores de  $f$  para as  $m$  amostras no conjunto de treinamento  $\mathcal{S}$  são conhecidas, logo, se for encontrada uma hipótese  $h$  que se aproxima de  $f$  para os membros de  $\mathcal{S}$ , então esta hipótese será uma boa estimativa de  $f$ , principalmente se  $\mathcal{S}$  for grande. Esse método se assemelha a um aluno que aprende na presença de seu professor, com cada erro cometido pelo aluno sendo corrigido pelo professor. Já no aprendizado não supervisionado os dados não possuem rótulos, isso é, não há

uma função  $f$  conhecida. Nesse caso, o algoritmo processa os dados de entrada com o objetivo de obter alguma versão resumida desses dados ou agrupá-los em subconjuntos de objetos semelhante. O aprendizado supervisionado normalmente é usado no contexto de classificação, quando queremos mapear a entrada para rótulos de saída, ou regressão, quando queremos mapear a entrada para uma saída contínua. Já as tarefas mais comuns no aprendizado não supervisionado são *clustering* (agrupamento), aprendizado de representação e estimativa de densidade.

- **Aprendizado ativo versus passivo:** A forma como o algoritmo se comporta no processo de aprendizagem define se ele é ativo ou passivo. Um algoritmo ativo interage com o ambiente no momento do treinamento, por exemplo, fazendo perguntas ou realizando experimentos, como perguntar ao usuário se determinada recomendação é útil, enquanto um algoritmo passivo apenas observa as informações fornecidas, tal como os rótulos, sem interagir ativamente. No exemplo de professor/aluno, temos que um aluno passivo apenas observa o que o professor fala, enquanto que o aluno ativo faz perguntas e pede explicações sobre o que não está claro.
- **Solicitude do professor:** A solicitude do professor é definida com relação ao quanto o processo de aprendizagem ajuda o algoritmo durante o treinamento, isso é, o quanto de informação é dada e como essa informação é dada. Quando apenas as informações mais relevantes são utilizadas no treinamento, isso ajuda o algoritmo a alcançar mais rapidamente seu objetivo. Porém, algumas vezes não há como saber quais informações são úteis, sendo necessário uma quantidade maior de dados e tempo para completar o treinamento. Além disso, o processo de aprendizado pode ocorrer de modo que os dados de entrada do algoritmo são gerados por um agente adversário com o objetivo de enganar ou representar o pior cenário que o modelo aprendido enfrentará. Estes três casos podem ser exemplificados como um professor que ajuda o aluno, um professor que não faz nada e um professor adversário, de modo que, se o aluno pode aprender contra um professor adversário, certamente terá sucesso na interação com qualquer outro professor.
- **Aprendizagem *online* versus *offline*/Batch:** De forma resumida, o aprendizado *offline*, também chamado de aprendizado *batch* (bloco), é uma abordagem na qual todos os dados são consumidos em blocos durante a etapa de treinamento para calcular a função  $h$ . Com isso, o modelo resultante não possui a capacidade de atualizar o que aprendeu após terminada a etapa de aprendizado. Por outro lado, no aprendizado *online* o treinamento é feito de forma incremental, com os dados de treinamento sendo disponibilizados de forma sequencial e o modelo atualizado a cada nova amostra. Nessa abordagem, o processo de treinamento

nunca termina, e o algoritmo está sempre atualizando o modelo. Apesar disso, quando se trata de padrões com pouca variedade e com uma grande quantidade de dados disponíveis, o aprendizado *offline* apresenta uma vantagem sobre o *online*, pois terminado o treinamento, o modelo já tem o domínio sobre os dados, enquanto que o aprendizado *online* irá errar bastante até ter consumido uma quantidade de dados grande o suficiente.

A forma como esses quatro parâmetros são configurados define como o conjunto de treinamento  $\mathcal{S}$  será usado para produzir uma função hipotética. Um filtro de *spam*, por exemplo, geralmente tem o treinamento supervisionado (com dados rotulados), passivo e *offline*.

Com o decorrer dos anos, os trabalhos em aprendizado de máquina começaram a convergir de várias fontes, de modo que diferentes tradições trouxeram métodos diferentes e vocabulários diferentes que, sendo assimilados, tornaram o aprendizado de máquina um campo interdisciplinar, compartilhando tópicos comuns com os campos matemáticos da estatística, teoria da informação, teoria dos jogos, otimização e diversas outras disciplinas separadas que contribuíram para o aprendizado de máquina, tais como árvores de decisões, redes neurais artificiais, aprendizado bayesiano, instance-based learning, algoritmos genéticos e etc. Apesar de todas essas áreas terem um papel importante para os estudos de aprendizado de máquina, este trabalho focou em métodos probabilísticos e redes neurais artificiais.

## 2.2 Modelos probabilísticos

Calcular a probabilidade de um evento observado pertencer a uma classe de eventos já conhecida é, certamente, uma das maneiras mais básicas de se criar um modelo de aprendizado de máquina. Da classificação à predições de eventos futuros dado o passado, a teoria da probabilidade oferece diversas ferramentas que podem ser aplicadas a qualquer problema de aprendizado de máquina envolvendo incertezas.

Alguns exemplos de modelos probabilísticos amplamente encontrados na literatura são: Regressão Logística, Classificadores Bayesianos e Cadeias de Markov, dentre outros. Para o problema estudado neste trabalho, foram utilizados modelos de cadeias de markov devido as características do problema. Para melhor compreender esses modelos, alguns conceitos de teoria da probabilidade são necessários, dessa forma, a seção a seguir trás uma breve revisão sobre esses conceitos.

## 2.2.1 Uma breve revisão de teoria da probabilidade

### 2.2.1.1 Variáveis aleatórias discretas

Uma **variável aleatória (VA)** é uma função definida sobre um espaço amostral  $S$  que atribui um valor real a cada elemento desse espaço amostral. Assim, em um experimento com espaço amostral  $S$ , uma variável aleatória  $X$  atribui um valor numérico  $X(s)$  para cada resultado possível  $s$  do experimento. Quando o número de valores possíveis de  $X$  for finito ou infinito enumerável, tem-se que  $X$  é uma **variável aleatória discreta**. A probabilidade do evento  $X = x$  é denotada por  $p(x) = P(X = x)$ , onde  $p(\cdot)$  denota a função de massa de probabilidade (*probability mass function*, PMF).

### 2.2.1.2 Probabilidade conjunta

A probabilidade conjunta é a probabilidade de dois eventos ocorrerem juntos. Assim, probabilidade conjunta é definido da seguinte maneira:

$$p_{X,Y}(x, y) = P(X = x, Y = y).$$

### 2.2.1.3 Probabilidade condicional

Sejam  $X$  e  $Y$  dois eventos associados ao experimento  $\mathcal{E}$ . A probabilidade condicional do evento  $Y$ , dado que o evento  $X = x$  ocorreu é definido da seguinte forma:

$$P(Y = y | X = x) = \frac{P(X = x, Y = y)}{P(X = x)}, \text{ desde que } P(X = x) \geq 0,$$

em que  $P(X = x, Y = y)$  é a probabilidade conjunta dos eventos  $X$  e  $Y$ . Uma das consequências dessa expressão é a **regra do produto**, em que, isolando a probabilidade conjunta tem-se:

$$P(X = x, Y = y) = P(X = x | Y = y)P(Y = y).$$

### 2.2.1.4 Distribuição marginal

A probabilidade de um evento para uma variável aleatória, independentemente dos resultados de outra variável aleatória, é chamada probabilidade marginal. Já a probabilidade marginal de uma variável aleatória na presença de variáveis aleatórias adicionais é conhecida como distribuição de probabilidade marginal, ou simplesmente distribuição marginal. Assim, dada uma distribuição conjunta em dois eventos  $P(X, Y)$ , a distribuição marginal é definida da seguinte forma:

$$P(X = x) = \sum_y P(X = x, Y = y) = \sum_y P(X = x | Y = y)P(Y = y),$$

em que são somados todos os estados possíveis de  $B$ . É possível definir  $P(B)$  de forma similar. Essa expressão é também chamada de **regra da soma** ou regra da probabilidade total. A probabilidade marginal é diferente da probabilidade condicional por considerar a união de todos os eventos para a segunda variável em vez da probabilidade de um único evento.

### 2.2.1.5 Regra de Bayes

Combinando a definição de probabilidade condicional com as regras do produto e da soma, obtém-se uma das mais importantes regras em probabilidade, a regra de Bayes, também chamada de Teorema de Bayes. Essa regra permite calcular a probabilidade de um evento  $X = x$ , baseado no conhecimento *a priori*  $Y = y$ . Assim, a regra de Bayes é definida por:

$$P(X = x \mid Y = y) = \frac{P(X = x, Y = y)}{P(Y = y)} = \frac{P(X = x)P(Y = y \mid X = x)}{\sum_{x'} P(X = x')P(Y = y \mid X = x')}.$$

### 2.2.1.6 Independência e independência condicional

Dado que as duas variáveis aleatórias  $X$  e  $Y$  são independentes, tem-se que, saber o valor de  $X$  não fornece informações sobre o valor de  $Y$  e vice-versa. A definição formal dessa ideia é:  $X$  e  $Y$  são consideradas independentes se

$$X \perp Y \iff P(X = x, Y = y) = P(X = x)P(Y = y),$$

para todos os  $x, y \in \mathbb{R}$ .

Muito frequentemente, a maioria das variáveis pode influenciar a maioria das outras variáveis, o que torna independência incondicional rara. Entretanto, geralmente é possível medir essa influência através de outras variáveis, em vez de medir diretamente. Desse modo, diz-se que  $X$  e  $Y$  são **condicionalmente independentes** dado  $Z$ , se:

$$X \perp Y \mid Z \iff P(X, Y \mid Z) = P(X \mid Z)P(Y \mid Z).$$

## 2.2.2 Markov Chain (Cadeias de Markov)

A cadeia de markov, propriamente dita, não é um algoritmo, mas sim um modelo estocástico que descreve uma sequência discreta (ou contínua) de eventos possíveis em que a probabilidade de cada evento depende apenas do estado do evento anterior. Além disso, o espaço dos estados de uma cadeia de Markov pode também ser discreto ou contínuo, porém, neste trabalho apenas as cadeias de Markov de estado e tempo discretos e finitos serão vistas.

Formalmente, uma sequência de variáveis aleatórias  $X_0, X_1, \dots$  assumindo valores no espaço de estados  $\{1, 2, \dots, M\}$  é chamada uma cadeia de Markov se para todo  $n \geq 0$ ,

$$P(X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, \dots, X_0 = i_0) = P(X_{n+1} = j | X_n = i). \quad (2.1)$$

O termo  $P(X_{n+1} = j | X_n = i)$  denota a **probabilidade de transição** do estado  $i$  para o estado  $j$ . Além disso, assume-se que a probabilidade de transição  $P(X_{n+1} = j | X_n = i)$  é independente do tempo, sendo a mesma para todo  $n$ . Devido a isso, essa cadeia é chamada de homogênea, estacionária ou invariante no tempo.

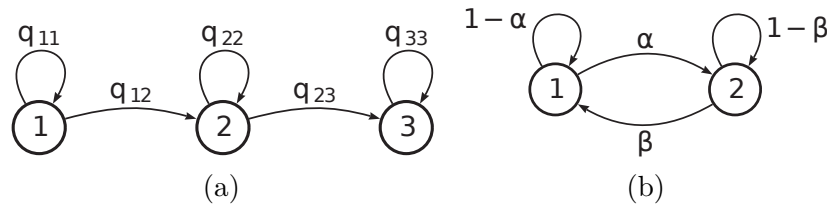
Baseado na definição 2.1, tem-se ainda que, dado todo o histórico  $X_0, X_1, \dots, X_n$ , apenas o termo mais recente,  $X_n$ , é importante para prever  $X_{n+1}$ . Essa condição é chamada de **propriedade de Markov**. A Propriedade de Markov pode ser resumidamente expressa como: dado o tempo  $n$ , que representa o presente,  $n - 1$ , que representa o passado, e  $n + 1$  que representa o futuro, a propriedade de Markov diz que, dado o presente, o passado e o futuro são **condicionalmente independentes**. Matematicamente, isso é expresso por  $X_{n+1} \perp X_{n-1} | X_n$ . Assim, conforme afirma Blitzstein e Hwang (2014), as cadeias de Markov são um meio termo entre a independência completa e a dependência completa.

A dinâmica de uma cadeia de Markov se dá a partir da probabilidade de transição do estado  $i$  para o estado  $j$ , expressa por  $P(X_{n+1} = j | X_n = i)$ . Essa informação pode ser genericamente representada por uma matriz chamada **matriz de transição**. No caso de  $M$  estados discretos possíveis, a matriz de transição  $Q$  é uma matriz  $M \times M$  dada por

$$Q = [q_{ij} = P(X_{n+1} = j | X_n = i)]. \quad (2.2)$$

Tomando como base a Figura 2.2a, é possível entender melhor como a matriz de transição expressa a probabilidade  $q_{ij}$  entre cada estado. Além disso,  $Q$  é uma matriz não-negativa cuja soma dos elementos de cada linha é igual a 1. Assim, a probabilidade de um estado  $i$  permanecer inalterado é igual a  $1 - \alpha$ , sendo  $\alpha$  a soma das probabilidades de transição de  $i$  para algum estado  $j$ , conforme exemplificado na Figura 2.2b.

Figura 2.2 – (a) Matriz de transição  $Q$  com 3 estados; (b) A soma de cada linha da matriz de transição é igual a 1.

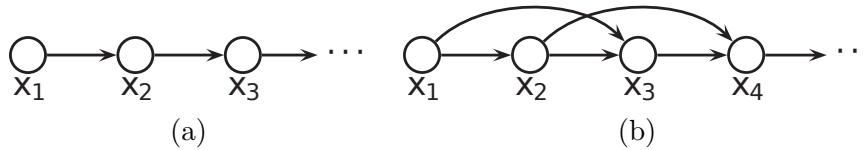


Fonte: MURPHY, 2012.

A partir dessas definições, percebe-se que a ideia básica por trás de uma cadeia de Markov é assumir que  $X_{n-1}$  captura todas as informações relevantes para prever o futuro,

sendo essa estatística considerada suficiente. Entretanto, a suposição de que o passado imediato,  $X_{n-1}$ , captura todas as informações relevantes sobre todo o histórico  $X_{1..n-2}$  é um pouco forte (MURPHY, 2012). Apesar de ser considerada suficiente para muitas aplicações, quando têm-se grandes sequências de eventos correlacionados, olhar apenas para o último estado pode não ser o bastante, assim, é possível flexibilizar um pouco essa independência condicional adicionando uma dependência de  $X_{n-2}$  à  $X_n$ . Essa cadeia é chamada de **cadeia de Markov de segunda ordem**. Note que a primeira cadeia de Markov aqui apresentada, que toma como relevante apenas  $X_{n-1}$ , é chamada também de **cadeia de Markov de primeira ordem**. Para exemplificar melhor a diferença, considere os dois grafos direcionados da Figura 2.3 representando duas sequências de eventos. O grafo 2.3a representa a cadeia de primeira ordem, pois apenas o estado do evento anterior é relevante para prever o estado do evento atual. Já na Figura 2.3b, observa-se que, além do estado anterior, também é levado em conta o estado anterior ao anterior, isso é,  $X_{n-1}$  e  $X_{n-2}$ , caso existam.

Figura 2.3 – (a) Cadeia de Markov de primeira ordem; (b) Cadeia de Markov de segunda ordem.



Fonte: MURPHY, 2012.

Para que a propriedade de Markov (que determina que um nó depende apenas de seus pais imediatos) não seja violada, é possível definir a cadeia de Markov de segunda ordem ampliando o espaço de estados, criando, assim, uma nova cadeia de Markov onde  $Y_n = (X_{n-1}, X_n)$  para  $n \geq 1$ , sendo  $Y_1, Y_2, \dots$  uma cadeia de Markov no espaço de estado  $\{(X_1, X_1), (X_1, X_2), (X_2, X_1), (X_2, X_2), \dots, (X_{n-2}, X_{n-1}), (X_{n-1}, X_{n-1}), \dots\}$ .

Da mesma forma, é possível lidar com dependências de ordem superior de maneira semelhante, ampliando ainda mais o espaço de estados para manter a propriedade de Markov. Dessa forma, é possível calcular as probabilidades de transição para escalas de tempo mais longas ao generalizarmos a matriz de transição da seguinte forma: a probabilidade de transição do estado  $i$  para  $j$  em uma cadeia de ordem  $m$  é a probabilidade de  $j$  ser o estado atual exatamente  $m$  passos após  $i$  ter sido o estado atual. Assim, a probabilidade  $q_{ij}^{(m)}$  é dada por

$$q_{ij}^{(m)} = P(X_{n+m} = j | X_n = i). \quad (2.3)$$

Entretanto, segundo Murphy (2012), mesmo a suposição de Markov de segunda

ordem pode ser inadequada se houver correlações de longo alcance entre as observações. Além disso, continuar construindo modelos de ordem cada vez mais elevados, aumentará cada vez mais o número de parâmetros, que poderão explodir.

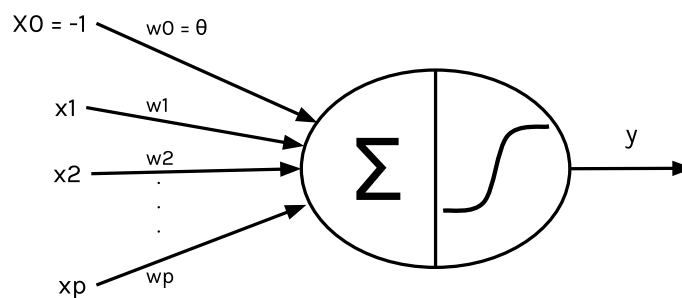
## 2.3 Redes Neurais Artificiais (RNA)

Inspirados pela hipótese de que a atividade mental consiste basicamente na interação eletroquímica entre os neurônios do cérebro animal, alguns pesquisadores pioneiros em inteligência artificial buscaram criar sistemas computacionais capazes de reproduzir esse comportamento artificialmente. Assim surgiram as redes neurais artificiais, com o primeiro trabalho da área publicado por McCulloch e Pitts (1943), sendo este também, reconhecidamente, o primeiro trabalho da área de inteligência artificial. Esse trabalho deu início a uma das ferramentas mais poderosas atualmente para o aprendizado de máquina atualmente, com modelos capazes de resolver problemas dos mais variados tipos, como visão computacional, processamento de linguagem natural, sistemas de recomendação e etc.

### 2.3.1 Modelo de McCulloch e Pitts

Em seu trabalho, McCulloch e Pitts (1943) apresentam um modelo matemático simples de um neurônio que é acionado quando uma combinação linear de suas entradas excede um limiar. O modelo matemático simples de um neurônio, que pode ser observado na Figura 2.4, é composto por  $p$  entradas  $\{x_i\}_{i=1}^p$  e possui  $p$  pesos sinápticos  $\{w_i\}_{i=1}^p$ . Além disso, o limiar de ativação do neurônio é denotado por  $\theta$ . Desse modo, a entrada da função de ativação é dada por  $\sum_{i=1}^p x_i w_i - \theta$ . A função de ativação pode ser tanto um limiar rígido, para o perceptron simples, como uma função logística, caso em que por vezes é utilizado o termo perceptron sigmoide (RUSSELL; NORVIG, 2010).

Figura 2.4 – Modelo matemático simples de um neurônio



Fonte: Próprio autor.

Apesar do pioneirismo de McCulloch e Pitts, o modelo por eles proposto possuía algumas limitações que, segundo eles, poderiam ser resolvidos ligando um grande número

de neurônios em redes com uma profundidade adequada, porém, na época, ninguém tinha ideia de como resolver o problema de calcular os pesos para cada uma desses neurônios.

### 2.3.2 Perceptron simples

O problema de como calcular os pesos foi solucionado por Rosenblatt (1958), em seu trabalho que foi o primeiro algoritmo de redes neurais até então proposto, o Perceptron. O perceptron era basicamente o neurônio proposto por McCulloch e Pitts, porém com uma regra de aprendizagem capaz de encontrar os pesos corretos. Esse processo se dá através da modificação dos pesos e do limiar do neurônio com base em cada entrada do neurônio até encontrar um conjunto capaz de resolver o problema que buscava-se solucionar ou até atingir uma regra de encerramento do processo de aprendizagem. Vale ressaltar que ainda é necessário definir, através de alguma heurística, os valores iniciais dos pesos. Desse modo, a regra de aprendizado pode ser descrita como:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t), \quad (2.4)$$

em que  $\Delta\mathbf{w}(t)$  é o ajuste necessário para atualizar os pesos de acordo com a entrada  $\mathbf{x}(t)$ . Esse ajuste é calculado com base no erro obtido entre a saída desejada  $d(t)$  e a saída da rede  $y(t)$ , ou seja,  $e(t) = d(t) - y(t)$ . Desse modo, o ajuste dos pesos a cada nova entrada  $\mathbf{x}(t)$  é dada por:

$$\Delta\mathbf{w}(t) = \eta\mathbf{x}(t)e(t), \quad (2.5)$$

em que  $\eta$  é uma variável, denominada fator ou taxa de aprendizagem, que regula o quanto  $\mathbf{w}(t+1)$  será ajustado em relação a entrada  $\mathbf{x}(t)$ . Assim, a regra de aprendizagem pode ser escrita como:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta\mathbf{x}(t)e(t). \quad (2.6)$$

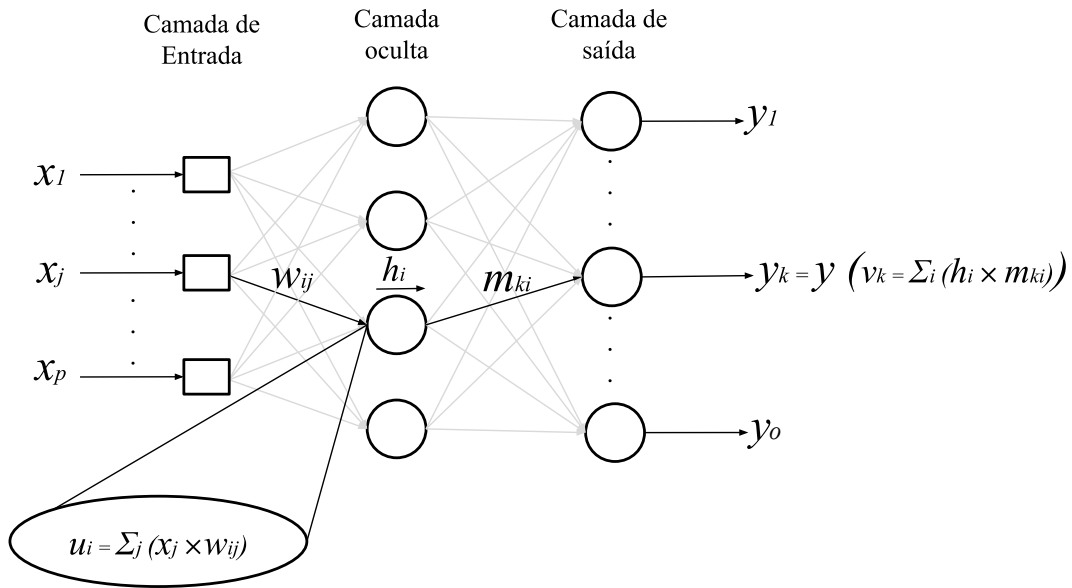
Embora o Perceptron tenha solucionado o problema da regra de aprendizagem, o que o permitiu ser utilizado em alguns problemas reais, ele era limitado a problemas lineares, ou seja, o perceptron tinha a capacidade de aprender apenas a aproximar funções de conjuntos de dados separáveis linearmente. Isso se dava pelo fato de que para solucionar problemas não lineares eram necessárias múltiplas camadas de neurônios, porém, a regra de aprendizagem de Rosenblatt não funcionava para mais de uma camada. Esse problema só foi solucionado em meados dos anos 1980, quando, pelo menos quatro grupos diferentes, reinventaram o algoritmo de aprendizado através da retropropagação do erro (RUSSELL; NORVIG, 2010).

### 2.3.3 Algoritmo de retropropagação do erro (*Backpropagation*)

Embora o algoritmo de retropropagação tenha sido reinventado por mais de um grupo de pesquisa, ele somente se tornou relevante com o trabalho de Rumelhart, Hinton e Williams (1986), o qual descreve várias redes neurais em que a retropropagação funciona de maneira muito mais eficiente do que as abordagens anteriores de aprendizagem, tornando possível o uso de redes neurais para resolver problemas que antes eram insolúveis.

O processo de treinamento de uma rede perceptron multicamadas se dá em dois sentidos: direto (*forward*), no qual o sinal vai da camada de entrada à camada de saída da rede; e inverso (*backward*), com o sinal saindo da camada de saída para as camadas intermediárias, chamadas de camadas ocultas. Para entender melhor, considere a representação da rede perceptron multicamadas da Figura 2.5.

Figura 2.5 – Rede perceptron multicamadas



Fonte: Próprio autor.

O sentido direto é semelhante ao visto no perceptron simples, de modo que a saída  $\mathbf{h}$  da camada oculta é a entrada da camada seguinte. Além disso, tomando como base a notação da Figura 2.5, é possível também calcular o erro para o  $k$ -ésimo neurônio da camada de saída na saída da rede por:

$$e_k(t) = d_k(t) - y_k(t). \quad (2.7)$$

Com isso, a regra de aprendizagem pode ser utilizada para atualizar os pesos dos neurônios

da camada de saída da seguinte forma:

$$m_{k,i}(t+1) = m_{k,i}(t) + \eta y'_k(t) e_k(t) h_i(t), \quad (2.8)$$

em que  $y'_k(t)$  é a derivada  $\frac{dy_k(t)}{dv_k(t)}$  da função de ativação e  $h_i$  é a saída do  $i$ -ésimo neurônio oculto. Do mesmo modo, a regra de aprendizagem é utilizada para atualizar os pesos da camada oculta:

$$w_{i,j}(t+1) = w_{i,j}(t) + \eta h'_i(t) \mathcal{E}_i(t) x_j(t), \quad (2.9)$$

em que  $h'_i(t)$  é a derivada  $\frac{dh_i(t)}{du_i(t)}$  da função de ativação do neurônio da camada oculta. Entretanto, não é possível atualizar os pesos  $\mathbf{w}$  da camada oculta a partir do erro de um único neurônio da camada de saída, visto que é necessário calcular a influência do neurônio oculto  $i$  sobre o erro de todos os neurônios de saída, representado por  $\mathcal{E}_i$ . Foi esse problema que o algoritmo de retropropagação do erro solucionou. Para tanto, o erro retropropagado para  $i$ -ésimo neurônio da camada oculta é obtido a partir do cálculo do gradiente de uma função de custo em relação ao vetor de pesos  $\mathbf{w}_i$ , sendo a função de custo do erro quadrático a mais frequentemente usada e definida por:

$$C = \frac{1}{2} \sum_{k=1}^o (d_k(t) - y_k(t))^2, \quad (2.10)$$

em que  $o$  é o número total de neurônios na camada de saída. De acordo com o gradiente descendente, cada mudança de peso na rede deve ser proporcional ao gradiente negativo do custo em relação ao peso específico que se está interessado em modificar. Logo:

$$\Delta w_{i,j} = -\eta \frac{\partial C}{\partial w_{i,j}}, \quad (2.11)$$

em que  $\eta$  é a taxa de aprendizagem.

Assim,

$$\frac{\partial C}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \frac{1}{2} \sum_{k=1}^o (d_k(t) - y_k(t))^2. \quad (2.12)$$

Isso pode ser denotado como a soma dos erros dos neurônios da camada de saída

$$\frac{\partial C}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \frac{1}{2} (e_1^2(t) + e_2^2(t) + \dots + e_k^2(t) + \dots + e_o^2(t)). \quad (2.13)$$

Logo,

$$\frac{\partial C}{\partial w_{i,j}} = \frac{1}{2} \left( \frac{\partial}{\partial w_{i,j}} e_1^2(t) + \frac{\partial}{\partial w_{i,j}} e_2^2(t) + \dots + \frac{\partial}{\partial w_{i,j}} e_k^2(t) + \dots + \frac{\partial}{\partial w_{i,j}} e_o^2(t) \right). \quad (2.14)$$

Ao analisar separadamente os termos do somatório da equação 2.14, ou seja

$$\frac{\partial e_k^2(t)}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} (d_k(t) - y_k(t))^2 = \frac{\partial}{\partial w_{i,j}} (d_k(t) - y_k(v_k(t)))^2, \quad (2.15)$$

em que  $v_k(t)$  representa as contribuições dadas pelos neurônios  $i$  conectados a  $j$ , como pode ser visto na Figura 2.5. Com isso, vemos que é possível calcular a derivada dessa expressão, resultando em

$$\frac{\partial e_k^2(t)}{\partial w_{i,j}} = -2e_k(t)y'_k \frac{\partial}{\partial w_{i,j}} v_k(t). \quad (2.16)$$

Como  $v_k(t) = \sum_{i=1}^q m_{k,i}(t)h_i(t)$ , logo, a derivada de

$$\frac{\partial}{\partial w_{i,j}} v_k(t) = \frac{\partial}{\partial w_{i,j}} \left( \sum_{i=1}^q m_{k,i}(t)h_i(t) \right). \quad (2.17)$$

Entretanto, somente o neurônio  $i$  da camada escondida tem o peso  $w_{i,j}$  como entrada. Assim, a derivada da equação 2.17 é reduzida a

$$\frac{\partial}{\partial w_{i,j}} v_k(t) = \frac{\partial}{\partial w_{i,j}} m_{k,i}(t)h_i(t) = m_{k,i}(t) \frac{\partial}{\partial w_{i,j}} h_i(t) \quad (2.18)$$

que pela regra da cadeia, obtém-se

$$\frac{\partial}{\partial w_{i,j}} v_k(t) = m_{k,i}(t)h'_i(t) \frac{\partial}{\partial w_{i,j}} u_i(t). \quad (2.19)$$

Conforme pode ser observado na Figura 2.5,  $u_i(t)$  equivale a soma ponderada das entradas conectadas no neurônio oculto  $i$ , logo, a partir da derivada  $\frac{\partial}{\partial w_{ij}} u_i(t)$  na equação 2.19, tem-se

$$\frac{\partial}{\partial w_{i,j}} v_k(t) = m_{k,i}(t)h'_i(t)x_j(t). \quad (2.20)$$

Desse modo, substituindo 2.20 em 2.16, obtém-se

$$\frac{\partial e_k^2(t)}{\partial w_{i,j}} = -2e_k(t)y'_k(t)m_{k,i}(t)h'_i(t)x_j(t). \quad (2.21)$$

Além disso, é possível reescrever a equação 2.14 da seguinte forma

$$\frac{\partial C}{\partial w_{i,j}} = \frac{1}{2} \left( \sum_{k=1}^o \frac{\partial}{\partial w_{i,j}} e_k^2(t) \right), \quad (2.22)$$

de modo que, considerando a equação 2.21, tem-se

$$\frac{\partial C}{\partial w_{i,j}} = \frac{1}{2} \left( \sum_{k=1}^o -2e_k(t)y'_k(t)m_{k,i}(t)h'_i(t)x_j(t) \right). \quad (2.23)$$

Como os termos  $h'_i(t)$  e  $x_j(t)$  não dependem de  $k$ , então

$$\frac{\partial C}{\partial w_{i,j}} = -h'_i(t)x_j(t) \left( \sum_{k=1}^o e_k(t)y'_k(t)m_{k,i}(t) \right). \quad (2.24)$$

Por fim, como visto na equação 2.11, o ajuste dos pesos é realizado no sentido contrário ao gradiente, dessa forma, tem-se que

$$\Delta w_{i,j} = \eta h'_i(t) \left( \sum_{k=1}^o e_k(t)y'_k(t)m_{k,i}(t) \right) x_j(t). \quad (2.25)$$

O procedimento para redes multicamadas com mais de uma camada oculta é o mesmo, repetindo o processo para as camadas mais internas. Além disso, vale ressaltar que as funções de ativações dos neurônios devem ser funções não lineares, visto que essas funções permitem que os nós aprendam estruturas mais complexas nos dados. Duas funções de ativação não linear amplamente utilizadas são as funções de ativação logística e tangente hiperbólica, denotadas, respectivamente, da seguinte maneira:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \quad (2.26)$$

$$\tanh(x) = \frac{1 - \exp(-x)}{1 + \exp(-x)}. \quad (2.27)$$

Além disso, quando deseja-se normalizar a saída de uma rede neural para uma distribuição de probabilidade sobre as classes, utiliza-se também uma outra função de ativação chamada **softmax**, que é, basicamente, uma generalização da função logística para múltiplas dimensões e é expressa por:

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (2.28)$$

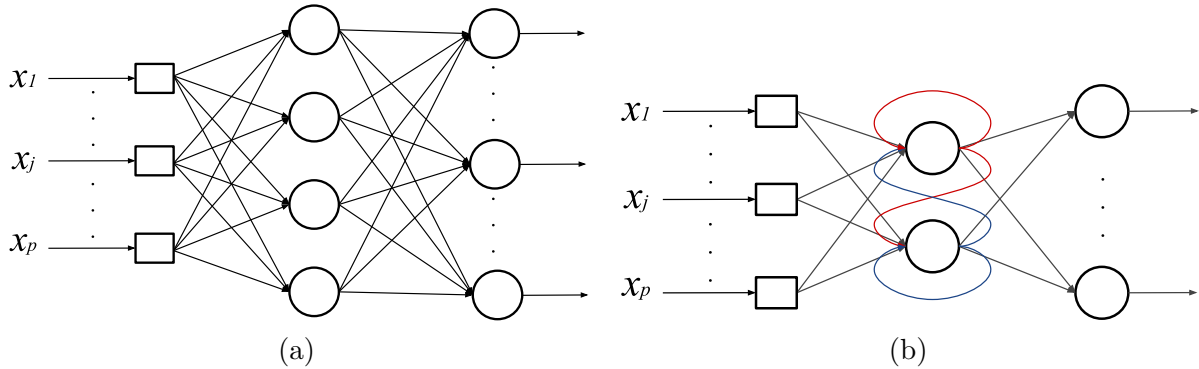
## 2.3.4 Redes Neurais Recorrentes (RNR)

### 2.3.4.1 Visão geral

Redes neurais recorrentes são um tipo de rede neural no qual a saída dos neurônios ocultos da etapa é ligada a entrada dos neurônios ocultos da etapa seguinte, de modo que as informações da etapa anterior ajudam a alimentar a etapa seguinte. Esse tipo de

arquitetura de redes neurais ajuda em problemas em que é necessário guardar alguma informação a cerca da entrada anterior, por isso é muito utilizada em problemas que envolve algum tipo de sequência. A Figura 2.6 ajuda a visualizar melhor a arquitetura das RNR comparada as redes neurais tradicionais, chamadas de redes *feed-forward*.

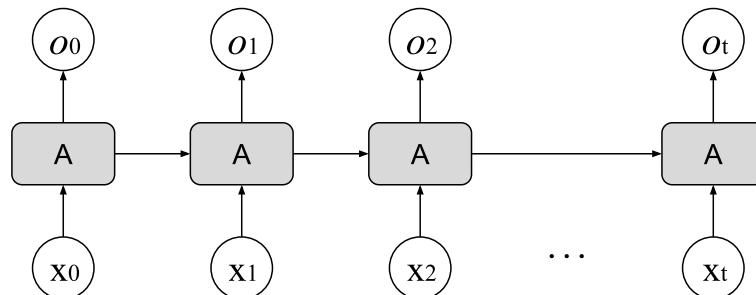
Figura 2.6 – (a) Rede neural *feed-forward*; (b) Rede neural recorrente.



Fonte: Próprio autor.

Ao observar a Figura 2.6b é possível notar que, de forma semelhante a rede *feed-forward*, cada neurônio da RNR recebe uma entrada  $x_t$  e produz um valor  $h_t$ , entretanto, um *loop* permite que algumas informações sejam passadas de uma etapa da rede para a próxima. Esta informação é o estado oculto, que é uma representação das entradas anteriores. Uma outra forma de desenhar o RNR é de forma semelhante a um grafo desdobrado em várias etapas de tempo, com cada componente da rede sendo representado por uma variável para cada intervalo de tempo, de modo a representar o estado do componente naquele momento. O tamanho desse grafo depende do comprimento da sequência de entrada, conforme ilustrado na Figura 2.7.

Figura 2.7 – RNR desdobrada

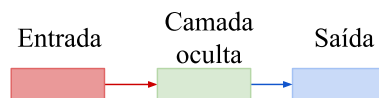


Fonte: Próprio autor.

### 2.3.4.2 Alguns exemplos de padrões de design importantes para RNR

Uma outra característica importante das RNR é que elas permitem ter sequências de comprimento variável com entradas e saídas, enquanto que as redes *feed-forward* funcionam apenas com tamanhos pré-determinados (recebem entradas de tamanho fixo e produzem tamanho fixo) - Figura 2.8. Com isso, é possível projetar uma ampla variedade de redes neurais recorrentes, tal como pode ser observado nas Figuras 2.9, 2.10, 2.11 e 2.12, nas quais cada retângulo é um vetor e as setas representam funções de ativação.

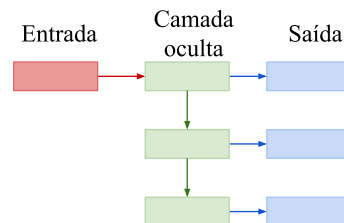
Figura 2.8 – Rede *feed-forward* - Um para um



Fonte: Próprio autor.

Na Figura 2.9 é possível observar uma RNR que recebe como entrada um vetor e tem como saída uma sequência de vetores. Este tipo de rede é útil para gerar descrições automáticas, como por exemplo, gerar uma descrição para uma imagem.

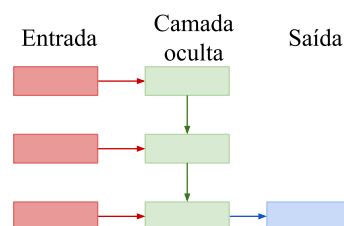
Figura 2.9 – RNR - Um para muitos



Fonte: Próprio autor.

Já na Figura 2.10, é mostrada uma rede que recebe uma sequência de vetores e gera um único vetor de saída. Análise de sentimento em frases é um dos problemas que redes com essa arquitetura conseguem resolver, pois, a partir da frase dada como entrada para a rede, é gerado um vetor que representa o sentimento que melhor representa a frase.

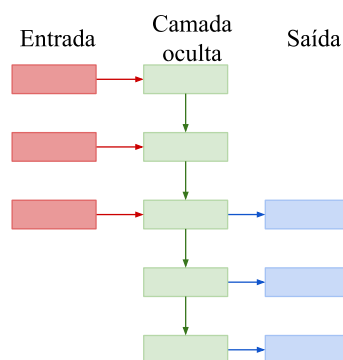
Figura 2.10 – RNR - Muitos para um



Fonte: Próprio autor.

A RNN exposta na Figura 2.11 é capaz de receber uma sequência de vetores e gerar uma outra sequência de vetores que não necessariamente tenha o mesmo tamanho. Essa arquitetura se mostrou muito eficaz em algoritmos de tradução automática, como o presente no Google Tradutor (WU et al., 2016).

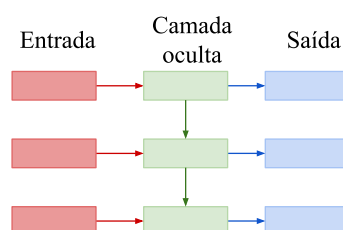
Figura 2.11 – RNR - Muitos para muitos



Fonte: Próprio autor.

Por último, tem-se na Figura 2.12 uma RNN que também recebe uma sequência de vetores e gera uma sequência de vetores como saída, entretanto, nesse caso, cada vetor de saída refere-se ao vetor de entrada. Esse tipo de rede é útil quando se deseja rotular cada *frame* de um vídeo, por exemplo.

Figura 2.12 – RNR - Muitos para muitos

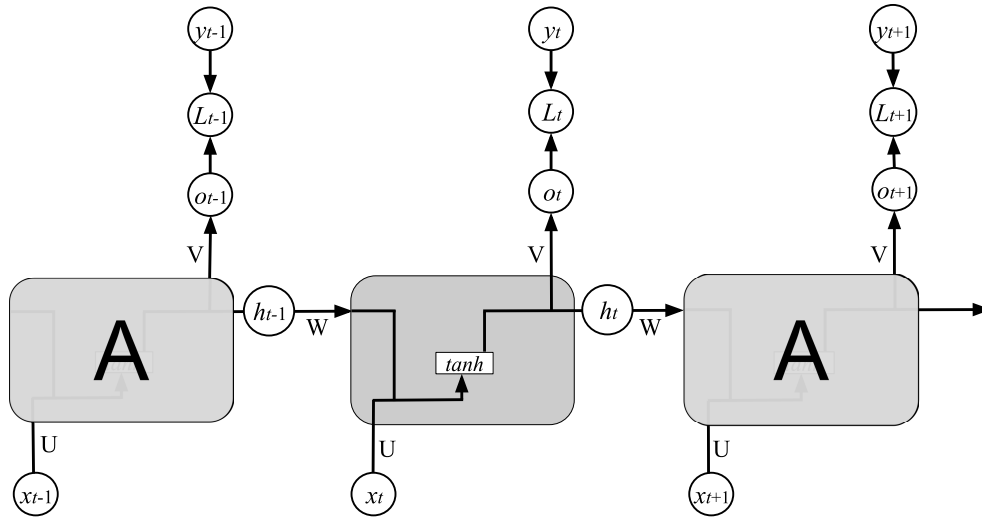


Fonte: Próprio autor.

### 2.3.4.3 Cálculo do gradiente

De forma semelhante as redes *feed-forward*, a atualização dos pesos da RNR se dá através do cálculo do gradiente para as perdas  $L$  em duas etapas, a direta (*forward*) e a reversa (*backward*). O algoritmo de retropropagação aplicado as redes RNR é uma adaptação do algoritmo utilizado nas redes *feed-forward*, pois é necessário executá-lo sobre cada etapa descrita na Figura 2.7. Esse algoritmo modificado é chamado de Retropropagação do Erro Através do Tempo (*Backpropagation Through Time (BPTT)*). Para melhor compreender o mecanismo de propagação do sinal através dos componentes da RNR, considere a seguinte versão expandida da Figura 2.7.

Figura 2.13 – RNR expandida



Fonte: Próprio autor.

O grafo da Figura 2.13 representa uma rede recorrente que mapeia uma sequência de entrada de valores  $\mathbf{x}$  para uma sequência correspondente de valores de saída  $\mathbf{o}$ . A distância do erro da saída  $\mathbf{o}$  para cada alvo de treinamento correspondente em  $\mathbf{y}$  é medida pela perda  $L$ . No caso geral, assume-se que os valores de  $\mathbf{o}$  são não normalizados, devido a isso, normalmente é utilizado a função de ativação *softmax* sobre  $\mathbf{o}$  para gerar probabilidades normalizadas e então poder compara-las com o alvo  $y$ , respectivamente. Além disso, as conexões da rede são parametrizadas por três matrizes de pesos:  $\mathbf{U}$ ,  $\mathbf{W}$  e  $\mathbf{V}$ . A matriz  $\mathbf{U}$  parametriza as entradas  $\mathbf{x}$ , a matriz  $\mathbf{W}$  parametriza as conexões ocultas e as saídas são parametrizadas por  $\mathbf{V}$ . Note também que a função de ativação da camada oculta é a tangente hiperbólica.

Agora, com esses componentes bem definidos, é possível desenvolver as equações de propagação direta *forward* para para essa RNR. Dessa forma, para cada etapa no tempo  $t$  de 1 a  $p$ , são aplicadas as seguintes equações.

$$\begin{aligned} \mathbf{h}_t &= \tanh(\mathbf{b} + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t) \\ \mathbf{o}_t &= \mathbf{c} + \mathbf{V}\mathbf{h}_t \\ \hat{\mathbf{y}} &= \text{softmax}(\mathbf{o}_t) \end{aligned} \tag{2.29}$$

em que  $\mathbf{h}_t$  é o vetor de pesos da camada oculta da rede no tempo  $t$  e  $\mathbf{b}$  e  $\mathbf{c}$  são os vetores *bias*. Este é um exemplo de uma rede recorrente que mapeia uma sequência de entrada para uma sequência de saída do mesmo comprimento equivalente a rede vista na Figura 2.12.

Note que para as redes de cada etapa (representadas como nós  $\mathbf{N}$  de um grafo) precisamos calcular o gradiente  $\nabla_{\mathbf{N}}L$  recursivamente com base no gradiente calculado nos nós predecessores. Assim, a recursão começa com os nós imediatamente anteriores à perda final:

$$\frac{\partial L}{\partial L_t} = 1, \quad (2.30)$$

em que assume-se que as saídas  $o_t$  são processadas pela função *softmax* para obter o vetor  $\hat{y}$  de probabilidades sobre a saída. Também assume-se que a perda é a *log-likelihood* negativa do verdadeiro  $y_t$ , dada a entrada. Dessa forma,  $\nabla_{\mathbf{o}_t}L$  nas saídas da etapa de tempo  $t$  para cada nó  $i$  é:

$$(\nabla_{\mathbf{o}_t}L)_i = \frac{\partial L}{\partial o_{i,t}} = \frac{\partial L}{\partial L_t} \frac{\partial L_t}{\partial o_{i,t}} = \hat{y}_{i,t} - \mathbf{1}_{i=y_t}. \quad (2.31)$$

Já na etapa reversa (*backward*) inicia-se pelo fim da sequência. No passo final  $p$ ,  $\mathbf{h}_p$  possui apenas  $\mathbf{o}_p$  como descendente, o que simplifica seu gradiente:

$$\nabla_{\mathbf{h}_p}L = \mathbf{V}^T \nabla_{\mathbf{o}_p}L. \quad (2.32)$$

Assim, é possível iterar para trás no tempo para retropropagar os gradientes ao longo do tempo, de  $t = p - 1$  até  $t = 1$ , lembrando que, para todo  $t < p$ ,  $\mathbf{h}_t$  tem como descendentes  $\mathbf{o}_t$  e  $\mathbf{h}_{t+1}$ . Portanto, seu gradiente é dado por

$$\begin{aligned} \nabla_{\mathbf{h}_t}L &= \left( \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} \right)^T (\nabla_{\mathbf{h}_{t+1}}L) + \left( \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \right)^T (\nabla_{\mathbf{o}_t}L) \\ &= \mathbf{W}^T \text{diag}(1 - (\mathbf{h}_{t+1})^2)^T (\nabla_{\mathbf{h}_{t+1}}L) + \mathbf{V}^T (\nabla_{\mathbf{o}_t}L), \end{aligned} \quad (2.33)$$

em que  $\text{diag}(1 - (\mathbf{h}_{t+1})^2)^T$  indica a matriz diagonal contendo os elementos  $1 - (\mathbf{h}_{it+1})^2$ , que é a matriz Jacobiana da tangente hiperbólica associada à unidade oculta  $i$  no tempo  $t + 1$

Uma vez obtidos os gradientes nos nós internos do grafo computacional, é possível obter os gradientes nos nós parâmetros. Como os parâmetros são compartilhados em muitas etapas de tempo, devemos calcular a contribuição de cada aresta do grafo para o gradiente individualmente. Assim, o gradiente dos parâmetros restantes é dado por

$$\nabla_{\mathbf{c}}L = \sum_t \left( \frac{\partial \mathbf{o}_t}{\partial \mathbf{c}} \right)^T \nabla_{\mathbf{o}_t}L = \sum_t \nabla_{\mathbf{o}_t}L, \quad (2.34)$$

$$\nabla_{\mathbf{b}} L = \sum_t \left( \frac{\partial \mathbf{h}_t}{\partial \mathbf{b}} \right)^\top \nabla_{\mathbf{h}_t} L = \sum_t \text{diag} \left( 1 - (\mathbf{h}_t)^2 \right) (\nabla_{\mathbf{h}_t} L), \quad (2.35)$$

$$\nabla_{\mathbf{v}} L = \sum_t \sum_i \left( \frac{\partial L}{\partial o_{i,t}} \right) \nabla_{\mathbf{v}_t o_{i,t}} = \sum_t (\nabla_{\mathbf{o}_t} L) \mathbf{h}_t^\top, \quad (2.36)$$

$$\begin{aligned} \nabla_{\mathbf{w}} L &= \sum_t \sum_i \left( \frac{\partial L}{\partial h_{i,t}} \right) \nabla_{\mathbf{w}_t} h_{i,t} = \\ &= \sum_t \text{diag} \left( 1 - (\mathbf{h}_t)^2 \right) (\nabla_{\mathbf{h}_t} L) \mathbf{h}_{t-1}^\top, \end{aligned} \quad (2.37)$$

$$\begin{aligned} \nabla_{\mathbf{u}} L &= \sum_t \sum_i \left( \frac{\partial L}{\partial h_{i,t}} \right) \nabla_{\mathbf{u}_t} h_{i,t} = \\ &= \sum_t \text{diag} \left( 1 - (\mathbf{h}_t)^2 \right) (\nabla_{\mathbf{h}_t} L) \mathbf{x}_t^\top. \end{aligned} \quad (2.38)$$

Não é preciso calcular o gradiente em relação a  $\mathbf{x}_t$  para o treinamento porque ele não tem nenhum parâmetro como ancestral no grafo, como pode ser observado na Figura 2.13.

### 2.3.5 Long Short-Term Memory (LSTM)

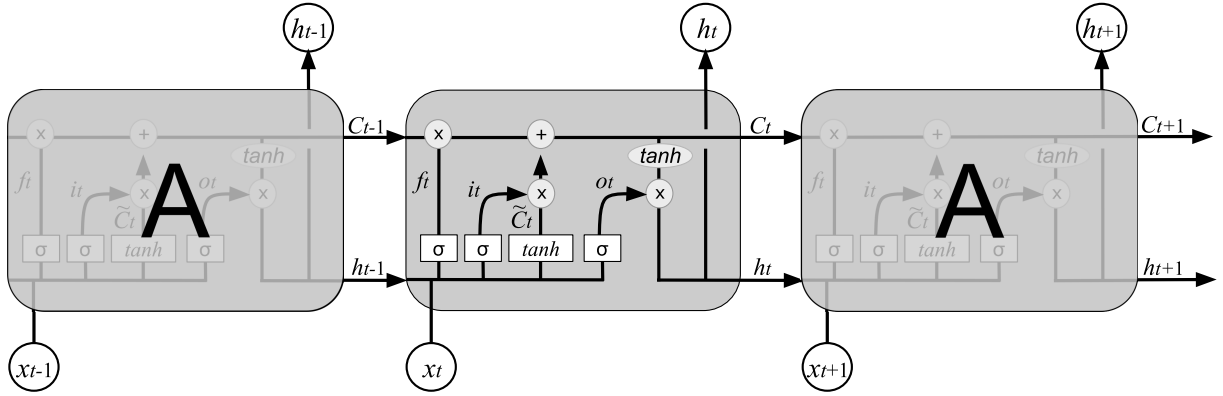
No início da década de 90, foi descoberto paralelo e independentemente pelos pesquisadores Hochreiter (1991), Bengio, Simard e Frasconi (1994) um problema que trazia uma grande limitação para as redes neurais recorrentes como as descritas na seção anterior: o problema do desaparecimento do gradiente. Basicamente, o problema é que quando se propaga o gradiente por vários estágios, eles tendem a desaparecer na maioria das vezes, e algumas poucas vezes eles tendem a explodir.

Além disso, mesmo que fosse possível criar uma rede com parâmetros estáveis, de modo a armazenar os gradiente sem que eles explodam, seria difícil aprender dependências de longo prazo, pois o gradiente de uma interação de longo prazo tem magnitude exponencialmente menor do que o gradiente de uma interação de curto prazo, visto que o sinal sobre essas dependências tenderá a ser escondido pelas menores flutuações decorrentes das dependências de curto prazo (GOODFELLOW; BENGIO; COURVILLE, 2016).

A rede LSTM foi propostas com uma arquitetura capaz de resolver o problema do desaparecimento e da explosão do gradiente. A ideia por trás do LSTM inicial (HOCHREITER; SCHMIDHUBER, 1997) são os *auto-loops* para produzir caminhos onde o

gradiente pode fluir por longos períodos. Uma importante evolução no LSTM foi fazer com que o peso desses *auto-loops* sejam condicionados ao contexto, em vez de fixos (GERS; SCHMIDHUBER; CUMMINS, 2000). Desse modo, em vez de ter uma única unidade que simplesmente aplica uma não linearidade, como é nas RNR comuns, as redes recorrentes LSTM têm “células LSTM” com uma recorrência interna (um *auto-loop*) chamada estado da célula, além da recorrência externa do RNR. Cada célula possui as mesmas entradas e saídas de uma rede recorrente comum, porém com mais parâmetros e um sistema com três unidades multiplicativas, geralmente chamadas de *gates*. Para melhor compreender o funcionamento de uma célula LSTM, considere a Figura 2.14.

Figura 2.14 – Célula LSTM



Fonte: Próprio autor.

A chave para os LSTMs é a linha horizontal que atravessa a parte superior do diagrama, essa linha representa o estado da célula. O estado da célula pode ser pensando como um transportador de informações que percorre todas as células da rede, acumulando informações por um longo período de tempo. O LSTM tem a capacidade de remover ou adicionar informações ao estado da célula, através dos *gates*, que como o nome sugere, agem como portões que regulam a passagem dessas informações através de não linearidades sigmóides.

Observando a Figura 2.14, têm-se que a primeira etapa da célula LSTM é decidir quais informações serão “esquecidas” do estado da célula. Esta decisão é tomada pelo portão de esquecimento (*forget gate*)  $f_j^{(t)}$  (para o tempo  $t$  e célula  $j$ ) que, através de uma não linearidade sigmóide sobre  $\mathbf{h}^{(t-1)}$  e  $\mathbf{x}^{(t)}$ , produz um número entre 0 e 1 para cada valor no estado da célula  $C^{(t-1)}$ . Um 1 representa “manter”, enquanto um 0 representa “esquecer”.

$$f_j^{(t)} = \sigma \left( b_j^f + \sum_k \mathbf{U}_{j,k}^f x_k^{(t)} + \sum_k \mathbf{W}_{j,k}^f h_k^{(t-1)} \right), \quad (2.39)$$

em que  $\mathbf{x}^{(t)}$  é o vetor de entrada atual e  $\mathbf{h}^{(t)}$  é o vetor da camada oculta atual, contendo as saídas de todas as células LSTM, e  $\mathbf{b}^f$ ,  $\mathbf{U}^f$  e  $\mathbf{W}^f$  são, respectivamente, vieses, pesos de entrada e pesos recorrentes para o portão de esquecimento  $f_j^{(t)}$ .

A próxima etapa é decidir quais novas informações serão armazenada no estado da célula. Esse processo é dividido em duas partes. Primeiro a unidade sigmóide do portão de entrada (*input gate*)  $i^{(t)}$  decide quais valores serão atualizados de forma semelhante ao portão de esquecimento. Em seguida, uma unidade *tanh* cria um vetor de novos valores candidatos,  $\tilde{C}^{(t)}$ , que podem ser adicionados ao estado. Desse modo, têm-se

$$i_j^{(t)} = \sigma \left( b_j^i + \sum_k \mathbf{U}_{j,k}^i x_k^{(t)} + \sum_k \mathbf{W}_{j,k}^i h_k^{(t-1)} \right), \quad (2.40)$$

$$\tilde{C}_j^{(t)} = \tanh \left( b_j^c + \sum_k \mathbf{U}_{j,k}^c x_k^{(t)} + \sum_k \mathbf{W}_{j,k}^c h_k^{(t-1)} \right), \quad (2.41)$$

em que  $(\mathbf{b}^i, \mathbf{U}^i, \mathbf{W}^i)$  e  $(\mathbf{b}^c, \mathbf{U}^c, \mathbf{W}^c)$  são, respectivamente, vieses, pesos de entrada e pesos recorrentes para o portão de entrada  $i^{(t)}$  e para unidade  $\tilde{C}_t$ .

A atualização do estado da célula antiga,  $C^{(t-1)}$ , para o novo estado da célula  $C^{(t)}$  é atualizado da seguinte forma:

$$C_j^{(t)} = f_j^{(t)} C_j^{(t-1)} + i_j^{(t)} \tilde{C}_j^{(t)}, \quad (2.42)$$

em que, ao multiplicar-se o antigo estado por  $f^{(t)}$ , são removidos os valores que devem ser esquecidos. Em seguida, adiciona-se os novos valores candidatos  $i^{(t)}$ .

Por fim, é necessário computar os valores da saída  $\mathbf{h}^{(t)}$  por meio do portão de saída  $o^{(t)}$ , que também usa uma unidade sigmóide.

$$o_j^{(t)} = \sigma \left( b_j^o + \sum_k \mathbf{U}_{j,k}^o x_k^{(t)} + \sum_k \mathbf{W}_{j,k}^o h_k^{(t-1)} \right), \quad (2.43)$$

$$h_j^{(t)} = o_j^{(t)} \tanh(C_j^{(t)}), \quad (2.44)$$

em que  $\mathbf{b}^o$ ,  $\mathbf{U}^o$  e  $\mathbf{W}^o$  são, respectivamente, vieses, pesos de entrada e pesos recorrentes para o portão de saída  $o^{(t)}$ .

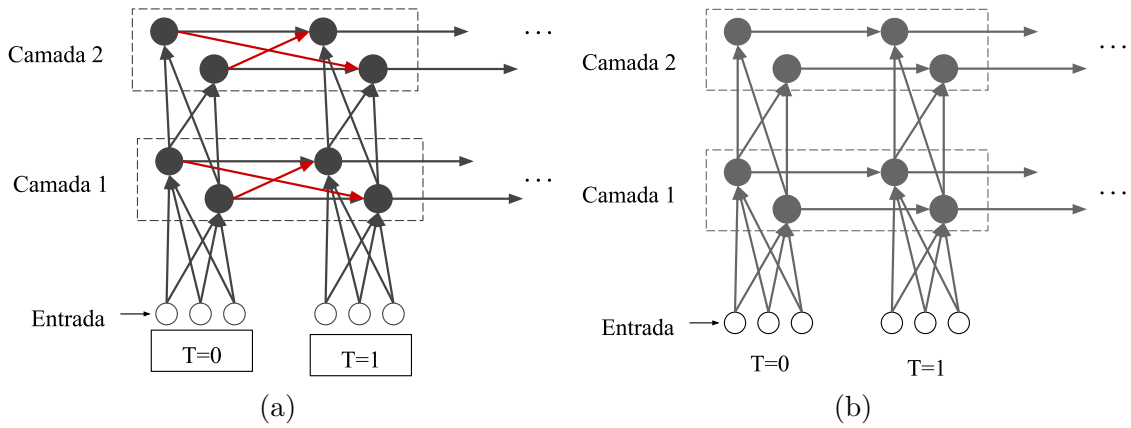
### 2.3.6 Independently Recurrent Neural Network (IndRNN)

De acordo com (Li et al., 2018), a rede LSTM, por usar funções de ativação tangente hiperbólica e sigmóides, possuem um decaimento do gradiente sobre as suas camadas. Com o objetivo de solucionar tais problemas, os autores propuseram uma rede neural recorrente

na qual os neurônios da mesma camada são independentes uns dos outros. Dessa forma, diferente da LSTM que trás um aumento na complexidade em relação as RNR, a rede IndRNN simplificou as conexões internas da RNR padrão. (Li et al., 2018) mostrou como isso possibilitou a IndRNN de ser facilmente regulada para evitar os problemas relacionados ao desaparecimento e explosão de gradiente, permitindo também a rede IndRNN aprender longas dependências.

Além disso, uma IndRNN pode trabalhar com funções de ativação não saturadas, tais como ReLU (*rectified linear unit*), e ainda ser treinada de forma robusta. Na Figura 2.15 é possível notar a diferença entre uma RNR padrão e a IndRNN.

Figura 2.15 – (a) RNR padrão; (b) IndRNN.



Fonte: LI et al., 2019.

Dessa forma, a rede IndRNN é descrita da seguinte maneira:

$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{x}_t + \mathbf{w} \odot \mathbf{h}_{t-1} + \mathbf{b}), \quad (2.45)$$

em que  $\mathbf{w}$  é o vetor de pesos recorrente e  $\odot$  representa o produto Hadamard. Cada neurônio em uma camada é independente dos demais e as conexões entre os neurônios podem ser alcançado pelo empilhamento de duas ou mais camadas de IndRNNs. Em resumo, a diferença da rede IndRNN para a RNR padrão é que, enquanto a RNR padrão utiliza uma matriz  $\mathbf{W}$  de pesos, a IndRNN utiliza um vetor de  $\mathbf{w}$  de pesos recorrentes. Dessa forma, para o  $i$ -ésimo neurônio, o estado oculto  $h_{i,t}$  pode ser obtido como

$$h_{i,t} = \sigma(\mathbf{u}_i\mathbf{x}_t + w_i h_{i,t-1} + b_i), \quad (2.46)$$

em que  $\mathbf{u}_i$  e  $w_i$  são, respectivamente, a  $i$ -ésima linha da matriz de pesos de entrada e o peso recorrente desse neurônio. Cada neurônio apenas recebe informação da entrada e de

seu próprio estado oculto nos passos anteriores. Ou seja, cada neurônio em um IndRNN lida com um tipo de padrão espaço-temporal de forma independente.

Por utilizar um vetor, a complexidade do treinamento da IndRNN acabando sendo reduzido, o que se reflete no custo computacional. Além disso, o cálculo do gradiente envolve apenas o termo exponencial de um valor escalar  $w_n$ , que pode ser facilmente regulado, o que permite limitar também o gradiente da função de ativação a um determinado intervalo, sendo essa a estratégia utilizada pela rede para evitar desaparecimento e explosão do gradiente.

## 2.4 Conclusão

Neste capítulo foi apresentado a fundamentação teórica e os algoritmos utilizados nos experimentos desta dissertação, sendo eles: Cadeias de Markov de Primeira Ordem, Cadeias de Markov de Segunda ordem, RNR padrão, LSTM e IndRNN.

Estes algoritmos foram escolhidos por se adaptarem bem ao problema de predição de sequências temporais no qual o problema de recomendação de próxima funcionalidade está inserido, por isso, o conhecimento sobre funcionamento desses algoritmos é de grande importância para compreensão dos resultados obtidos.

## 3 Sistemas de Recomendação

Sistemas de recomendação são ferramentas e técnicas de software que fornecem sugestões de itens que devem ser úteis ao usuário. As sugestões fornecidas visam apoiar seus usuários em vários processos de tomada de decisão, tais como, quais itens comprar, que música ouvir, qual notícia ler e etc. Com isso, os sistemas de recomendação provaram ser meios valiosos para os usuários on-line lidarem com a sobrecarga de informações e se tornaram uma das ferramentas mais poderosas e populares do comércio eletrônico. Como resultado, várias técnicas para geração de recomendações foram propostas nas últimas duas décadas, sendo muitas delas implantadas com sucesso em ambientes comerciais (RICCI; ROKACH; SHAPIRA, 2011a).

Este capítulo apresenta uma visão geral sobre sistemas de recomendação, apresentando desde o início deste campo de pesquisa, os principais trabalhos e técnicas até os diferentes tipos de sistemas de recomendação e os problemas por eles abordados.

### 3.1 Uma breve história

Apesar de a ideia de usar um computador para filtrar um fluxo de informações recebidas de acordo com as preferências de um usuário remontar à década de 1960 (HENSLEY, 1963), sistemas de recomendação somente tornaram-se uma área de pesquisa independente após o surgimento dos primeiros trabalhos de pesquisa sobre filtragem colaborativa no início da década 1990, com destaque para os trabalhos de Goldberg et al. (1992), que deu origem ao termo filtragem colaborativa, e Resnick et al. (1994), que deu continuidade às ideias de Goldberg, mas introduzindo uma componente que automatizava as predições sobre quais itens recomendar baseado no esquema de vizinhos mais próximos.

Alguns trabalhos anteriores mais recentes da época já exploravam algumas técnicas hoje associadas à sistemas de recomendação, tais como a filtragem baseada em conteúdo (DEERWESTER et al., 1990; FOLTZ; DUMAIS, 1992; SALTON; BUCKLEY, 1988), mas que até então tinham como objetivo apenas a filtragem de itens, não a recomendação de itens ao usuário. Em geral, as aplicações desses trabalhos consistiam em recuperar documentos de texto que contivessem em seu conteúdo palavras-chaves relevantes disponibilizadas pelo usuário, daí a origem do termo. Diferente da filtragem baseada em conteúdo, as filtragens colaborativas de Goldberg et al. (1992), Resnick et al. (1994) utilizavam-se das avaliações dos usuários sobre os documentos para recomendar os que fossem mais relevantes a outros usuários. Segundo Resnick, a ideia por trás da filtragem colaborativa é simples: as pessoas que concordaram em suas avaliações subjetivas de artigos no passado provavelmente concordarão novamente no futuro.

O termo “sistema de recomendação” surgiu com Resnick e Varian (1997), que defendiam esse termo como sendo uma terminologia mais genérica do que filtragem colaborativa, já que sistemas de recomendação podem existir sem nenhuma colaboração entre as pessoas. Além disso, a área de sistemas de recomendação passou a englobar não somente filtragem colaborativa, mas também outras técnicas como a filtragem baseada em conteúdo (PARK et al., 2012). Poucos anos depois, sistemas de recomendação já estavam presentes em várias aplicações comerciais, como por exemplo no comércio eletrônico, sendo a Amazon.com um dos primeiros a adotar sistemas de recomendação para a recomendação de itens como livros, filmes, músicas e etc (SCHAFFER; KONSTAN; RIEDL, 1999).

De acordo com Ricci, Rokach e Shapira (2011a), o interesse em sistemas de recomendação cresceu dramaticamente nos últimos anos devido principalmente aos seguintes fatos:

- Sites grandes como Amazon.com, YouTube, Netflix, Yahoo, Tripadvisor e IMDb têm sistemas de recomendação como uma importante parte de suas plataformas de avaliações.
- Diversas conferências e workshops dedicados a sistemas de recomendação, em especial ACM Recommender Systems (RecSys), estabelecida em 2007 e sendo agora o principal evento anual em pesquisa e aplicações de tecnologia.
- A oferta de cursos de graduação e pós-graduação agora são dedicados inteiramente a sistemas de recomendação em instituições de ensino superior em todo o mundo; tutoriais sobre sistemas de recomendação são muito populares em conferências de ciência da computação.
- Edições especiais em revistas acadêmicas que cobrem pesquisas e desenvolvimentos no campo de sistemas de recomendações.

## 3.2 Classificação dos Sistemas de Recomendações

Dado a usabilidade de sistemas de recomendação nos mais variados problemas, diversos tipos de sistemas de recomendação surgiram com o tempo, variando em termos do domínio utilizado, do conhecimento usado, mas principalmente em relação ao algoritmo de recomendação. Além disso, outras diferenças estão relacionadas à forma como as recomendações são finalmente montadas e apresentadas ao usuário em resposta às suas solicitações (RICCI; ROKACH; SHAPIRA, 2011a). Com isso, diversos trabalhos que apresentam uma forma de classificar os diferentes tipos de sistemas de recomendação podem ser encontrado na literatura (BURKE, 2002; BURKE, 2007; PARK et al., 2012; BOBADILLA et al., 2013; AGGARWAL, 2016b; SOHAIL; SIDDIQUI; ALI, 2017).

Neste trabalho optou-se por se citar uma mescla das definições dos modelos básicos de sistemas de recomendação definidos por Burke (2007), com uma visão mais geral dos

tipos de sistemas de recomendação, servindo como base para distinguir melhor as diferentes abordagens utilizadas; sistemas de recomendação sensíveis a certos domínios, descritos por Aggarwal (2016b), que oferece algumas definições complementares aos modelos básicos, apresentando sistemas de recomendação que se baseiam nos domínios dos problemas, levando em conta informações de contexto nos quais os itens a serem recomendados estão inseridos; e por fim, uma visão mais aprofundada em sistemas de recomendação sensíveis à sequências, apresentado por Quadrana, Cremonesi e Jannach (2018), sendo tais tipo de sistemas de recomendação mais próximos ao problema de recomendação de próxima funcionalidade abordado neste trabalho.

### 3.2.1 Modelos Básicos de Sistemas de Recomendação

Segundo Burke (2007), as técnicas de recomendação podem ser distinguidas com base em suas fontes de conhecimento, ou seja, a origem do conhecimento necessário para fazer recomendações. Em alguns sistemas, esse conhecimento é o conhecimento das preferências dos usuários, enquanto que em outros é o conhecimento teórico ou inferencial sobre o domínio do problema, que pode ser adicionado por um especialista. Além disso, a forma com a qual o pode se obter a avaliação do usuário sobre determinado item pode ser através de **feedback explícito**: no qual o usuário avalia diretamente os itens a partir de notas, variando dentro de uma escala (normalmente de 1-5 ou 1-10), ou como positivo (gostei) e negativo (não gostei); ou através de **feedback implícito**: que corresponde as ações do cliente como comprar itens ou clicar em páginas da Web, utilizando a sequência histórica de ações do usuário para fazer previsões sobre ações futuras.

As subseções a seguir trazem uma visão geral sobre as diferentes classes dos modelos básicos de sistemas de recomendação.

#### 3.2.1.1 Filtragem colaborativa

A implementação mais simples e original dessa abordagem recomenda ao usuário ativo os itens que outros usuários com gostos semelhantes gostaram no passado. A similaridade no gosto de dois usuários é calculada com base na similaridade no histórico de classificação dos usuários. A filtragem colaborativa é considerada a técnica mais popular e amplamente implementada em Sistemas de Recomendação (RICCI; ROKACH; SHAPIRA, 2011a).

Como um exemplo da filtragem colaborativa, pode-se pensar em um catálogo de filmes de uma plataforma de *streaming* no qual os usuários podem avaliar os filmes assistidos. Baseado nessas avaliações, é possível dizer que os usuários que avaliaram uma lista de filmes de maneira similar possuem preferências de filmes semelhantes. Com isso, os filmes que foram bem avaliados por um usuário podem ser recomendados a usuários com preferências parecidas e que ainda não assistiram a esses filmes. O *feedback* explícito é o

mais comum nesse contexto, comumente utilizando uma nota de 1-5 (estrelas) ou apenas com positivo (gostei) e negativo (não gostei) para avaliar os filmes.

Devido à sua simplicidade, eficiência e sua capacidade de produzir recomendações precisas e personalizadas, os métodos de vizinhos mais próximos desfrutam de considerável popularidade em sistemas de recomendação baseado em filtragem colaborativa. Esses algoritmos são baseados nas seguintes observações: (i) usuários semelhantes exibem padrões semelhantes de comportamento de classificação (filtragem colaborativa baseada no usuário); (ii) itens semelhantes recebem classificações semelhantes (filtragem colaborativa baseada em itens).

No exemplo do catálogo de filmes citado anteriormente, pode-se classificar a recomendação de filmes bem avaliados por um usuário a outros usuários semelhantes como filtragem colaborativa baseada no usuário. O mesmo raciocínio foi usado em Shardanand e Maes (1995) para recomendar músicas para usuários com gostos semelhantes, baseando-se nas avaliações dos mesmos sobre um conjunto de músicas. Resnick et al. (1994) também se utilizavam dos algoritmos de vizinhos mais próximos com filtragem colaborativa baseada no usuário para recomendar artigos para usuários semelhantes. Já Liu e Lee (2010) utilizam uma versão adaptada dos vizinhos mais próximos para recomendar amigos em redes sociais utilizando informações dos usuários nas redes sociais para sugerir as amizades.

Já a filtragem colaborativa baseada em itens é mais comum quando se deseja recomendar itens levando em consideração como os itens semelhantes a este foram avaliados pelo usuário. Esse raciocínio pode novamente se estender ao exemplo do catálogo de filmes, recomendando-se filmes semelhantes aos que foram bem avaliados pelo usuário. Esse método teve grande destaque em sistemas de comércio eletrônico, no qual produtos semelhantes aos bem avaliados pelo usuário lhes são recomendados. Em Sarwar et al. (2002) é feita uma análise do desempenho das predições feitas utilizando filtragem colaborativa baseada em itens comparada à predições feitas utilizando a filtragem baseada em usuários em um sistema de comércio eletrônico, chegando a conclusão que há um aumento na qualidade das predições e da performance no sistema baseado em itens em relação ao baseado em usuário.

O principal desafio ao projetar métodos de filtragem colaborativa é que as matrizes *item-usuário* que guardam as avaliações são esparsas, já que não é comum que todos os usuários avalie todos os itens (AGGARWAL, 2016b). Uma solução que tem ganhado bastante destaque nos últimos anos para o problema de esparsidade é a utilização de técnicas de **fatoração de matrizes** baseada no modelo de fator latente (*latent factor models*). Diferente das técnicas de vizinhos mais próximos, os modelos de fator latente são abordagens que tentam explicar as avaliações caracterizando itens e usuários em um número de fatores (normalmente entre 20 a 100 fatores) inferidos das avaliações dos itens pelos usuários. Dessa forma, modelos de fatoração de matrizes mapeiam usuários e itens para um

espaço de fator latente comum de mesma dimensionalidade, ou seja, dada uma matriz de usuários-item de dimensionalidade  $m \times n$ , a fatoração de matriz decompõe essa matriz em duas matrizes  $m \times k$  e  $k \times n$ , isso permite que as interações usuário-item sejam modeladas como produtos internos nesse espaço. Assim, quando há uma alta correspondência dos fatores do item e do usuário, então é possível gerar uma recomendação (KOREN; BELL; VOLINSKY, 2009). Uma vantagem das técnicas de fatoração de matrizes é que possível trabalhar com informações inferidas não diretamente, como *feedback* implícito.

### 3.2.1.2 Filtragem baseada em conteúdo

Nos sistemas de recomendação baseados em conteúdo, os atributos descritivos dos itens são usados para fazer recomendações. O termo “conteúdo” refere-se a essas descrições. A similaridade dos itens é calculada com base nos atributos descritivos dos itens comparados. Desse modo, sistemas baseados em conteúdo aprendem a recomendar itens baseados nas avaliações do usuários combinados com os atributos descritivos do conteúdo dos itens. Por exemplo, se um usuário avaliou positivamente um filme classificado como comédia e não se tem acesso às avaliações dos outros usuários, o sistema pode recomendar outros filmes desse gênero que contém palavras chaves em sua descrição semelhantes as da descrição do primeiro filme.

### 3.2.1.3 Sistemas de recomendação baseados no conhecimento

Os sistemas baseados no conhecimento recomendam itens com base no conhecimento de como certas característica do item atendem às necessidades e preferências dos usuários, bem como o quão útil é o item para esse usuário. Ou seja, o processo de recomendação é realizado com base no quanto a descrição dos itens é compatível com os requisitos do usuário, ou em como atendem as restrições desses requisitos. Nestes sistemas, uma função de similaridade estima o quanto a necessidade do usuário corresponde às recomendações.

Esses sistemas são particularmente úteis no contexto de itens que não são comprados com muita frequência ou em situações que não há muitas avaliações para os itens. Exemplos incluem itens como imóveis, automóveis, solicitações de turismo, serviços financeiros ou bens de luxo. Como os itens são comprados raramente e com diferentes tipos de opções detalhadas, é difícil obter um número suficiente de avaliações para um item em questão, surgindo daí a necessidade de utilizar as funções de similaridade (AGGARWAL, 2016b). Semelhantemente ao que acontece nos sistemas de recomendação baseado no conteúdo, sistemas baseados em conhecimento dispensam avaliações dos itens em algumas situações. Sua similaridade com sistemas baseado em conteúdo faz com que eles sejam considerados “parentes” por alguns pesquisadores (SMYTH, 2007).

Os sistemas baseados em conhecimento tendem a funcionar melhor do que outros no início de sua implantação, mas se não estiverem equipados com componentes de

aprendizado, poderão ser superados por outros métodos superficiais que podem explorar os logs da interação humano/computador (como na filtragem colaborativa) (RICCI; ROKACH; SHAPIRA, 2011a).

#### 3.2.1.4 Sistemas de recomendação demográfica

Nos sistemas de recomendação demográfica, as informações demográficas sobre o usuário são aproveitadas no processo de aprendizagem dos classificadores, que podem mapear informações demográficas específicas com as preferências dos usuários. Desse modo, as recomendações são feitas com base no perfil demográfico do usuário. O pressuposto é que recomendações diferentes devem ser geradas para diferentes nichos demográficos. Muitos sites adotam soluções de personalização simples e eficazes baseadas em dados demográficos. Por exemplo, os usuários são enviados para sites específicos com base em seu idioma ou país. Ou as sugestões podem ser personalizadas de acordo com a idade do usuário. Embora essas abordagens tenham sido bastante populares na literatura de *marketing*, houve relativamente pouca pesquisa apropriada sobre sistemas de recomendação utilizando informações demográficas (RICCI; ROKACH; SHAPIRA, 2011a).

#### 3.2.1.5 Sistema de recomendação baseado na comunidade

Esse tipo de sistema recomenda itens com base nas preferências dos amigos dos usuários. Esta técnica segue o ditado “Diga-me quem são seus amigos e eu direi quem você é” (ARAZY; KUMAR; SHAPIRA, 2009; BEN-SHIMON et al., 2007). Esse tipo de sistema de recomendação modela e adquire informações sobre as relações sociais dos usuários e as preferências dos amigos do usuário. A recomendação é baseada nas classificações fornecidas pelos amigos do usuário. De fato, esses sistemas de recomendação estão acompanhando o surgimento das redes sociais e permitem uma aquisição simples e abrangente de dados relacionados às relações sociais dos usuários (RICCI; ROKACH; SHAPIRA, 2011a). Devido ao crescimento das redes sociais, esse tipo de sistema vem gerando um interesse crescente na comunidade, sendo também referido como Sistemas de Recomendação Social, também descrito na seção 3.2.2.2 como parte dos Sistemas de Recomendação Sensíveis ao Contexto.

#### 3.2.1.6 Sistemas híbridos de recomendação

Esses sistemas de recomendação são baseados na combinação das técnicas mencionadas anteriormente. Um sistema híbrido que combina as técnicas A e B tenta usar as vantagens de A para corrigir as desvantagens de B. Por exemplo, os métodos de filtragem colaborativa sofrem de problemas de novos itens, ou seja, eles não podem recomendar itens que não possuem classificações, enquanto que tal problema não limita as abordagens de filtragem baseada em conteúdo, uma vez que a previsão de novos itens se baseia em seus atributos que normalmente são facilmente disponíveis. Dadas duas (ou mais) técnicas

básicas de sistemas de recomendação, é possível combina-las de varias maneiras para criar um novo sistema híbrido.

### 3.2.2 Sistemas de recomendação sensíveis ao contexto

Segundo Aggarwal (2016b), os sistemas de recomendação sensíveis ao contexto adaptam suas recomendações à informações adicionais que definem a situação específica sob a qual as recomendações são feitas. Essas informações adicionais são chamadas de contexto. Em diferentes domínios, como dados temporais, dados de localização e dados sociais, o contexto da recomendação desempenha um papel crítico, capaz de alterar totalmente o caráter da recomendação.

À vista disso, a noção de sistemas contextuais de recomendação foi desenvolvida para levar em conta as informações adicionais que surgem nesses domínios. Essa noção é usada com diferentes modificações para cada tipo de dado. As informações contextuais, que podem incluir horário, local ou perfis sociais, são levadas em consideração pelos sistemas de recomendação sensíveis ao contexto enquanto fazem recomendações.

Por exemplo, as peças de roupas recomendados por uma loja podem depender tanto da estação do ano quanto da localização do cliente. Além disso, certos feriados podem também influenciar no tipo de roupa que o cliente procura. Essas informações contextuais podem melhorar significativamente a eficácia do processo de recomendação, podendo elas serem agrupadas para gerar contextos mais específicos ainda ou usadas separadamente em contextos mais genéricos, como será apresentado nas subseções a seguir. Segundo Aggarwal (2016b), os sistemas de recomendação sensíveis ao contexto são incrivelmente poderosos, visto que as ideias subjacentes são relevantes para uma ampla variedade de configurações específicas de domínios.

Uma visão geral de sistemas de recomendação que utilizam esses aspectos contextuais é apresentada a seguir.

#### 3.2.2.1 Sistemas de recomendação baseados em localização

A localização é um contexto comumente usado nos sistemas de recomendação. Com a crescente popularidade de telefones celulares com GPS, o uso do contexto de localização é útil em uma variedade de cenários, como encontrar cinemas, restaurantes ou outros locais de entretenimento próximos ao usuário. A localização pode influenciar o processo de recomendação de várias maneiras, das quais as mais comuns são a utilização da localização para inferir determinadas preferências do usuário, tais como gosto, cultura, roupas, hábitos alimentares e etc. A localização é também uma informação contextual útil para recomendar lugares próximos ao local do usuário, tais como restaurantes, cinemas, parques, lojas e mais uma série de locais públicos e comerciais que possam interessar o usuário.

### 3.2.2.2 Sistemas de recomendação social

Para Aggarwal (2016b), da perspectiva dos sistemas de recomendação, o contexto social geralmente tem relevância ao se fazer as recomendações. Por exemplo, a escolha dos amigos, *tags* e círculos sociais de um usuário pode influenciar suas preferências e consequentemente o processo de recomendação. Sinha, Swearingen et al. (2001) mostraram que entre recomendações de um sistema de recomendação e recomendações de amigos, as recomendações dos amigos são preferidas. Considerando uma abordagem utilizando filtragem colaborativa, seria o equivalente a dizer que pelo fato dos usuários serem amigos, eles têm preferências semelhantes e isso torna-os semelhantes. Esse tipo de recomendação é bastante usado para recomendar novos amigos em redes sociais, visto que geralmente as pessoas compartilham os mesmos amigos ou fazem parte dos mesmos grupos sociais.

### 3.2.2.3 Sistemas de recomendação sensíveis ao tempo

As recomendações podem ser afetadas por muitos aspectos temporais, como dias da semana, feriados, estações e assim por diante. Uma recomendação relevante para o contexto do fim de semana pode não ser relevante para os dias da semana e vice-versa. As recomendações de roupas durante o verão e o inverno podem ser diferentes. Além disso, as recomendações podem ser úteis apenas durante um período limitado de tempo, tornando-se obsoletas após esse período.

Aggarwal (2016b) apresenta uma visão aprofundada sobre as diferentes formas de usar informações temporais para gerar uma eficácia maior da previsão, separando-os em dois tipos de modelos:

**Filtragem Colaborativa Temporal:** esses modelos utilizam as avaliações obtidas a partir de *feedback* explícito do usuário, porém essas avaliações podem ganhar ou perder importância em relação as outras dependendo do período, tempo, ou época na qual essas avaliações foram dadas. Alguns desses modelos consideram que avaliações mais recentes possuem uma importância maior que avaliações mais antigas e por isso atribuem um peso maior a elas durante as recomendações. Há também modelos que baseiam-se em períodos de tempo para realizar as recomendações, tais como os exemplos citados anteriormente das recomendações de peças roupas de uma loja que varia de acordo com a época do ano, mês, dia ou até mesmo a hora. Nesses métodos, o tempo se torna uma variável contextual que é explorada para fazer recomendações.

**Modelos Temporais Discretos:** são relevantes para o caso em que os dados subjacentes são recebidos como sequências de ações discretas. Esses dados podem ser encontrados em vários cenários de aplicações, a maioria dos quais associados a *feedback* implícito do usuário, em vez de avaliações explícitas. Alguns exemplos desses cenários são: recomendações de consultas a *websites*, baseando-se nas sequências de consultas feitas

anteriormente pelo usuário; compras de supermercado, considerando a ordem na qual o cliente escolhe os produtos como uma sequência temporal, com as recomendações sugerindo os próximos produtos que o cliente estaria interessado com base nos que ele já escolheu; sequências de cliques em *websites*, bem como as sequências de acesso as páginas nesses *websites* (FU; BUDZIK; HAMMOND, 2000; MOBASHER; COOLEY; SRIVASTAVA, 2000; MOBASHER et al., 2002).

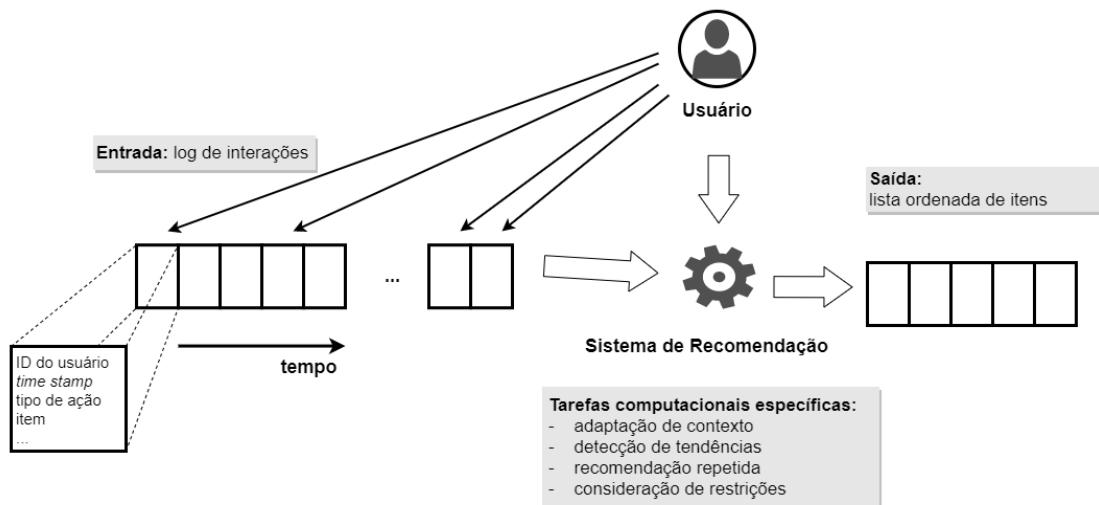
Os problemas abordados com *modelos temporais discretos* possuem características semelhantes ao problema abordado neste trabalho, em especial problemas que envolvem recomendações baseadas nas sequências de acesso à páginas web, podendo o problema abordado neste trabalho ser visto como uma extensão deste problema para o ambiente *offline* de uma aplicação *desktop*. Devido a isto, algumas definições mais aprofundadas destes cenários e modelos são abordadas na subseção 3.2.3.

### 3.2.3 Sistemas de Recomendações Sensíveis à Sequência

Aplicações em que objetivo é recomendar certos itens utilizando apenas as sequência de iterações anteriores do usuário, acabam muitas vezes não se adequando aos sistemas de recomendação tradicionais por não conterem certas informações que para esses sistemas tradicionais são relevantes, tais como dados de avaliações dos usuários ou atributos descritivos dos itens. Em tais situações, normalmente o que se tem são os *logs* de interação sequencial do usuário com a aplicação. Nesses cenários, sistemas de recomendação sensíveis à sequência possuem a capacidade de aprenderem a prever o próximo melhor item com base nos *logs*.

Em seu trabalho, Quadrana, Cremonesi e Jannach (2018), apresentam uma descrição mais detalhadas dos tipos de sistemas de recomendação que utilizam sequências de iterações do usuário para gerar recomendações e como eles divergem dos sistemas de recomendação tradicionais. Além disso, a partir da revisão dos trabalhos existentes na área, Quadrana, Cremonesi e Jannach fazem também uma categorização das tarefas e objetivos das recomendações correspondentes a esses sistemas. Quadrana, Cremonesi e Jannach ressaltam que alguns cenários discutidos em seu trabalho também representam certas formas de sistema de recomendação baseado em contextos, discutidos na subseção 3.2.2, tais como o sistemas de recomendação sensíveis ao tempo 3.2.2.3. Porém, a diferença reside no fato de que sistemas de recomendação sensíveis à sequência tem foco maior na ordem sequencial dos eventos em si, e menos no momento exato da ocorrência das iterações anteriores do usuário. Na Figura 3.1 é possível observar uma visão geral de como Quadrana, Cremonesi e Jannach descrevem o fluxo em um sistema de recomendação sensível à sequência.

Figura 3.1 – Visão geral dos problemas de recomendação sensíveis à sequência



Fonte: QUADRANA; CREMONESI; JANNACH, 2018.

### 3.2.3.1 Caracterização dos sistemas de recomendação sensíveis à sequências

Os problemas de recomendação sensíveis à sequência diferem da configuração tradicional de várias formas, porém, de maneira geral, as principais diferenças estão no modo como os dados de entrada e saída são organizados, bem como nas tarefas computacionais realizadas pelos recomendadores. Resumidamente, as diferenças consistem em:

**Entrada:** Enquanto que na configuração tradicional a entrada dos sistemas de recomendação consistem de matrizes usuário-item que representam as avaliação dos usuários para cada item específico, a entrada principal para os problemas de recomendação sensíveis à sequência é uma lista ordenada das ações anteriores do usuário em função do tempo. Além disso, os usuários podem ser conhecidos pelo sistema ou podem ser anônimos. Informações extras como o tipo de ação que o usuário está realizando também podem ser consideradas no processo de recomendação, além de outras informações que descrevem com mais detalhes o usuário, os itens ou a própria ação realizada pelo usuário.

**Saída:** Em resumo, a saída de um sistema de recomendação sensíveis à sequência são listas ordenadas de itens. Os resultados são semelhantes aos de uma recomendação de recomendador tradicional com classificação de itens, que tem como saída uma lista de itens relevantes. Entretanto, a ordem dos objetos na lista de recomendações é relevante em alguns cenários de recomendação sensíveis à sequência. Além disso, há cenários em que o usuário, em vez de considerar a lista de recomendações como um conjunto de alternativas, deve considerar todas as recomendações seguindo a ordem

fornecida. Exemplos típicos incluem a recomendação de uma sequência de faixas musicais (*playlist*) ou a recomendação de uma série de videoaulas em um curso.

**Tarefas computacionais:** Dentre as diversas tarefas computacionais de sistemas de recomendação encontrado na literatura, alguns estão presentes apenas em sistemas de recomendação sensíveis à sequência, como tarefas nas quais a ordem das ações é relevante, ou podem ser padrões de co-ocorrência, nos quais é importante que duas ações ocorram juntas, por exemplo, dentro da mesma sessão. Em alguns casos, os padrões de distância também podem ser relevantes, por exemplo, quando o problema é calcular um bom momento para lembrar o usuário de algo através de uma recomendação. Uma característica importante dos recomendadores sensíveis à sequência é que muitas vezes esses padrões são obtidos de maneira implícita, não necessitando que o usuário avalie quais ações são mais ou menos importantes.

### 3.2.3.2 Categorização das tarefas em sistemas de recomendação sensíveis à sequência

Como pôde ser observado nas subseções anteriores, diversas tarefas podem ser realizadas com o auxílio de sistemas de recomendação sensíveis à sequência nos mais variados cenários e aplicações. Segundo Quadrana, Cremonesi e Jannach (2018), é possível identificar quatro principais objetivos na literatura que podem ser alcançados com o auxílio de recomendadores sensíveis à sequência e categorizá-los a partir disso. As definições dessas categorias podem ser vistas de maneiras gerais como:

**Adaptação de contexto:** a principal característica de sistemas de recomendação que se adaptam ao contexto é o fato de considerarem não só os interesses do usuário, mas também o contexto no qual está inserido. Em recomendadores sensíveis à sequência, esses contextos podem ser vistos como as iterações que o usuário teve com a aplicação. Dependendo da disponibilidade de dados de histórico dos usuários e da relevância das interações mais recentes, pode-se diferenciar os seguintes tipos de situações de adaptação ao contexto:

- **Recomendação baseada nas N-últimas interações:** são contextos nos quais apenas as últimas  $n$ -interações do usuário são levadas em conta para a recomendação. Por exemplo, em um site de comércio eletrônico, o sistema de recomendação pode considerar como relevante somente os  $n$ -últimos itens pesquisado pelo usuário, ao invés de utilizar todo o histórico de pesquisas do usuário para recomendar os próximos itens. Essa limitação pode ser útil em situações que não há muita informação de interações passadas, quando se trata de um usuário anônimo ou quando interações muito antigas não influenciam na predição do próximo item.

- **Recomendação baseada na sessão:** nesse cenário, apenas a última sequência de interações do usuário é conhecida, sendo essa sequência limitada a uma sessão. No exemplo do site de comércio eletrônico, as recomendações seriam baseadas apenas na sequência de itens que o usuário pesquisou na sessão atual.
- **Recomendação sensível à sessão:** e por último, há cenários em que se tem o histórico de interações do usuário de todas as sessões anteriores e atual. Essa situação é possível quando os usuários podem ser identificados, permitindo ao sistema de recomendação sensível à sequência recomendar utilizando as interações do usuário a curto ou longo prazo.

**Deteccção de tendência:** Sistemas de recomendação sensíveis à sequência também podem ser utilizados para detectar tendências em determinados conjuntos de dados, porém esse problema é pouco explorado. Os tipos de informação de tendências que podem ser extraídos dos dados sequenciais podem ser diferenciados da seguinte maneira:

- **Tendências da comunidade:** Considerar a popularidade dos itens na comunidade do usuário é uma boa estratégia para gerar recomendações mais eficientes. Entretanto, em determinados domínios, a popularidade dos itens pode mudar com o tempo ou as tendências podem ter características sazonais, como por exemplo o tipo de roupa em determinadas estações do ano. Sistema de recomendação sensíveis à sequência possuem a capacidade de detectar essas mudanças nos padrões de popularidade a partir do histórico de interações dos usuários e adaptar-se a essas mudanças de forma automática.
- **Tendências individuais:** Assim como o interesse da comunidade pode mudar com o tempo, os interesses pessoais do usuários também podem mudar, seja naturalmente, a partir de algum evento ou influenciado pelas tendências da comunidade. Devido a quantidade menor de dados em relação ao domínio das tendências da comunidade, esse problema acaba sendo mais difícil.

**Recomendação repetida:** Em algumas aplicações, faz sentido recomendar itens já conhecidos ou já adquiridos pelo usuário. Tais cenários podem ocorrer em situações que o usuário costuma comprar com frequência certos itens ou em casos que o usuário gostaria de ser lembrado de certo item, como em uma “lista de interesses” em uma aplicação de comércio eletrônico. Sistemas de recomendação sensíveis à sequência podem auxiliar essas tarefas da seguinte maneira:

- **Identificação de padrões repetidos no comportamento do usuário:** detectar padrões de comportamento repetidos a partir do histórico de interação do usuário, como por exemplo a compra de itens consumíveis, tais como tinta

para impressora ou alimentação. A detecção de ações repetidas pode ser bem relevantes no ambiente de aparelhos móveis, como *smartphones*, onde padrões de comportamento repetido do usuário podem ser usados para fornecer atalhos para aplicativos que são frequentemente iniciados em uma determinada sequência pelo usuário. O objetivo nesse caso é melhorar a experiência do usuário com o aparelho.

- Recomendações repetidas como lembretes: nesse cenário, o objetivo principal é ajudar os usuários a lembrarem de coisas que eles acharam interessantes no passado. Esses lembretes variam de acordo com o domínio da aplicação. Por exemplo, no domínio de recomendações musicais, recomendar ao usuário artistas que ele gostava mas não tem escutado recentemente, ou determinado canal no YouTube que o usuário deixou de assistir aos vídeos. Essas recomendações podem ser feitas detectando automaticamente esses cenários ou a partir de uma lista feita pelo próprio usuário. Nesses cenários, os sistemas de recomendação sensíveis à sequência podem fazer as recomendações quando detectam um certo número de interações do usuário sem a presença desses itens.

#### **Consideração de restrições de ordem e padrões de sequências observados:**

Considerar a ordem dos itens nas sequências pode ter um papel fundamental em algumas tarefas de recomendação em sistemas de recomendação sensível à sequência. Certas restrições na ordem podem ser definidas com objetivo de especificar ainda mais o cenário das recomendações, como por exemplo na recomendação de itens que dão continuidade a outros itens, tais como filmes, livros ou acessórios. Além disso, pode-se tentar identificar padrões de consumo sequencial a partir do comportamento do usuário, como detectar que se um usuário assistiu um certo filme, ele provavelmente vai querer assistir a continuação dele.

Segundo Quadrana, Cremonesi e Jannach (2018), quando se considera esses fatores que podem influenciar a ordem, diversas variações de uma aplicações podem ter que ser levada em conta, como por exemplo a relevância ou não da ordem das recomendações, da ordem exata dos eventos passados ou da existência de restrições implícitas de ordem. No contexto da recomendação da próxima faixa de música, as restrições da ordem das recomendações podem determinar se as faixas de uma *playlist* possuem alguma continuidade, tal como o aumento do andamento das músicas. Já no contexto de comércio eletrônico, a existência de restrições de ordem implícita define quando faz sentido recomendar certo acessório associado a um item e quando não faz, como por exemplo o fato de fazer sentido recomendar ao usuário uma película para o celular que ele acabou de comprar, mas não um celular para a película que ele acabou de comprar.

### 3.3 Avaliando sistemas de recomendação

Um dos primeiros passos para selecionar um algoritmo apropriado para determinado problema é decidir em quais propriedades do problema deve se concentrar ao fazer essa escolha. De fato, sistemas de recomendação têm uma variedade de propriedades que podem afetar a experiência do usuário, como acurácia, robustez, escalabilidade e assim por diante. No início a maioria dos sistemas de recomendação propostos eram avaliados e classificados com base na acurácia da previsão das escolhas do usuário. No entanto, atualmente é amplamente aceito que, apesar da acurácia ser crucial na avaliação do recomendador, utilizar somente ela como métrica é insuficiente para avaliar sistemas de recomendação (SHANI; GUNAWARDANA, 2011; AGGARWAL, 2016a; GE; DELGADO; JANNACH, 2010).

Além disso, o objetivo de um sistema de recomendação nem sempre se trata de conseguir antecipar os gostos do usuário, pois os usuários também podem estar interessados em descobrir novos itens, explorar rapidamente diversos itens, preservar sua privacidade, nas respostas rápidas do sistema e em muitas outras propriedades da interação com o mecanismo de recomendação. Daí a importância em identificar o conjunto de propriedades que podem influenciar o sucesso de um sistema de recomendação no contexto de uma aplicação específica e avaliá-lo com base nisso.

Sistemas de recomendação podem ser avaliados usando métodos *online* ou *offline*. Em um sistema *online*, as reações do usuário são medidas em relação às recomendações apresentadas. Portanto, a participação do usuário é essencial em sistemas *online* para medir o impacto direto do sistema de recomendação no usuário final, além de poder fornecer uma verdadeira medida da eficácia do sistema. No entanto, como as avaliações *online* exigem a participação ativa do usuário, muitas vezes não é possível usá-las em *benchmarking* e pesquisa.

Já os métodos *Offline* são, de longe, os métodos mais comuns para avaliar sistemas de recomendação de uma perspectiva de pesquisa e prática. Nesses métodos, os experimentos são realizados usando um conjunto de dados pré-coletados de usuários que escolhem ou classificam itens. Com esse conjunto de dados, pode-se tentar simular o comportamento dos usuários que interagem com um sistema de recomendação. Ao fazer isso, assume-se que o comportamento do usuário quando os dados foram coletados será semelhante ao comportamento do usuário quando o sistema de recomendação for implantado. Para Shani e Gunawardana (2011), a vantagem desse método é que não requerem interação com usuários reais e, portanto, permitem comparar uma ampla gama de algoritmos candidatos a baixo custo. Já a desvantagem é que só é possível analisar algumas características do sistema, normalmente associadas ao poder de previsão dos algoritmos.

Do exposto, tem-se que, considerando que a origem dos dados provém de uma base

de dados que representa o histórico de interação do usuário, as metodologias de avaliação *offline* se adequam melhor ao contexto deste trabalho, avaliando a capacidade do sistema de recomendação a partir da capacidade de previsão dos algoritmos sobre esses dados. Nesse contexto, diferentes métricas podem ser consideradas para verificar a confiabilidade das previsões, qualidade das recomendações, diversidade da lista de recomendações, a popularidade geral dos itens recomendados ou o quanto o algoritmo de recomendação contempla o catálogo de itens. Com isto, as seções seguintes apresentam uma visão mais detalhada dos métodos de avaliação *offline* em recomendadores sensíveis a sequência, no qual os sistemas de recomendações de próxima funcionalidade em softwares estão inseridos.

### 3.3.1 Avaliando sistemas de recomendação sensíveis à sequência

Embora, a princípio, métodos de avaliação *offline* e *online* possam ser aplicados a sistemas de recomendação sensíveis à sequência, assim como nos demais sistemas de recomendação, algumas estratégias de avaliação ligeiramente diferentes são usadas em estudos *offline*. Segundo Quadrana, Cremonesi e Jannach (2018), vários aspectos dessas estratégias de avaliação *offline* estão intimamente relacionadas às usadas para Sistemas de Recomendação Sensíveis ao Tempo (apresentados na Seção 3.2.2.3), principalmente no que diz respeito a ordem temporal das ações do usuário no conjunto de dados. Isso se dá pois, diferente de uma avaliação *online*, em que as sequências de ações do usuário são montadas em tempo real, na avaliação *offline* tudo que se tem são registros das ações passadas do usuário, que devem estar ordenadas em sequências temporais para representar ao máximo o comportamento que o usuário teve.

#### 3.3.1.1 Particionamento de conjunto de dados

Dentre as etapas de avaliação de qualquer algoritmo de aprendizado de máquina, uma das que mais pode influenciar nos resultados obtidos é o particionamento do conjunto de dados utilizado. No geral, esse particionamento consiste em separar as amostras em dois conjuntos, sendo um para o treinamento do algoritmo e outro para avaliar a acurácia das previsões dos algoritmos treinados, também chamado de conjunto de teste. Diversas estratégias podem ser utilizadas para definir tanto quais dados serão usados para treinamento/teste bem como a quantidade de dados para cada uma dessas etapas.

Em geral, essa separação é feita com 80% das amostras presentes no conjunto de dados sendo utilizados para treinamento e os 20% restantes utilizados para teste. Entretanto, no que se refere a quais dados serão utilizados para cada etapa, enquanto que no geral algoritmos de aprendizado de máquina apenas selecionam aleatoriamente as amostras para cada etapa, segundo Quadrana, Cremonesi e Jannach (2018), os sistemas de recomendação sensíveis a sequência possuem duas formas de particionamento que são

influenciados pela característica das sequências temporais dos dados: divisão no **nível de evento** e no **nível da sessão**.

O particionamento a **nível de evento** consiste em dividir as amostras a partir de um certo evento, com todos as amostras anteriores a esse ponto pertencendo ao conjunto de treinamento e as seguintes pertencendo ao conjunto de teste. Quando as amostras são particionadas no **nível da sessão**, considera-se que as sequências de eventos contidos dentro de uma sessão não devem ser separados, assim o ponto de divisão sempre ocorre entre o fim de uma sessão e o início de outra.

### 3.3.1.2 Definição dos itens de alvos

Em sistemas de recomendação tradicionais, o que se deseja é predizer as avaliações do usuário para determinados itens ou ranquear os melhores itens para o usuário, tendo como objetivo recomendar itens que realmente são relevantes para o usuário. Porém, em sistemas de recomendação sensíveis a sequências, normalmente o que se está interessado é em predizer ações futuras de um usuário (ou, geralmente, eventos) de determinados tipos e associados a certos itens (QUADRANA; CREMONESI; JANNACH, 2018).

Além disso, os dados de testes podem também incluir sequências de eventos ou sessões inteiras reservadas para o conjunto de testes. Nesses casos, a ideia geral na maioria das abordagens de avaliação é ocultar um subconjunto ou todos os eventos da sequência a ser testada e prever as próximas ações do usuário. As duas principais abordagens de avaliação encontradas na literatura são:

- **Previsão agnóstica de sequência:** Nesse caso, a ordem das ações ocultas não é relevante, sendo uma recomendação considerada bem-sucedida se predizer corretamente uma ou mais das ações ocultas. Por exemplo, ao avaliar um sistema de *e-commerce* que gera recomendações com base na compra anterior de outros produtos, mesmo que o sistema recomende uma lista com vários itens não relevantes ao usuário, se entre as recomendações houver um item relevante, então essa recomendação como um todo pode ser considerada relevante, tal como feito em Zhao et al. (2012).
- **Predição de próximo item dado-N:** Nesta abordagem, os primeiros  $N$  elementos da sequência são “revelados” ao algoritmo e a tarefa do recomendador é prever a próxima ação imediata. Nesses casos, não são considerados apenas os eventos ocorridos, mas também a ordem na qual eles ocorreram. O valor de  $N$  pode ser tanto um valor predefinido (JANNACH; LERCHE; JUGOVAC, 2015) como incrementalmente aumentado (HIDASI et al., 2015; QUADRANA et al., 2017).

### 3.3.2 Métricas de Avaliação

Para muitos cenários de sistemas de recomendação sensíveis à sequência, métricas de classificação padrão e métricas de *ranking* podem ser usadas para avaliação de desempenho. Segundo Quadrana, Cremonesi e Jannach (2018), as métricas de classificação padrão, como precisão e sensibilidade, podem ser utilizadas em cenários de recomendações clássicas sensíveis ao contexto e que utilizam as técnicas baseadas em sequências apenas para gerar recomendações melhores. Porém, quando o domínio da aplicação exige recomendações que atendam a certas restrições de ordem, métricas de *ranking* devem ser preferidas às métricas de classificação.

#### 3.3.2.1 Acurácia

Em sistemas de recomendação, a acurácia é uma importante métrica que auxilia a avaliar quão bem o sistema consegue fazer predições dos itens que interessam ao usuário. Já no domínio dos sistemas de recomendação sensíveis a sequência, a acurácia ajuda a mensurar a precisão da predição da próxima ação mais provável do usuário com base na sequência de ações passadas adquiridas através de *feedback* implícito, podendo essa próxima ação estar contida em uma lista de próximas ações mais prováveis, ranqueada com base na probabilidade de cada ação.

A acurácia da previsão é de longe uma das propriedades mais discutidas na literatura de sistemas de recomendação (SHANI; GUNAWARDANA, 2011; AGGARWAL, 2016a). Uma suposição básica em um sistema de recomendação é que um sistema que fornece previsões mais precisas será preferido pelo usuário. Assim, muitos pesquisadores tem preferência por encontrar algoritmos que fornecem melhores previsões (SHANI; GUNAWARDANA, 2011).

Em experimentos *offlines*, normalmente tem-se um conjunto de dados que consiste em itens que cada usuário usou. Em seguida, são selecionados alguns dados de certos usuários como casos de teste, ocultando-se algumas das seleções para que o recomendador preveja o conjunto de itens que tais usuários usariam. Desse modo, têm-se as seguintes possibilidades:

Tabela 3.1 – Possíveis resultados de uma recomendação de um item para um usuário.

|           | Recomendado              | Não recomendado           |
|-----------|--------------------------|---------------------------|
| Usado     | Verdadeiro positivo (VP) | Falso negativo (FN)%      |
| Não usado | Falso positivo (FP)      | Verdadeiro negativo (VN)% |

Fonte: (SHANI; GUNAWARDANA, 2011).

Um observação importante a se fazer é que como em avaliações *offlines* os dados

utilizados são apenas aqueles conhecidos pelos usuários, é necessário supor que itens não utilizados não seriam usados mesmo que tivessem sido recomendados, sendo eles desinteressantes ou inúteis para o usuário. Dessa forma, pode-se calcular a acurácia, precisão e sensibilidade da seguinte maneira:

$$Precisão = \frac{\#VP}{\#VP + \#FP} \quad (3.1)$$

$$Sensibilidade = \frac{\#VP}{\#VP + \#FN} \quad (3.2)$$

$$Acurácia = \frac{\#VP + \#VN}{\#VP + \#FN + \#VN + \#FP} \quad (3.3)$$

### 3.3.2.2 Hit Rate (HR)

O HR expressa a quantidade de acertos sobre todas as predições feitas e é descrito da seguinte maneira:

$$HR = \frac{\#hits}{n}, \quad (3.4)$$

em que cada *hit* representa uma recomendação correta e  $n$  representa o número total de recomendações. Além disso, quando as recomendações são dadas em listas com as *top-N* recomendações, o HR leva em consideração apenas se o item correto está entre esses *top-N*, não levando em consideração a posição deste item na lista. Em outras palavras, o HR trata todos os *hits* igualmente, independentemente de onde eles apareçam na lista dos  $N$  principais itens recomendados.

### 3.3.2.3 Average Reciprocal Hit-Rate (ARHR)

O ARHR (DESHPANDE; KARYPIS, 2004a) foi projetada para conjuntos de dados de *feedback* implícito e, assim como as demais métricas de *rank*, tem como objetivo avaliar a qualidade da lista de recomendações dada pelo sistema de recomendação. A ideia básica é que cada item tenha uma utilidade para o usuário, seja ele muito ou pouco útil. O ideal é que os itens realmente úteis ao usuário sejam colocados o mais alto possível na lista de recomendações, pois são mais propensos a serem notados (em virtude de sua posição) e eventualmente selecionados.

Deshpande e Karypis (2004a) destacam que uma limitação da métrica *hit-rate* (HR) é o fato dessa métrica não levar em consideração a posição do item correto na lista dos *top-N* itens recomendados. Ou seja, um acerto que ocorre na primeira posição é tratado igualmente com um acerto que ocorre na *n*ésima posição. Essa limitação é tratada pelo ARHR, que recompensa cada acerto com base em onde ocorreu na lista.

Se  $h$  é o número de acertos (*hits*) que ocorreram nas posições  $p_1, p_2, \dots, p_h$  nas listas *top-N* (ou seja,  $1 \leq p_i \leq N$ ), tem-se que para  $n$  recomendações, o ARHR é igual a

$$ARHR = \frac{1}{n} \sum_{i=1}^h \frac{1}{p_i}. \quad (3.5)$$

Desse modo, os *hits* que ocorrem nas primeiras posições das listas *top-N* têm um peso maior que os *hits* que ocorrem nas últimas posições das listas. O valor mais alto da ARHR é igual ao HR e acontece quando todos os acertos ocorrem na primeira posição, enquanto o valor mais baixo da ARHR é igual à  $\frac{HR}{N}$ , ocorrendo quando todos os acertos se dão na última posição das listas *top-N*.

### 3.3.2.4 Normalized Discounted Cumulative Gain (NDCG)

Em alguns cenários, a ideia de que os usuários estão interessados apenas nos itens mais no topo da lista de recomendação, não dando muita atenção nos últimos itens, não é aplicável. Um exemplo disso é em sistemas de recomendação de notícias, na qual o usuário pode também estar interessado nas notícias listadas por último. Um outro exemplo é quando o usuário está procurando um item específico em uma lista de recomendação não tão grande, nesse caso, é conveniente que o item esteja nas primeiras posições, porém o importante é que o item correto seja recomendado, mesmo que ele esteja nas últimas posições.

Nesses cenários, o NDCG é muito útil por ser uma métrica de *rank* em que as posições são descontadas logaritmicamente, não penalizando tanto os itens das últimas posições quanto o ARHR, visto que o NDCG decai mais lentamente (SHANI; GUNAWARDANA, 2011). O NDCG foi proposto como uma normalização do *discounted cumulative gain* (DCG), sendo ambos descritos no trabalho de Järvelin e Kekäläinen (2002).

Nesta métrica, a relevância de cada item em uma lista com os *top-N* itens recomendados é usada como uma medida de valor agregado com base na sua posição na lista, sendo o ganho somado progressivamente da posição 1 para  $n$ . Desse modo, considerando uma lista de recomendação com os  $N$  principais itens, o cálculo do DCG acumulado na posição  $p$  desta lista é definido como:

$$DCG_p = \sum_{i=1}^p \frac{rel_i}{\log_2(i+1)}, \quad (3.6)$$

em que  $rel_i$  o valor da relevância do item  $i$  e sendo o cálculo este valor definido de acordo com a necessidade do recomendador de maneira independente do DCG. Por exemplo, quando os itens recomendados têm relevância classificada apenas em se é útil ou não para o usuário, os valores de  $rel$  são binários, ou seja,  $rel_i \in \{0, 1\}$ .

O NDCG é definido por (Järvelin; Kekäläinen, 2002) como a razão do *discounted cumulative gain* (DCG) pelo seu valor ideal, o *ideal discounted cumulative gain* (IDCG). O IDCG, por sua vez, é definido como o valor máximo possível do DCG na posição  $p$ :

$$IDCG_p = \sum_{i=1}^{REL_p} \frac{rel_i}{\log_2(i+1)}, \quad (3.7)$$

em que  $REL_p$  representa a lista ordenada da relevância dos  $N$  principais itens até a posição  $p$ . Com isso, o NDCG em  $p$  é dado por:

$$NDCG_p = \frac{DCG_p}{IDCG_p}. \quad (3.8)$$

Vale ressaltar que em um algoritmo de recomendação perfeito, o  $DCG_p$  será o mesmo que o  $IDCG_p$  produzindo um NDCG de 1,0. Além disso, os valores do NDCG para todas as recomendações podem ser calculados para obter uma medida do desempenho médio do algoritmo de recomendação.

### 3.4 Conclusão

Neste capítulo, é abordado, do ponto de vista histórico e funcional, os diversos tipos de sistemas de recomendação e como o problema de recomendação de próxima funcionalidade em softwares se enquadra no que é relatado na literatura. Também é apresentado alguns trabalhos relacionados, como a recomendação de páginas em sites. Entretanto, não foi encontrado abordagens semelhantes no que diz respeito a softwares *offlines*, como é o caso deste trabalho. Além disso, são descritas métricas de desempenho capazes de mensurar a eficácia e a qualidade das predições de um sistema de recomendação. Dessa forma, foi possível modelar a metodologia utilizada no experimentos. A partir da visão geral que este capítulo trás sobre sistemas de recomendação, é possível compreender a relevância que um sistema de recomendação de próxima funcionalidade trás.

## 4 Sistema de recomendação de próxima funcionalidade

### 4.1 Apresentação do problema

De maneira geral, o problema consiste em recomendar as próximas funcionalidades que o usuário gostaria de acessar em um software/sistema computacional, baseando-se nas funcionalidades anteriormente acessadas por esse usuário. Tal problema é semelhante a um sistema de recomendação em *e-commerce* (JANNACH; LERCHE; JUGOVAC, 2015), no sentido de que o usuário recebe recomendações de produtos relacionados aos que ele acessou até o momento, assim como também é semelhante a sistemas de recomendação de páginas de *websites* (FU; BUDZIK; HAMMOND, 2000; DESHPANDE; KARYPIS, 2004b), que recomendam outras páginas à partir dos padrões de acessos presentes no histórico do usuário.

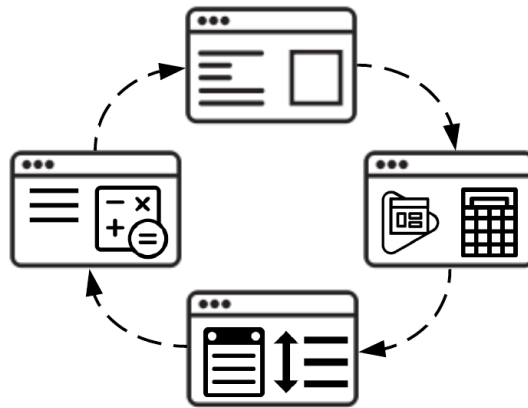
De forma semelhante a esses recomendadores, a melhor maneira de implementar um sistema de recomendação de próxima funcionalidade para usuários de um sistema é analisando os padrões de interação do usuário com o software (daqui em diante chamado de sistema computacional ou apenas sistema por ser um termo mais abrangente). Uma das maneiras de se obter essas informações seria registrando as interações do usuário com as funcionalidades do sistema em *logs*, o que se assemelharia a um histórico de acesso. A partir desse histórico seria possível implementar duas abordagens:

1. **pares (funcionalidade-funcionalidade):** de forma semelhante ao que acontece com produtos relacionados em *e-commerces*, poderia-se associar certas funcionalidades à outras. Assim, sempre que uma funcionalidade  $A$  fosse acessada a partir de outra funcionalidade  $B$ , criaria-se o par de funcionalidades  $(A, B)$ .
2. **sequências de funcionalidades:** nesse caso, cada funcionalidade seria associada à sequência de funcionalidades acessadas antes dela, sendo cada sequência única de acordo com a ordem na qual as funcionalidades foram acessadas.

Apesar de ambas as abordagens fazerem sentido no escopo do problema, a visão de sequência se adéqua melhor ao problema, visto que existe um fluxo recorrente de acesso às funcionalidades (Figura 4.1). Além disso, fluxos muito específicos dificilmente seriam considerados na abordagem de pares de funcionalidades, de modo que, quanto maior a sequência considerada, mais particular ela se torna. Desse modo, o problema pode ser interpretado como um problema de predição de série temporal, tendo em vista que as

sequências de funcionalidades estão ordenadas em função do instante em que cada uma foi acessada. Com isso, as recomendações das funcionalidades do sistema em questão correspondem as previsões dessa série.

Figura 4.1 – Representação de uso de funcionalidades recorrentes



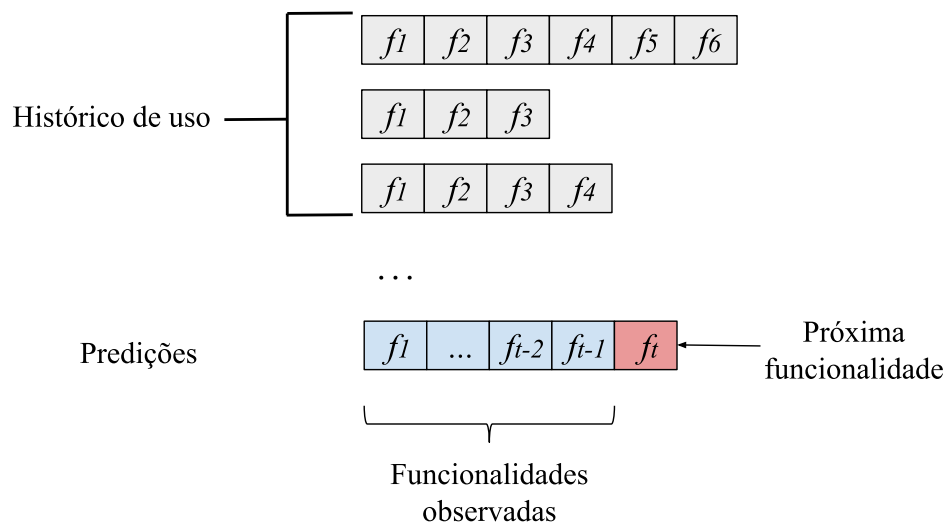
Fonte: Próprio autor.

Uma característica inerente ao problema de recomendação de próxima funcionalidade é que há repetição de funcionalidades, ou seja, as funcionalidades são acessadas mais de uma vez pelo usuário, portanto o sistema de recomendação deve ser capaz de recomendar itens repetidos, conforme descrito na subseção 3.2.3.2. Apesar de parecer algo frequente em sistemas de recomendação, esse é um comportamento evitado nos principais sistemas de recomendação, tais como em plataformas de *streaming* de filmes, *e-commerce*, redes sociais e etc., visto que normalmente os usuários destes sistemas não tem tanto interesse em consumir itens repetidos.

Uma outra característica importante quando se está trabalhando com histórico de ações dos usuários é a delimitação dessas ações em subconjuntos que expressem um começo e fim das sequências de ações. Por exemplo, não faz sentido considerar todas as interações do usuário de um sistema qualquer em um período de 3 dias como parte de uma sequência única eventos correlacionados, visto que, normalmente, o usuário começa e encerra suas atividades todos os dias. Essas delimitações podem ser tanto temporal (tais como o histórico dos navegadores) como a partir de eventos que caracterizem o início e o fim de um fluxo de ações correlacionadas, como por exemplo a ação de abrir e fechar o sistema ou realizar o *login* e *logout*. Desse modo, pode-se então caracterizar essa sequência de ações, com início e fim, como uma sessão do usuário no sistema.

A Figura 4.2 ilustra bem esse comportamento, na qual cada sequência de funcionalidades no histórico representa uma sessão do usuário. A partir do histórico, é possível treinar os algoritmos de predição, de modo que, com o algoritmo treinado, é possível prever as próximas funcionalidades relevantes para o usuário com base nas funcionalidades observadas até o momento na sessão atual.

Figura 4.2 – Descrição do problema de recomendação de próxima funcionalidade. Blocos cinzas são o histórico de funcionalidades acessadas pelo usuário no passado. Blocos azuis representam as funcionalidades acessadas mais recentemente, enquanto que o bloco vermelho representa a próxima funcionalidade que o usuário deseja acessar.



Fonte: Próprio autor.

#### 4.1.1 Generalização das sequências

Até o momento, o principal cenário descrito para este problema foi o uso do sistema por parte de um usuário específico, com o algoritmo de recomendação tentando prever as funcionalidades relevantes a esse usuário com base em seu histórico de uso. Porém, a partir desse cenário, pode-se presumir outros dois:

1. O primeiro seria o cenário ideal, no qual um usuário sabe exatamente o que está fazendo e o sistema de recomendação aprende corretamente, a partir do histórico desse usuário, quais funcionalidades recomendar no momento adequado.

2. O segundo cenário, porém, seria de um usuário inexperiente acessando várias funcionalidades aleatórias em busca da que ele deseja, o que faria o sistema de recomendação aprender esse comportamento e passar a recomendar erroneamente.

ambos os cenários descritos são perfeitamente plausíveis, porém o segundo é totalmente indesejado. Uma alternativa simples e inspirado na filtragem colaborativa (subseção 3.2.1.1) para tentar amenizar esse problema, seria utilizar modelos treinados com dados de usuários mais experientes para auxiliar os usuários novos. Além de recomendar funcionalidades mais adequadas, o recomendador auxiliaria usuários novos que ainda não tenham um fluxo de trabalho bem definido, que não conheçam bem o sistema em questão ou não possuem dados suficientes para treinar os modelos.

A ideia base da filtragem colaborativa é que pessoas que gostaram de um mesmo item tendem a gostar de forma semelhante de outros itens. Essa ideia pode ser estendida ao problema recomendação de próxima funcionalidade ao considerar que pessoas que utilizam um mesmo sistema tendem a escolher certas funcionalidades com mais frequência ao realizarem tarefas semelhantes.

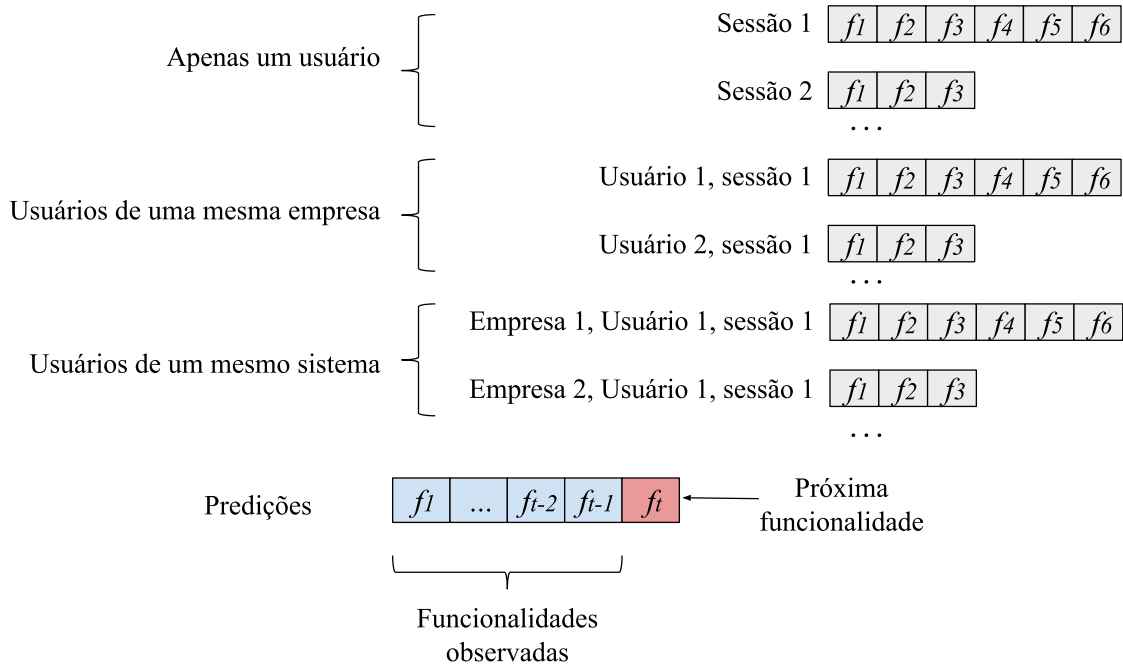
Aprofundando-se ainda mais nesse raciocínio, pode-se dizer que usuários que, além de usarem o mesmo sistema, são funcionários de uma mesma empresa contratante desse sistema, tendem a realizar tarefas mais semelhantes ainda. Dessa forma, modelos treinados com dados de outros usuários poderiam ser utilizados para gerar recomendações a usuários novos, aproveitando-se das experiências já adquiridas.

Tendo em vista isso, três tipos de modelos podem ser considerados:

- **um usuário:** treinados apenas com dados de um usuário e que só podem ser usados por esses mesmos usuários.
- **uma empresa:** treinados com todos os dados dos usuários de uma empresa que utiliza determinado sistema, podendo este modelo ser usado por qualquer usuário da empresa.
- **um sistema:** treinados com todos os dados dos usuários de um sistema, podendo este modelo ser usado por qualquer usuário do sistema.

A Figura 4.3 ilustra de uma forma mais clara como os modelos são treinados: (i) um usuário, (ii) uma empresa e (iii) um sistema. Do exposto, tem-se recomendação específica para um usuário, na qual se leva em consideração o histórico de utilização do sistema realizado por um certo usuário. Tem-se ainda recomendação por empresa, opção na qual o histórico de utilização refere-se as ações realizadas pelos diversos usuários de uma empresa específica. Por fim, tem-se a recomendação por sistema, em que o histórico de utilização refere-se as ações realizadas por todos os usuários da empresa que utilizam um determinado sistema.

Figura 4.3 – sequência de uso de funcionalidades

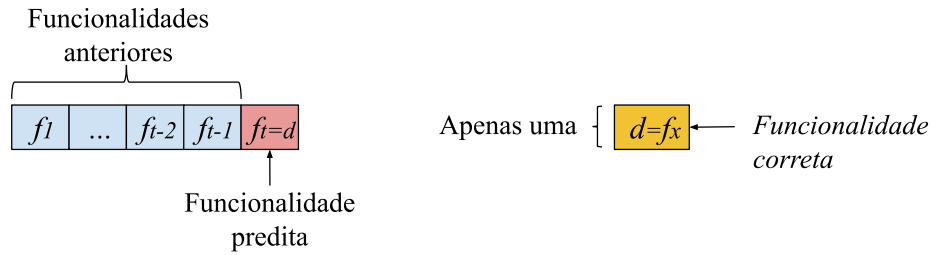


Fonte: Próprio autor.

#### 4.1.2 Ranking de próximas funcionalidades

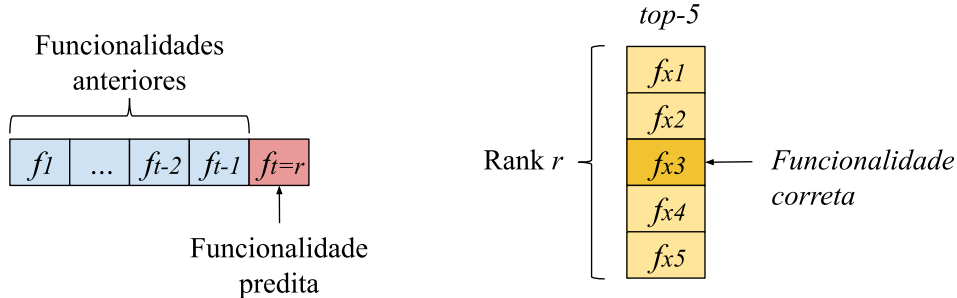
Conforme visto nos capítulos anteriores, muitas vezes a saída de um sistema de recomendação consiste não somente de um item, mas de uma lista de itens mais prováveis. Evidentemente, isso aumenta as chances de o usuário receber uma recomendação relevante, podendo ainda a qualidade dessa lista ser quantificada de acordo com as métricas citadas na seção 3.3.2. Na maioria das vezes essa lista vem limitada a uma quantidade  $N$  de itens ranqueados de acordo com sua relevância.

Para facilitar o entendimento, essa lista de itens mais relevantes é chamada de *top-N*, no qual  $N$  expressa a quantidade de itens presentes na lista. Desse modo, considerando uma lista de funcionalidades  $L$  de tamanho  $t_{>0}$  e com  $f$  funcionalidades observadas até então, sendo  $1 \leq f \leq t - 1$ , tem-se que a lista *top-N* pode ter tamanho  $N$  de  $1 \leq N \leq t - f$ .

Figura 4.4 – Predição sem *ranking*, apenas a uma funcionalidade é recomendada

Fonte: Próprio autor.

Na Figura 4.4 é possível observar como é feita a predição quando a saída do algoritmo de recomendação consiste de apenas um item classificado como o mais provável. No exemplo da figura, para ser feita a predição da próxima funcionalidade  $f_t$ , são levadas em consideração todas as funcionalidades anteriores  $f_1, \dots, f_{t-1}$ , de modo que para que a recomendação seja correta, a saída  $d$  do algoritmo deve ser igual a  $f_x$  desejada pelo usuário.

Figura 4.5 – Predição com *ranking*, uma lista *top-5* é recomendada

Fonte: Próprio autor.

Diferente da Figura 4.4, na Figura 4.5 a saída do algoritmo de recomendação consiste de uma lista ranqueada de  $N$  itens classificados como mais relevantes ao usuário. Assim como o exemplo anterior, para ser feita a predição da próxima funcionalidade  $f_t$ , são levadas em consideração todas as funcionalidades anteriores  $f_1, \dots, f_{t-1}$ , de modo que para que a recomendação seja correta, basta que o item realmente relevante para o usuário esteja presente na saída  $r$  do algoritmo.

Essa estratégia de listas com  $N$  itens mais relevantes é bastante utilizada em diversos outros contextos de sistemas de recomendação (ex.: *e-commerce*, redes sociais, mecanismos de pesquisa e etc), porém, a maior diferença entre esses contextos se dá pelo fato de que no problema de recomendação de próxima funcionalidade apenas um item dentre todos os demais pode ser escolhido, enquanto que nos demais contextos o

usuário pode escolher mais de um item da lista. De maneira geral, o objetivo de se utilizar essa abordagem ao recomendar uma próxima funcionalidade é aumentar as chances de recomendar o item correto.

Vale ainda ressaltar que, quando temos uma lista *top-1*, ela é equivalente ao processo de predição normal, enquanto que uma lista *top-N* de tamanho igual ou próximo a  $t$  não tem muito sentido, visto que é equivalente a apresentar todos os itens ao usuário e deixa-lo escolher o correto, tornando inútil o processo de recomendação. Em resumo, o valor de  $N$  é um hiperparâmetro e deve ser ajustado de modo a apresentar de maneira sucinta os itens possivelmente mais relevantes ao usuário.

#### 4.1.3 Categorização do sistema de recomendação de próxima funcionalidade

A partir das definições, cenários e domínios abordados na Seção 3.2, pode-se caracterizar o problema da recomendação de próxima funcionalidade em softwares estudado nesse trabalho como um problema de recomendação sensível à sequência, haja visto os seguintes aspectos:

**Entrada:** os dados de entrada consistem de sequências de interações do usuário com cada funcionalidade do sistema, ordenadas pelo momento de acesso e sendo cada interação caracterizada pelo tipo de ação que o usuário realizou. Por exemplo, a ação de abrir a tela de certa funcionalidade é descrita como uma ação de *open*, enquanto que a ação de selecionar uma tela que já está aberta é descrita como *activate* e por fim, a ação de fechar a tela de uma funcionalidade é descrita como *close*.

**Saída:** a saída do sistema de recomendação de próxima funcionalidade consiste de uma lista de  $n$ -possíveis próximas funcionalidades que o usuário gostaria de acessar, ordenada da mais provável a menos provável, e sendo permitido a escolha de apenas um dos itens da lista.

**Tarefas computacionais:** em resumo, o sistema de recomendação de próxima funcionalidade deve gerar as recomendações a partir do histórico de sequências de interações do usuário. Para alcançar tal objetivo, algumas tarefas computacionais relativas aos sistemas de recomendação sensível à sequência são necessárias, tais como:

- **Adaptação de contexto:** o problema de recomendação da próxima funcionalidade está diretamente ligado a adaptação de contexto, visto que o processo de treinamento e predição dos modelos de recomendação são feitos a partir de uma combinação de duas situações de adaptação ao contexto, com o processo de treinamento utilizando os dados de todas as sessões passadas (recomendação sensível a sessão) porém limitando o tamanho de cada sequência às  $n$ -últimas iterações, enquanto que o processo de predição

toma como contexto somente as  $n$ -últimas iterações do usuário na sessão atual.

- **Recomendação repetida:** é comum no problema de recomendação de próxima funcionalidade a repetição de recomendação de itens já recomendados ao usuário, visto que acessar a mesma funcionalidade várias vezes é algo intrínseco ao domínio do problema. É justamente pela questão da repetição que surge a necessidade de recomendar as funcionalidade mais relevantes ao usuário, evitando assim a necessidade de muitos cliques para se realizar uma tarefa repetitiva. Esse processo é semelhante ao que ocorre em Lu, Lin e Ciou (2014), Natarajan, Shin e Dhillon (2013), que tenta prever qual o próximo aplicativo que o usuário irá acessar baseando-se nos aplicativos que ele abriu recentemente.

A partir dessas características, dentre outras, é possível classificar o problema de recomendação de próxima funcionalidade como um problema de recomendação sensível a sequência. Essa classificação é importante tanto para efeito de conhecimento de métricas e algoritmos mais utilizados em problemas do tipo, como para comparação da metodologia desse trabalho com problemas semelhantes, visto que até o presente momento não foi encontrado na literatura nenhum trabalho que abordasse o problema de recomendação de próxima funcionalidade em softwares.

## 4.2 Definição formal

Matematicamente, o problema de interesse consiste na predição da funcionalidade  $f_t \in \{1, \dots, M_f\}$  no instante de tempo  $t$  a partir das funcionalidades anteriormente observadas  $f_{t-1}, f_{t-2}, \dots, f_1$  para um determinado usuário  $u$ .

Assumindo que  $F_t$  representa uma VA discreta que denota o frame no instante  $t$  e que  $F_t$  assume um valor qualquer no conjunto  $\mathcal{F} = \{1, \dots, M_f\}$ , em que  $M_f$  é o número total de funcionalidades. De modo similar, assumindo que  $U$  é uma VA representando usuários e assumindo valores em  $\mathcal{U} = \{1, \dots, M_u\}$ , em que  $M_u$  é o número total de usuários. Considerando ainda que os softwares/sistemas são representados por uma VA  $S$ , onde cada atribuição consiste em uma valor em  $\mathcal{S} = \{1, \dots, M_s\}$ , em que  $M_s$  é o número total de sistemas e, de forma semelhante, as empresas são representadas por uma VA  $E$  que assume valores em  $\mathcal{E} = \{1, \dots, M_e\}$ , sendo  $M_e$  o número total de empresas.

Está-se interessado em encontrar  $N$  recomendações, definidas por

$$\begin{aligned}\hat{f}_t^{(1)} &= \arg \max_{f_t \in \mathcal{F}} P(F_t = f_t \mid F_{t-1} = f_{t-1}, \dots, F_1 = f_1, U = u, S = s, E = e), \\ \hat{f}_t^{(2)} &= \arg \max_{f_t \in \mathcal{F} \setminus \{\hat{f}_t^{(1)}\}} P(F_t = f_t \mid F_{t-1} = f_{t-1}, \dots, F_1 = f_1, U = u, S = s, E = e), \\ &\vdots \\ \hat{f}_t^{(N)} &= \arg \max_{f_t \in \mathcal{F} \setminus \{\hat{f}_t^{(1)}, \dots, \hat{f}_t^{(N-1)}\}} P(F_t = f_t \mid F_{t-1} = f_{t-1}, \dots, F_1 = f_1, U = u, S = s, E = e).\end{aligned}$$

Vale ressaltar que o valor  $N$  é uma parâmetro que pode variar. Ele representa o tamanho do *ranking* de recomendações. O valor  $\hat{f}_t^{(1)}$  representa a primeira escolha mais provável para a tela no instante  $t$ ; de modo similar,  $\hat{f}_t^{(N)}$  denota a  $N$ -ésima opção no *ranking* de recomendações para a tela no instante  $t$ .

O termo  $P(F_t = f_t \mid F_{t-1} = f_{t-1}, \dots, F_1 = f_1, U = u, S = s, E = e)$ , ou simplesmente  $p(f_t \mid f_{t-1}, \dots, f_1, u, s, e)$ , denota a probabilidade da funcionalidade no instante  $t$  dado os valores das funcionalidades em todos os instantes anteriores, o usuário, o sistema e a empresa. Essa especificação a nível de usuário consiste na recomendação mais granular possível. A partir do cálculo de probabilidades no nível mais granular, pode-se então encontrar as probabilidades para quaisquer nível de especificação (seja do sistema ou empresa), em que os valores das variáveis que não são de interesses são eliminadas através de operações de somas. Como exemplo, caso esteja-se interessado em realizar a predição especificando somente o sistema  $S$  e a empresa  $E$ , a predição  $\hat{f}_t^{(1)}$  seria dada por

$$\begin{aligned}\hat{f}_t^{(1)} &= \arg \max_{f_t \in \mathcal{F}} p(f_t \mid f_{t-1}, \dots, f_1, s, e), \\ &= \arg \max_{f_t \in \mathcal{F}} \sum_{u \in \mathcal{U}} p(f_t, u \mid f_{t-1}, \dots, f_1, s, e) \\ &= \arg \max_{f_t \in \mathcal{F}} \sum_{u \in \mathcal{U}} p(f_t \mid f_{t-1}, \dots, f_1, u, s, e) p(u \mid f_{t-1}, \dots, f_1, s, e).\end{aligned}$$

## 4.3 Conclusão

Neste capítulo, é descrito a formulação do problema de recomendação de próxima funcionalidade de softwares, tanto matematicamente como de forma ilustrada. É apresentado também uma forma de criar modelos mais genéricos a partir das sequências dos usuários, agrupando essas sequências em níveis mais abrangentes do que somente a nível de usuário, de modo que a experiência de vários usuários consegue ser agregada em um único modelo. Além disso, é explicado as diferenças entre os dois tipos de predição de próxima funcionalidade considerados: através de uma única funcionalidade ou através de uma lista de funcionalidades mais prováveis. A partir do conhecimento descrito neste capítulo, é possível entender os cenários de testes desenvolvido para os experimentos, bem como as implicações dos resultados neles obtidos.



## 5 Experimentos e Resultados

Neste capítulo são apresentadas a base de dados e a metodologia utilizada nos experimentos, bem como uma discussão sobre os resultados obtidos para as soluções propostas neste trabalho, com o objetivo de solucionar o problema de recomendação de próxima funcionalidade em softwares, descrito no Capítulo 4. Para tanto, foram utilizados algoritmos estatísticos, como as cadeias de markov (seção 2.2.2) de primeira e segunda ordem, e redes neurais recorrentes (seção 2.3.4.1), tais como LSTM (seção 2.3.5) e IndRNN (seção 2.3.6).

Os experimentos foram realizados em um ambiente configurado com Python 3.6 e utilizando as seguintes bibliotecas: Keras 2.2.2 (CHOLLET et al., 2015) com TensorFlow 1.15.0 (ABADI et al., 2015) como *backend*, Numpy 1.14.4, e Sklearn 0.20.1. Além disso, é importante destacar que os experimentos foram executados utilizando-se uma configuração computacional com um Intel Core i3-9100F e 16 GB de RAM DDR4.

### 5.1 Base de dados

O conjunto de dados utilizado é composto pelas interações dos usuários com as funcionalidades de seis sistemas de contabilidade desenvolvidos pela Fortes Tecnologia. Nesse contexto, o termo funcionalidade denota cada janela do sistema que possui uma funcionalidade específica. Assim, cada vez que um funcionário sai de uma janela (ou funcionalidade) para outra, essa transição é, juntamente com alguns metadados, armazenada de modo que, quando a sessão do usuário é encerrada, o conjunto de transições salvas formam uma sequência que representa o fluxo de trabalho desse usuário. É a partir dessas sequências que os algoritmos irão aprender a reconhecer e recomendar as funcionalidades que o usuário gostaria de acessar de acordo com seu fluxo de trabalho.

A extração das sequências dos dados armazenados é feito a partir de um pré-processamento que ordena cada transição com base no *timestamp* dessa transição. Além disso, o início e fim de cada sequência é definido com base no período de tempo que o usuário permanece com uma sessão ativa no sistema, de modo que quando uma sessão é encerrada, a última funcionalidade acessada é também o último item da sequência. As vantagens de definir o início e fim das sequências com base na sessão é que uma sessão consegue, na maioria das vezes, limitar bem um fluxo de trabalho que faça sentido. A desvantagem é que isso também permite algumas sequências pequenas e muitas vezes incompletas, já que a sessão pode ser encerrada por inatividade.

O problema de sequências pequenas é que não é possível inferir com certeza qual

tarefa esse usuário estaria realizando, visto que muitas tarefas compartilham subsequências semelhantes. Já o problema com o encerramento da sessão por inatividade é que, como a sequência é encerrada incompleta, os algoritmos não irão associar essas sequências as tarefas que os usuários estavam realizando. Devido à esses problemas, optou-se por considerar apenas sequências com um tamanho mínimo de 6 transições.

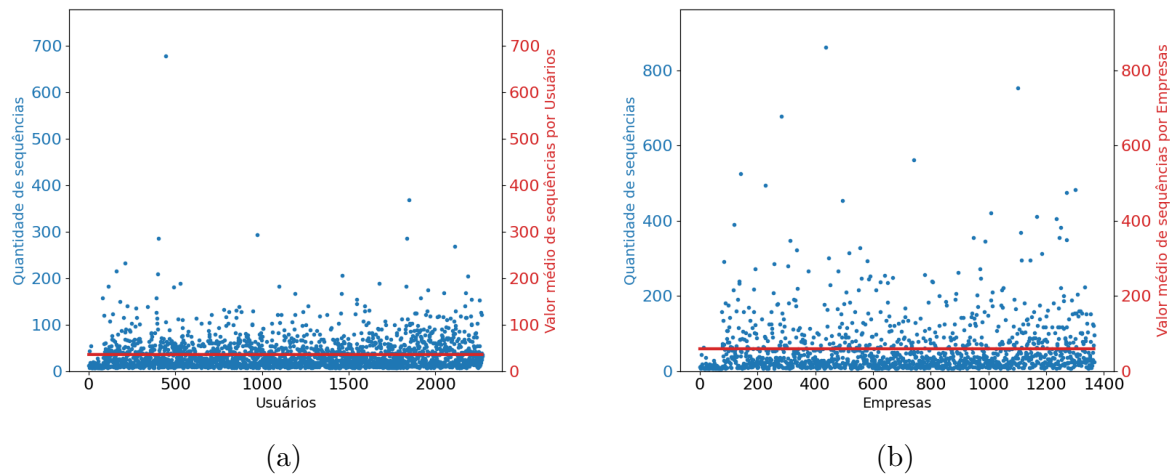
Para realização dos experimentos, foram utilizados dados de um período de testes de aproximadamente 3 semanas, na qual os usuários de 6 sistemas de contabilidade tiveram suas interações registradas e pré-processadas conforme descrito anteriormente. Esse pré-processamento resultou em uma base de dados com 81211 sequências, distribuídas entre 2270 usuários pertencentes a 1366 empresas contratantes desses sistemas. Isso pode ser melhor observado na Tabela 5.1 juntamente com a quantidade média e mediana de sequências por usuários, empresas e sistemas.

Tabela 5.1 – Quantidade de sequências por usuários, empresas e sistemas na base de dados.

|                 | <b>Total</b> | <b>Média (seq)</b> | <b>Mediana (seq)</b> | <b>Desv. Padrão</b> |
|-----------------|--------------|--------------------|----------------------|---------------------|
| <b>Usuários</b> | 2270         | 35,77              | 25                   | 35,22               |
| <b>Empresas</b> | 1366         | 59,45              | 32                   | 76,72               |
| <b>Sistemas</b> | 6            | 13535,16           | 215,5                | 21897,05            |

Através da Tabela 5.1, é possível verificar que quanto mais se generaliza as sequências, maior é o desvio padrão e mais acentuada é a diferença entre a média e a mediana da quantidade média de sequências. De maneira geral, esse efeito ocorre tanto devido a diferença entre a quantidade de funcionalidades acessadas por cada usuário como também, no caso de empresas e sistemas, devido a quantidade de usuários que cada empresa e sistema possuem, sendo essa diferença consideravelmente maior para o caso dos sistemas. A partir das Figuras 5.1a e 5.1b é possível notar visualmente a diferença entre a relação da quantidade e a média de itens por usuário e empresa.

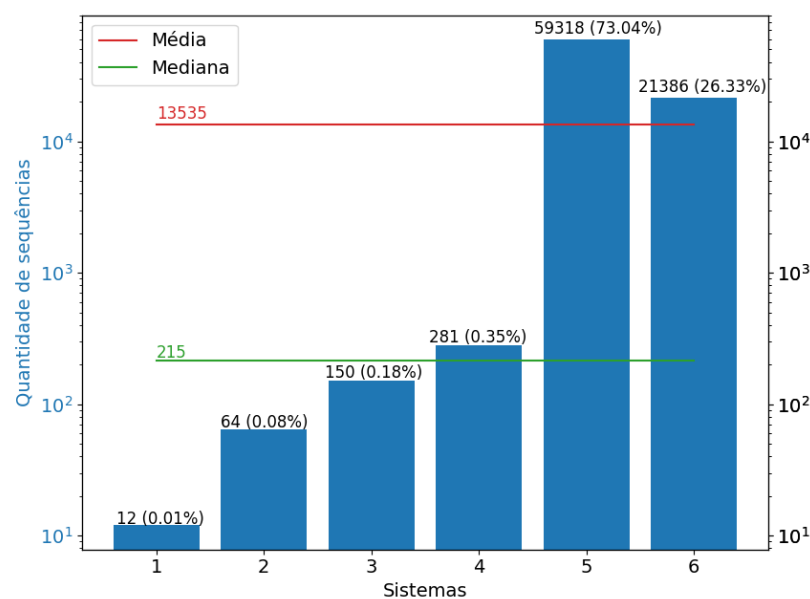
Figura 5.1 – (a) Distribuição da quantidade de sequências por usuários; (b) Distribuição da quantidade de sequências por empresas.



Fonte: Próprio autor.

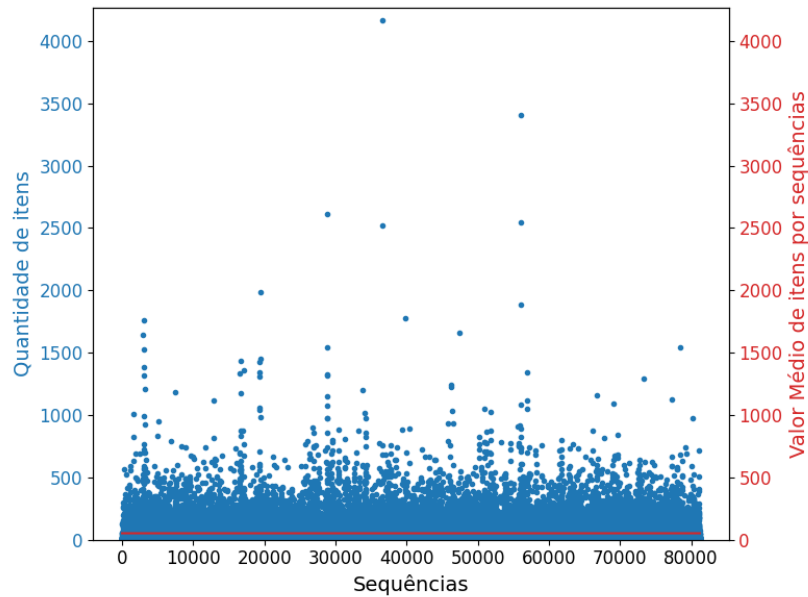
Com relação aos sistemas, essa diferença se dá consideravelmente maior pelo fato de o sistema representado pelo número 5 na Figura conter 73,04% de todas as sequências, enquanto que o sistema 6 contém 21,38% das sequências. Desse modo, 99,37% de todas as sequências da base de dados ocorreram utilizando-se os sistemas 5 e 6.

Figura 5.2 – Quantidade de sequências por sistemas



Fonte: Próprio autor.

Figura 5.3 – Quantidade de itens por sequências. Média em vermelho



Fonte: Próprio autor.

Além disso, tem-se que a quantidade média de itens por sequência é de 52, com um desvio padrão de 85,95 e uma mediana de 22. Essas variações podem ser melhor entendida a partir da Figura 5.3, na qual pode ser visto a relação de quantidade de itens por sequência, com a média de itens por sequência representado em vermelho.

## 5.2 Metodologia

A metodologia empregada neste trabalho teve como objetivo principal avaliar o desempenho dos algoritmos em recomendar as próximas funcionalidades para os usuários de determinados sistemas baseados no histórico de uso, levando em consideração a descrição do problema dado no Capítulo 4.

Cada histórico é representado na forma de um **conjunto de sequências**, onde cada item dessas sequências representa uma funcionalidade. Além disso, como visto na seção 4.1.1, é possível generalizar as sequências com base em alguns atributos, influenciando na forma como os conjuntos de sequências são definidos. Dessa forma, foram estabelecidos três cenários de testes que levam em conta esses atributos, sendo eles:

- **usuários:** Neste cenário, apenas o histórico de uso do próprio usuário é considerado relevante para se fazer as recomendações.
- **empresa:** Já neste cenário, para realizar as recomendações de próximas funcionalidades a determinado usuário de uma empresa, o histórico de uso de todos os usuários dessa empresa é relevante.

- **sistema:** De forma semelhante ao cenário da empresa, aqui o histórico de outros usuários é considerado ao gerar as recomendações para determinado usuário, porém, nesse caso, o histórico de todos os usuários desse sistema é utilizado, independentemente da empresa ou qualquer outro atributo, sendo esse o cenário macro.

Assim, em cada um desses cenários, os conjuntos das sequências são agrupados de forma distinta. Para o cenário onde apenas os dados do próprio usuário são relevantes, cada conjunto contém apenas as sequências de um usuário. Já para o cenário empresa, todas as sequências dos usuários de determinada empresa são agrupadas em um mesmo conjunto. Semelhantemente, para o cenário do sistema, todas as sequências dos usuários de um sistema são agrupadas em um único conjunto. O objetivo desses dois últimos cenários (empresa e sistema) é verificar se a experiência de usuários diferentes, porém com uma característica em comum, pode ajudar na recomendação de outros usuário. Ou seja, verificar o quanto a filtragem colaborativa é eficiente no problema de recomendação de próxima funcionalidade.

Cada experimento é separado em duas etapas, sendo a primeira etapa chamada **etapa de treinamento**, onde parte do conjunto de sequências de cada usuário/empresa/sistema é separado para ser utilizado como entrada do algoritmo de aprendizado e gerar um modelo recomendador. A segunda etapa é chamada **etapa de teste**. É nessa etapa que avalia-se o quão preciso é o modelo gerado pelo algoritmo, para isso, utiliza-se o restante das sequências do conjunto que não foram utilizados na etapa de treinamento. Tendo em vista isso, optou-se por dividir o conjunto de sequências em 80% das sequências para a etapa de treinamento e os 20% restante para a etapa de testes. Vale ressaltar que esse procedimento é feito para cada cenário.

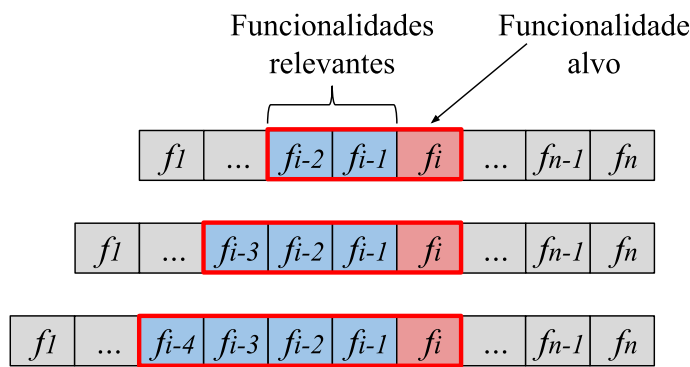
A forma como os algoritmos geram o modelo recomendador é próprio de cada um, porém a entrada desses algoritmos é a mesma: uma sequência de valores que representam a ordem na qual os usuários acessaram cada funcionalidade. Note que esses valores são codificações *one-hot* que representam cada tela. Como já detalhado no capítulo 4, a forma como os algoritmos aprendem a prever uma funcionalidade se dá a partir da observação das funcionalidades anteriores a ela.

Ou seja, dada uma sequência  $f_1, \dots, f_i \dots f_n$ , se o objetivo é aprender a recomendar a funcionalidade  $f_i$  corretamente, então deve se levar em consideração as funcionalidades anteriores  $f_{i-1}, f_{i-2}, \dots$ . Porém, como as sequências não possuem um tamanho fixo e podem ser excessivamente compridas, é necessários estabelecer um limite à quantidade de funcionalidades anteriores que o algoritmo poderá observar. Isso evita problemas de perda de informações correlacionadas devido a grandes sequências e número excessivo de parâmetros, conforme explicado no Capítulo 2.

Tendo em vista isso, cada sequência que será utilizada no treinamento é pré-

processada em subsequências menores para então serem usadas como entrada dos algoritmos. O tamanho dessas sequências depende de quantas funcionalidades anteriores serão observadas para aprender a recomendar funcionalidade desejada. Por exemplo, caso a quantidade de funcionalidades observadas seja 2, o tamanho da subsequência será de 3, pois ela é composta do histórico relevante e a própria funcionalidade desejada. A partir da Figura 5.4 é possível entender melhor como a quantidade de funcionalidades anteriores relevantes (destacadas em azul) podem variar para uma mesma sequência. Além disso, o tamanho total da subsequência é demarcado em vermelho.

Figura 5.4 – Variação na quantidade de funcionalidades relevantes

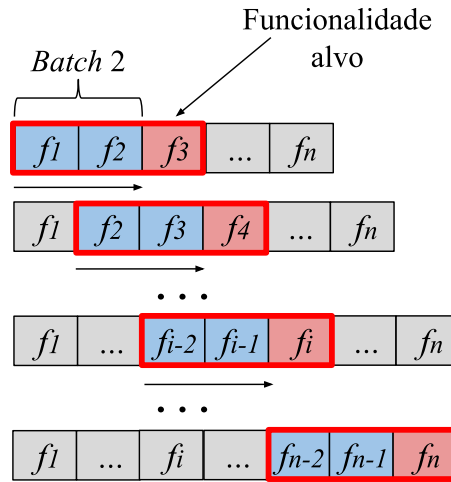


Fonte: Próprio autor.

Essa quantidade de funcionalidades anteriores necessárias para prever a funcionalidade desejada é também chamado de “janela” ou *batch*. Note que, ao definir tamanho total das subsequências, todas as sequências posteriores que o algoritmo irá receber como entrada deve possuir esse mesmo tamanho e formato. Já para a entrada do modelo gerado, as sequências devem ter apenas o tamanho do *batch* utilizado no treinamento, já que a saída do modelo gerado é justamente a funcionalidade desejada.

Já a forma como cada subsequência de treinamento é gerada é detalhada na Figura 5.5, onde é possível observar como uma janela desliza sobre a sequência de funcionalidades, gerando as subsequências de acordo com o tamanho predefinido. Além disso, é possível observar que em azul são destacadas as funcionalidades relevantes para prever a funcionalidade que deseja-se aprender a recomendar corretamente (destacada em vermelho claro), e com a quantidade de funcionalidades relevantes limitada ao tamanho do *batch*. Um processo semelhante é feito para as subsequências de teste, porém, lembrando que nesse caso, as subsequências são do tamanho do *batch*, enquanto que a funcionalidade alvo deve ser predita pelo modelo gerado. Esse procedimento nos permite treinar e testar os algoritmos como um problema de classificação, onde as funcionalidades relevantes se equiparam às características da amostra e funcionalidade alvo à classe. A diferença para um problema de classificação comum é que, nesse contexto, são geradas centenas de classes.

Figura 5.5 – Processo de geração das subsequências



Fonte: Próprio autor.

Além disso, cada algoritmo tem uma capacidade máxima de processamento dessas subsequências. Por exemplo, para as cadeias de markov, esse valor está diretamente ligado a ordem da cadeia, isso é, a cadeia de markov de ordem 1 considera apenas a última funcionalidade anterior como relevante, enquanto que a cadeia de markov de ordem 2 considera as duas últimas funcionalidades anteriores. Já as redes neurais recorrentes possuem uma capacidade maior em processar grandes sequências, porém neste trabalho, devido as sequências não serem excessivamente grandes, foram avaliados *batches* de tamanho 2, 3, 4 e 5 para essas redes.

Outro parâmetro levado em consideração nos experimentos foi o tamanho do *rank* utilizado para as recomendações geradas. Isso nos permitiu compreender o quanto uma lista de recomendações pode ser melhor que apenas recomendar uma funcionalidade.

## 5.3 Métricas

As performances dos algoritmos foram avaliadas a partir de três métricas: Acurácia (seção 3.3.2.1), ARHR (seção 3.3.2.3) e NDCG (seção 3.3.2.4). Essas duas últimas métricas foram utilizadas para medir a qualidade do *rank*. Para os contextos em que foi gerada uma lista de recomendação, o cálculo da acurácia foi feito de modo que, um acerto ocorre sempre que a próxima funcionalidade correta encontra-se entre as *top-N* recomendações, independente de sua posição. Além disso, considerando o uso comercial desta aplicação, o tempo de treinamento e predição foram métricas importantes, devido ao fato de que um tempo longo de treinamento requer mais recursos computacionais, enquanto uma predição demorada pode acabar não auxiliando o usuário no momento correto.

## 5.4 Resultados

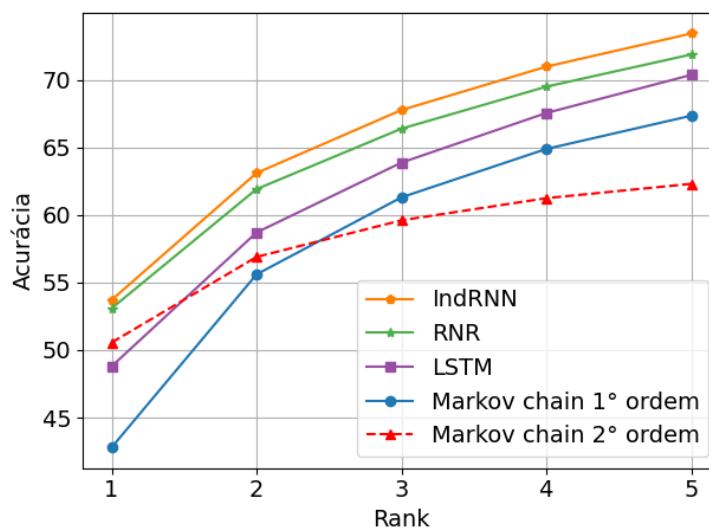
Nesta seção são apresentados os resultados dos experimentos conduzidos com o objetivo de avaliar as soluções propostas para o problema de recomendação de próxima funcionalidade. Como já descrito nas seções anteriores, os experimentos foram realizados em três cenários diferentes, no qual em cada cenário a base de dados foi interpretada de um modo diferente no que diz respeito ao agrupamento das sequências em determinados conjuntos. Dessa forma, as subseções a seguir descrevem detalhadamente cada cenário e ao final é apresentado um comparativo geral entre os resultados de cada cenário.

### 5.4.1 Usuário

Neste cenário, as sequências são agrupadas por usuário, ou seja, o conjunto de sequências associado a cada usuário é o seu próprio histórico de uso. Este é o cenário mais básico, pois, se queremos recomendar algo que o usuário goste, basta olhar o que ele costuma usar. Dessa forma, cada algoritmo de aprendizado foi treinado e testado individualmente para cada usuário. Em outras palavras, para cada usuário, foi gerado um modelo de cada algoritmo e testado nesse mesmo usuário.

O objetivo principal desses experimentos foi verificar a precisão média das recomendações para cada usuário. Além disso, para se investigar as consequências de se usar uma lista de recomendações ao invés de uma simples recomendação, foram geradas lista de recomendações que continham de uma até cinco recomendações (as listas de tamanho 1 equivalem a uma simples recomendação), sendo possível observar a precisão das recomendações na Figura 5.6.

Figura 5.6 – Acurácia por tamanho do *rank* da lista de recomendações

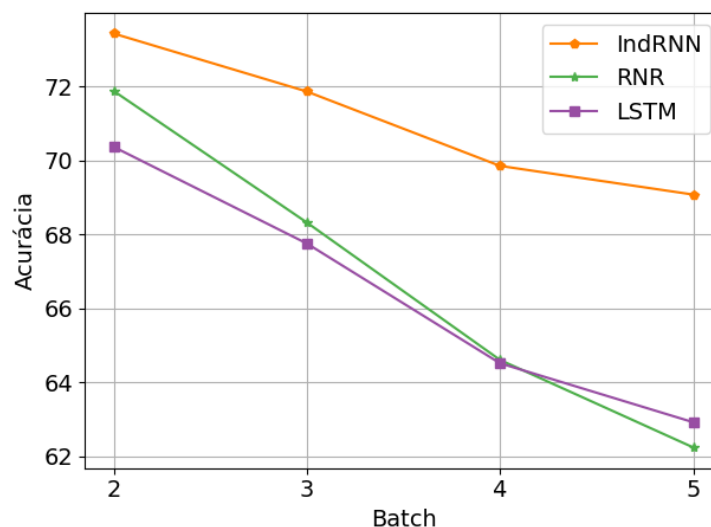


Fonte: Próprio autor.

Neste experimento optou-se por treinar as redes neurais recorrentes com *batch* 2 por se assemelharem mais as cadeias de markov no que diz respeito a quantidade de informações utilizadas para gerar as predições. A partir do gráfico, é possível observar que dois modelos que possuíam uma acurácia inferior a 50%, a partir de uma lista com duas recomendações já passam a ter uma precisão acima de 55%. Além disso, há uma melhora expressiva em todos os algoritmos, especialmente nas redes neurais recorrentes, com acurácia acima de 70% para listas de recomendação de tamanho 5. Também é possível notar que, enquanto que para listas com apenas uma recomendação a cadeia de markov de 2º ordem é mais precisa que a cadeia de markov de 1º ordem, ao aumentar-se a quantidade de itens recomendados nas listas de recomendação, a cadeia de 1º ordem apresenta acurácias maiores.

Esse é um comportamento um tanto contraintuitivo, pois, ao aumentar a quantidade de informações observadas no treinamento espera-se que sejam encontrados contextos mais particulares, o que por sua vez permitiria o reconhecimento de tarefas com um fluxo de telas mais específico. Porém, o que se observa é que o algoritmo mais genérico consegue acertar mais qual a próxima funcionalidade que será acessada pelo usuário. Esse mesmo comportamento foi observado ao se aumentar o *batch* das redes neurais recorrente, como pode ser observado na Figura 5.7. No gráfico, é possível observar o quanto as acurácias caem ao se levar em consideração uma sequência maior de funcionalidades passadas para tentar prever a próxima.

Figura 5.7 – Redes neurais recorrentes com tamanhos diferentes de *batch*

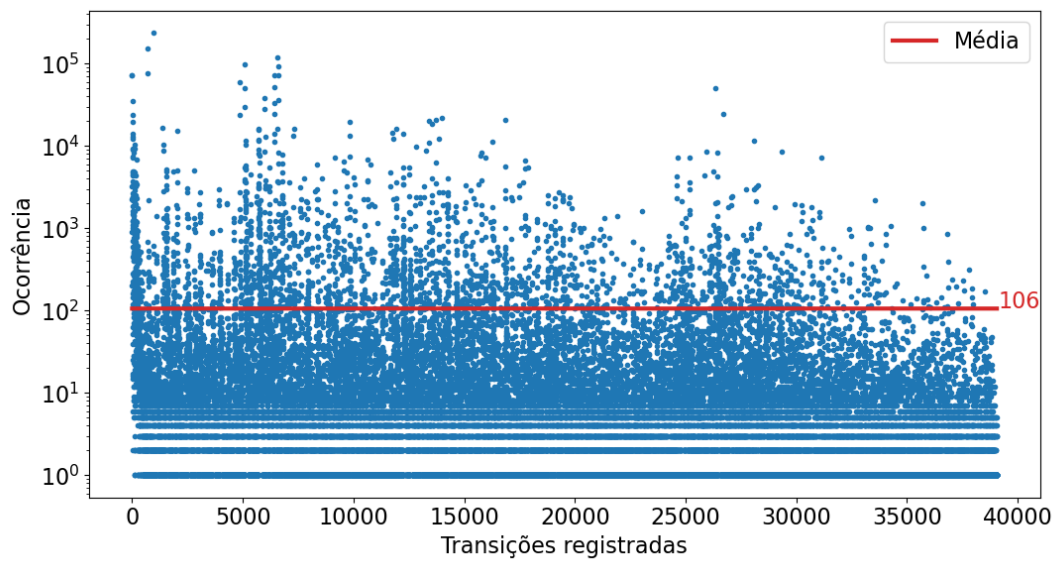


Fonte: Próprio autor.

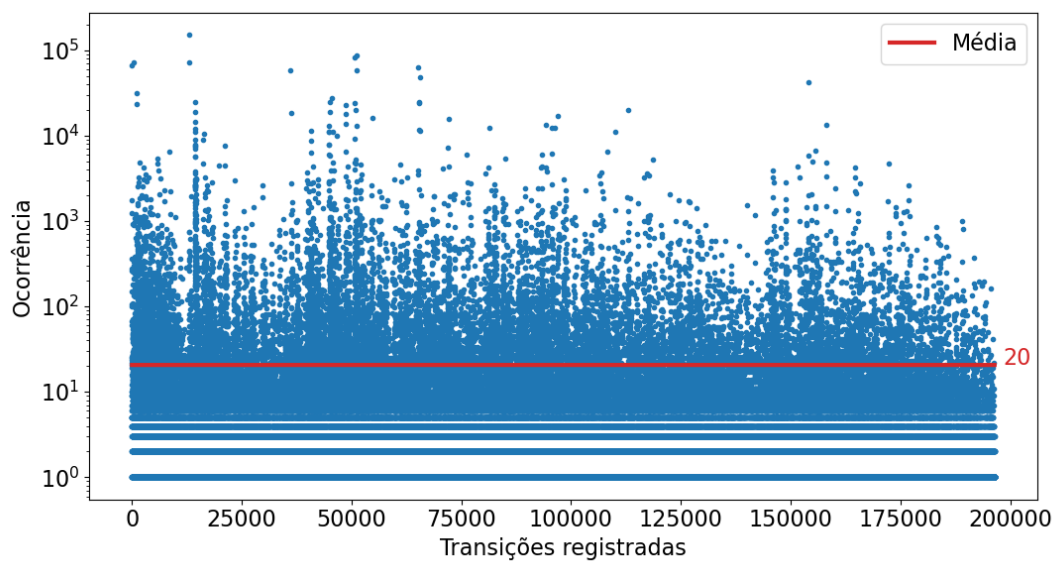
Isso pode ocorrer devido ao fato de haver pares de funcionalidades consecutivas muito frequentes, ou seja, algumas funcionalidades são mais acessadas após certas funci-

onalidades. Esse comportamento faz com que essas transições sejam mais recorrente do que as demais e, conseqüentemente, acabam elevando a acurácia das predições nos testes. Esse comportamento pode ser visualizado na Figura 5.8a, onde é possível notar que há transições de funcionalidades que, na base de dados, ocorreram mais de 50 mil vezes, com uma média de 106 ocorrências por transição. Ao aumentar o tamanho do *batch*, essas transições são diluídas em transições cada vez mais particulares e, conseqüentemente com menos ocorrências, como pode ser observado na Figura 5.8b para transições de *batch* 2.

Figura 5.8 – (a) Quantidade de ocorrências por transições de *batch* 1 na base de dados;  
(b) Quantidade de ocorrências por transições de *batch* 2 na base de dados.



(a)



(b)

Fonte: Próprio autor.

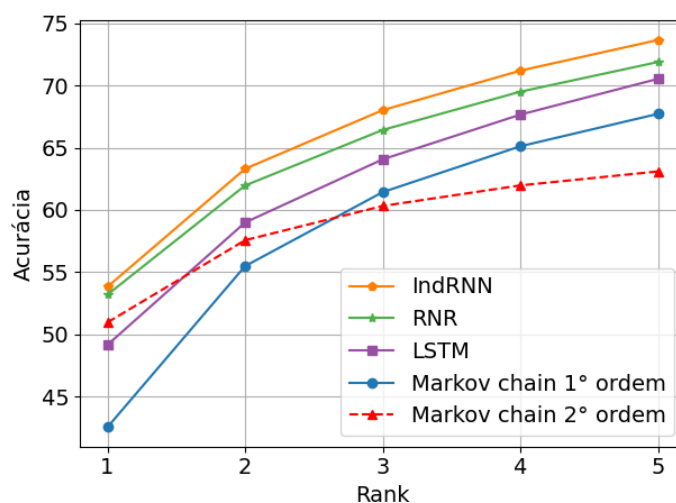
A partir das figuras 5.8a e 5.8b, é possível notar que enquanto a quantidade média de ocorrências por transição de *batch* 2 diminui 5x em relação às transições com *batch* 1, a quantidade total de transições registradas aumentaram 5x, o que reforça o argumento de que transições mais genéricas com um número elevado de ocorrências foram diluídas em transições mais específicas e com menos ocorrências, afetando assim as acurácias dos modelos nos testes.

Com isso, pode-se concluir que, para esse cenário, as redes neurais recorrentes obtiveram um melhor desempenho sobre as cadeias de markov, com destaque para as redes IndRNN e RNR padrão, que possuem uma arquitetura mais simples, porém se mantiveram acima das demais redes em todos os experimentos. Além disso, é possível concluir também que modelos treinados com transições menores, isso é, com um tamanho de *batch* menor, se adequam melhor ao problema de recomendação de próxima funcionalidade em ambientes com características semelhantes ao dos sistemas que compõem a base de dados utilizada.

### 5.4.2 Empresa

Pode-se dizer que a diferença entre este cenário e o anterior é basicamente conceitual, pois, ao agrupar em um só conjunto as sequências dos usuários da empresa, é como se estivesse sendo criado um outro usuário, porém com uma diferença importante o suficiente para se considerar esse contexto como um cenário válido de teste: a experiência de todos os usuários da empresa em um só modelo. De forma semelhante ao cenário anterior, para se verificar o quanto o tamanho do *rank* na lista de recomendações ajuda nas predições, as redes neurais foram treinadas com sequências de *batch* de tamanho 2. Os resultados podem ser observados na Figura 5.9.

Figura 5.9 – Acurácia por tamanho do *rank* da lista de recomendações

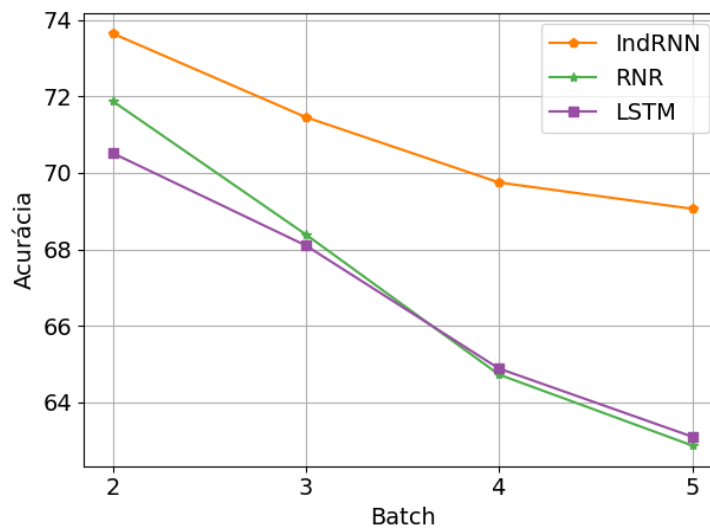


Fonte: Próprio autor.

Comparado aos resultados do cenário anterior, houve um pequeno aumento na acurácia de todos os algoritmos, porém um aumento não tão expressivo. Entretanto, se for levado em consideração que, se dividido pela quantidade de empresas na base de dados (Tabela 5.1), o número de usuários por empresa é inferior a 2, é perceptível o motivo dessa diferença não ser tão expressiva.

Além disso, a partir da Figura 5.10 nota-se que aumentando o tamanho do *batch* das sequências utilizadas no treinamento das redes neurais recorrente, é observado um comportamento semelhante ao cenário anterior, porém com uma acurácia um pouco maior para todos os algoritmos.

Figura 5.10 – Redes neurais recorrentes com tamanhos diferentes de *batch*



Fonte: Próprio autor.

Dessa forma, apesar de dar indícios de uma melhora, não se pode concluir com certeza o quanto a utilização da experiência compartilhada de todos os usuários de uma empresa ajuda na predição das próximas funcionalidades para os usuários dessa empresa.

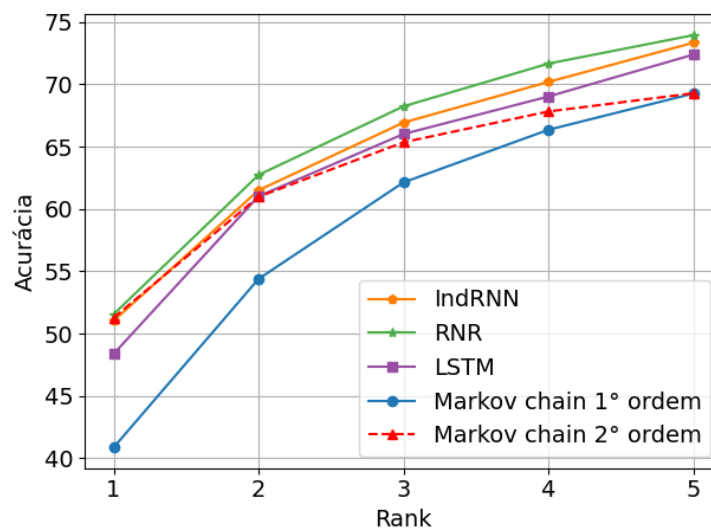
### 5.4.3 Sistemas

Assim como o cenário que considera as sequências agrupadas por empresa, o objetivo nos experimentos deste cenário foi investigar o quanto compartilhar as experiências dos usuários pode melhorar o desempenho do sistema de recomendação de próxima funcionalidade. Por ser o cenário mais universal, que compreende em um único modelo as experiências de milhares de usuários, este foi o cenário no qual foram gerados os maiores modelos. Além disso, diferente do cenário da empresa, que não contém informações suficientes na base de dados, neste cenário foi possível comparar os resultados de sistemas

com grandes quantidade de usuário a sistemas com poucos usuários, conforme exposto na Figura 5.2.

Os resultados dos experimentos para o cenário do sistema podem ser observados na Figura 5.11, onde é possível observar novamente um incremento na acurácia dos modelos gerados. Além disso, diferente dos dois cenários anteriores, os modelo da cadeia de markov de 2° ordem obtiveram uma precisão maior para todos os tamanhos de *rank* em relação as cadeias de markov de 1° ordem. Outra diferença neste cenário está no fato de que a RNR padrão obteve resultados melhores que a IndRNN. Apesar de não ser uma diferença grande, ela se manteve para todos os tamanhos de *rank*.

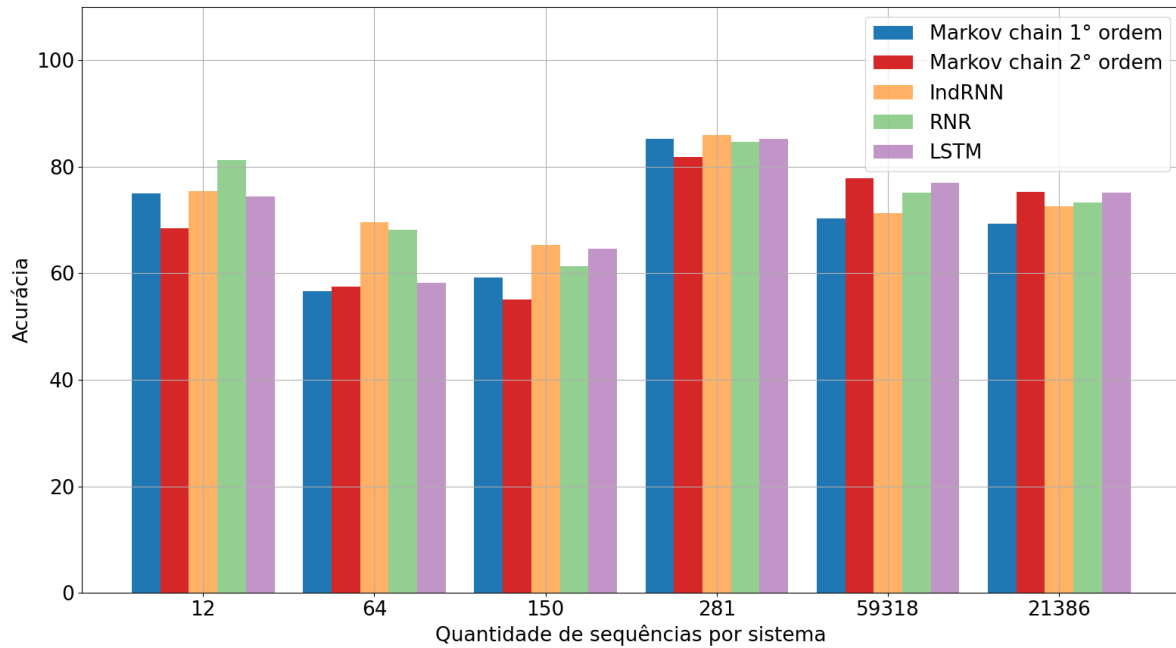
Figura 5.11 – Acurácia por tamanho do *rank* da lista de recomendações



Fonte: Próprio autor.

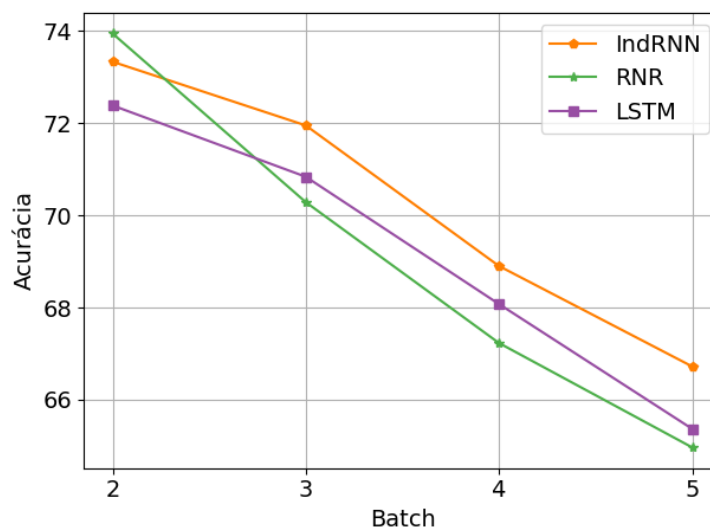
Devido ao fato dos sistemas presentes na base de dados possuírem tamanhos diferente, estes tornam-se uma boa ferramenta para avaliar o quanto a quantidade de sequências disponíveis para os algoritmos no treinamento influenciam na acurácia dos modelos gerados. Diante disto, a Figura 5.12 trás um comparativo da acurácia dos modelos em cada sistema, além da quantidade de sequências que compõem esses sistemas. É interessante notar como o modelo da cadeia de markov de 2° ordem se sobressai perante os demais modelos nos dois maiores sistemas, enquanto que nos sistemas com poucas sequências ele possui as menores acurácias. Diante disto, é possível concluir que, a baixa acurácia da cadeia dos modelos da cadeia de markov de 2° ordem para os cenários anteriores, se dá também devido a baixa quantidade de dados disponíveis para cada usuário/empresa. Porém, vale salientar que esse problema consegue ser contornado pelos modelos das redes neurais recorrentes.

Figura 5.12 – Acurácia por quantidade de sequências para cada sistema



Fonte: Próprio autor.

Com relação ao tamanho do *batch* para as redes neurais recorrentes, tem-se que, embora a RNR tenha obtido o melhor desempenho dentre os algoritmos, ao aumentar-se o tamanho do *batch* sua acurácia fica abaixo das demais redes (Figura 5.13). Já comparado aos cenários anteriores, os resultados de todas as redes foram melhores.

Figura 5.13 – Redes neurais recorrentes com tamanhos diferentes de *batch*

Fonte: Próprio autor.

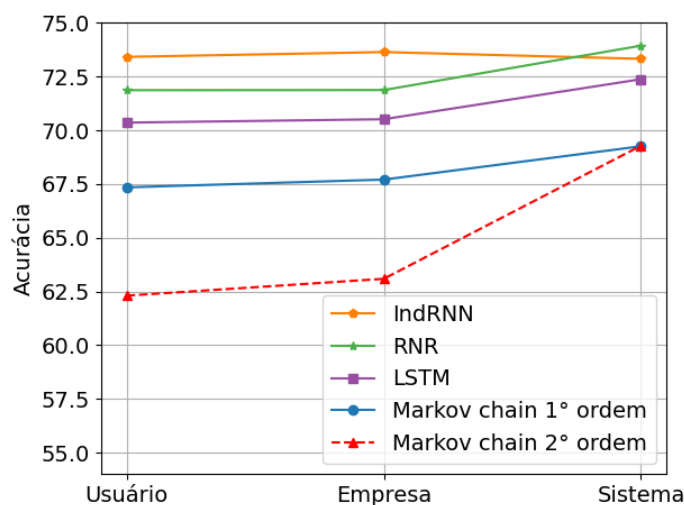
Desse modo, a partir dos resultados desses experimentos, pode-se concluir que, com base na acurácia, o cenário mais global, que utiliza todas as sequências do usuários do sistema para gerar o modelo, é o que melhor se adéqua à tarefa de recomendação de próxima funcionalidade nos sistemas representados na base de dados utilizada. Além disso, foi possível perceber a necessidade que os modelos da cadeia de markov de 2° tem de grande quantidade de dados.

#### 5.4.4 Comparativo geral

A partir dos experimentos, nos três cenários anteriores foi possível observar que, por uma leve diferença, a utilização de um modelo que engloba o histórico de uso de todos os usuários de um sistema possui uma capacidade de predição de próxima funcionalidade maior. Além disso, é possível concluir que a utilização de listas de recomendação com *rank* de tamanho 5 é uma excelente alternativa para gerar melhores recomendações aos usuários, visto que foi alcançado uma melhora de mais de 50% na acurácia de todos os modelos. Apesar disso, há outros fatores que influenciam na adoção desses modelos em uma aplicação real. Desta forma, nesta seção é apresentado um comparativo geral dos resultados obtidos nos cenários anteriores juntamente com outras métricas de avaliação.

A Figura 5.14 trás uma visão geral dos resultados obtidos nos três cenários para listas de recomendação com *rank* de tamanho 5. Na figura é possível perceber como do cenário de usuário para empresa a acurácia quase que se mantém inalterada, porém, para o cenário do sistema, fica evidente uma melhora na precisão dos modelos, especialmente no modelo da cadeia de markov de 2° ordem por necessitar de grandes volumes de dados para cada modelo.

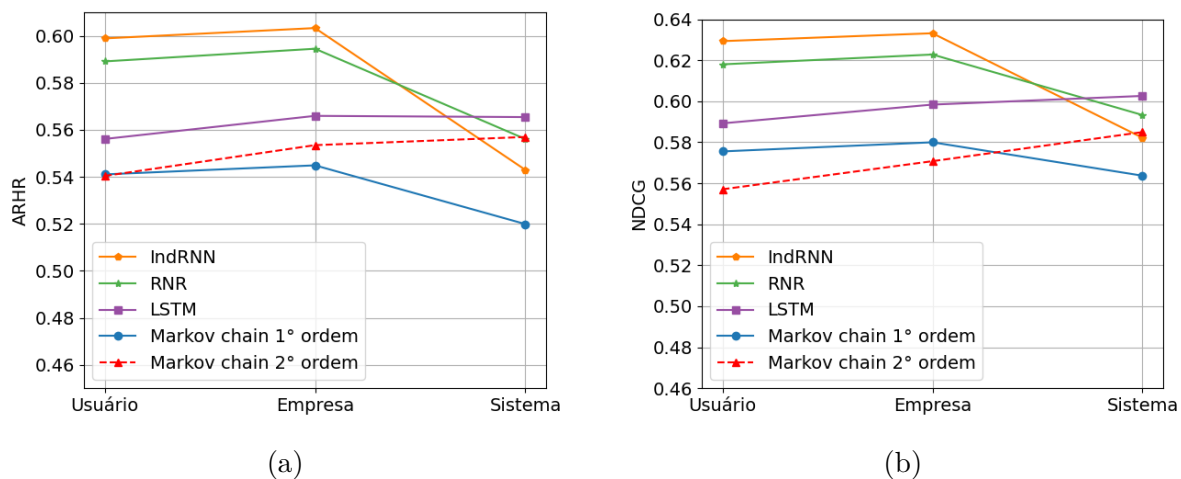
Figura 5.14 – Comparativo da acurácia para todos os cenários com *rank* 5



Fonte: Próprio autor.

Apesar de a acurácia expressar o quão preciso são os modelos em prever as próximas funcionalidades que o usuário acessaria, é importante em um sistema de recomendação ter-se uma visão geral sobre a qualidade dessas recomendações. Para tanto, foram utilizadas duas métricas: ARHR e NDCG. O objetivo de ambas é o mesmo: calcular a qualidade das listas de recomendações atribuindo valores entre 0 e 1 com base no quanto a funcionalidade correta está próxima da primeira posição da lista (quanto mais próxima da primeira posição, mais próximo é o valor de 1). Por serem métricas que apenas avaliam a qualidade do *rank*, essas métricas não levam em consideração recomendações incorretas. Em outras palavras, essas métricas não avaliam o quanto os modelos acertam, mas sim o quão bem eles acertam. Na Figura 5.15 têm-se essas duas métricas para os três cenários.

Figura 5.15 – (a) ARHR para *rank* 5; (b) NDCG para *rank* 5.



Fonte: Próprio autor.

A partir das figuras, é possível notar que as redes IndRNN e RNR padrão possuem as melhores listas de recomendações para os dois primeiros cenários, ou seja, as recomendações corretas vêm no topo das listas, porém, para o terceiro cenário essa qualidade cai para próximo da rede LSTM e da cadeia de markov de 2º ordem. Isso sugere que, embora neste cenário os modelos acertem mais, os itens corretos não estão nas primeiras posições das listas de recomendações geradas. Além disso, percebe-se uma melhora na qualidade das recomendações dos modelos de markov de 2º nos cenários mais globais.

Além das métricas já descritas, tomou-se também como critério de avaliação o tempo de treinamento e predição dos modelos, pois, considerando o uso em aplicações reais, esses valores podem ter um peso forte na escolha dos modelos se considerarmos recursos computacionais limitados ou a necessidade de recomendações rápidas. Diante disto, a Tabela 5.2 apresenta tanto um comparativo geral dos resultados do experimentos, como também trás as informações do tempo médio de treinamento e predição de cada algoritmo.

Tabela 5.2 – Acurácia média e tempo de treinamento para as *top-5* recomendações. O desvio padrão é denotado por  $\pm$ .

|         | Modelos         | Acurácia (%)                          | ARHR                                  | NDCG                                  | Tempo (seg)    |                |
|---------|-----------------|---------------------------------------|---------------------------------------|---------------------------------------|----------------|----------------|
|         |                 |                                       |                                       |                                       | treinamento    | teste          |
| Usuário | Markov Chain    | 67, 34 $\pm$ 12, 88                   | 0, 541 $\pm$ 0, 153                   | 0, 575 $\pm$ 0, 155                   | <b>0, 0004</b> | 0, 0088        |
|         | Markov Chain 2° | 62, 30 $\pm$ 15, 83                   | 0, 540 $\pm$ 0, 182                   | 0, 557 $\pm$ 0, 186                   | <b>0, 0004</b> | <b>0, 0065</b> |
|         | RNR             | 71, 87 $\pm$ 12, 76                   | 0, 589 $\pm$ 0, 168                   | 0, 617 $\pm$ 0, 168                   | 4, 44          | 0, 0572        |
|         | LSTM            | 70, 36 $\pm$ 13, 81                   | 0, 556 $\pm$ 0, 183                   | 0, 589 $\pm$ 0, 181                   | 7, 31          | 0, 0917        |
|         | IndRNN          | <b>73, 42 <math>\pm</math> 12, 23</b> | <b>0, 598 <math>\pm</math> 0, 165</b> | <b>0, 629 <math>\pm</math> 0, 164</b> | 4, 59          | 0, 0584        |
| Empresa | Markov Chain    | 67, 71 $\pm$ 12, 32                   | 0, 544 $\pm$ 0, 149                   | 0, 579 $\pm$ 0, 151                   | <b>0, 0006</b> | 0, 0166        |
|         | Markov Chain 2° | 63, 09 $\pm$ 15, 06                   | 0, 553 $\pm$ 0, 178                   | 0, 570 $\pm$ 0, 181                   | 0, 0007        | <b>0, 0118</b> |
|         | RNR             | 71, 88 $\pm$ 12, 07                   | 0, 594 $\pm$ 0, 165                   | 0, 622 $\pm$ 0, 164                   | 7, 25          | 0, 0763        |
|         | LSTM            | 70, 52 $\pm$ 13, 07                   | 0, 565 $\pm$ 0, 179                   | 0, 598 $\pm$ 0, 177                   | 11, 46         | 0, 1185        |
|         | IndRNN          | <b>73, 64 <math>\pm</math> 11, 18</b> | <b>0, 603 <math>\pm</math> 0, 160</b> | <b>0, 633 <math>\pm</math> 0, 159</b> | 7, 58          | 0, 0765        |
| Sistema | Markov Chain    | 69, 26 $\pm$ 09, 55                   | 0, 519 $\pm$ 0, 086                   | 0, 563 $\pm$ 0, 095                   | <b>0, 1654</b> | 24, 43         |
|         | Markov Chain 2° | 69, 28 $\pm$ 10, 06                   | 0, 556 $\pm$ 0, 137                   | 0, 584 $\pm$ 0, 144                   | 0, 1776        | <b>11, 35</b>  |
|         | RNR             | <b>73, 94 <math>\pm</math> 07, 77</b> | 0, 556 $\pm$ 0, 113                   | 0, 593 $\pm$ 0, 114                   | 2616           | 26, 94         |
|         | LSTM            | 72, 38 $\pm$ 08, 75                   | <b>0, 565 <math>\pm</math> 0, 135</b> | <b>0, 602 <math>\pm</math> 0, 132</b> | 2979           | 31             |
|         | IndRNN          | 73, 33 $\pm$ 06, 42                   | 0, 542 $\pm$ 0, 108                   | 0, 582 $\pm$ 0, 109                   | 2735           | 20, 89         |

A partir da Tabela 5.2, é possível observar que, de longe, a maior discrepâncias entre os modelos dos algoritmos utilizados está no tempo de treinamento das cadeias de markov em relação as redes recorrentes. A explicação para tal fato é simples, as cadeias de markov aplicam simples operações de soma em seu treinamento, enquanto as redes recorrentes, principalmente LSTM, possui varias operações de multiplicação de matrizes. Além disso, é possível observar que o cenário do usuário, apesar de ter as maiores acurácias é também o que possui os maior desvio padrão. Isso provavelmente se dá ao fato de que nesse cenário, cada foi gerado com um numero de sequências bem inferior ao que acontece no cenário de sistema. Algo interessante de se notar é que, se tratando do tempo necessário para predizer as próximas funcionalidades, a diferença de tempo entre os modelos não é tão grande quando comparada ao treinamento.

## 6 Conclusão e Trabalhos Futuros

### 6.1 Conclusão

Neste trabalho, é proposto e avaliado um sistema de recomendação de próxima funcionalidade em softwares/sistemas computacionais que tem como objetivo auxiliar os usuários a utilizarem essas aplicações de modo mais eficiente através de recomendações inteligentes das próximas funcionalidades que o usuário gostaria de/deveria acessar. Esse sistema foi modelado como um sistema de recomendação sensível à sequência, visto que as recomendações são baseadas no histórico de uso das funcionalidades do sistema pelos usuários.

Para avaliar a empregabilidade desse sistema de recomendação, foi utilizado uma base de dados que representa o uso em ambiente real de 6 sistemas de contabilidade com milhares de usuários. As recomendações foram geradas a partir de modelos treinados com algoritmos de aprendizado de máquina comumente usados em sistemas de recomendação também sensíveis à sequências e predições de séries temporais em geral, sendo esses: cadeias de Markov de primeira e segunda ordem, Rede Neural Recorrente padrão, LSTM e IndRNN. O processo de aprendizado se deu a partir da representação dos históricos de interação dos usuários com as funcionalidades dos sistemas em forma de sequências. Além disso, os experimentos foram divididos em três cenários, nos quais os históricos dos usuários foram compartilhadas com os demais usuários em diferentes níveis, tendo como objetivo avaliar o quanto o compartilhamento da experiência de cada usuário ajuda nas recomendações de próxima funcionalidade.

Os resultados mostram que a utilização de listas de recomendações com as 5 funcionalidades mais prováveis aumentou o desempenho dos algoritmos em aproximadamente 50%, saindo de acurácias em torno de 50% para 70%. Embora 70% não seja o ideal quando se fala em modelos de aprendizado de máquina, vale salientar que, considerando que ao todo haviam mais de 1000 funcionalidades nos sistemas, recomendações com 70% de acurácia já oferecem uma boa ajuda aos usuários. Além disso, enquanto que para uma baixa quantidade de dados as redes neurais recorrentes conseguiram um bom desempenho, os modelos de cadeia de markov de segunda ordem necessitaram de uma grande quantidade de sequências para alcançar uma precisão semelhante. Já em relação ao tempo de treinamento dos modelos, os dois modelos das cadeias de markov foram superiores em todos os cenários. Entretanto, para o tempo de predição das próximas funcionalidades do sistema, todos os modelos alcançaram tempos semelhantes.

Podemos concluir também que a escolha dos algoritmos e do cenário para um sistema

de recomendação em próxima funcionalidade em um ambiente real depende basicamente de dois fatores: a forma como os modelos são treinados e atualizados e o quanto as tarefas que os usuários realizam são semelhantes. Como, principalmente as redes recorrentes, possuem um alto custo computacional, em um ambiente onde é necessário atualizar os modelos todos os dias não é recomendável utilizar essas redes, sacrificando assim um pouco da acurácia com modelos de cadeias de markov, porém com um treinamento mais rápido. Porém, caso os modelos sejam treinados em intervalos grandes de tempo, as redes recorrentes são uma boa escolha, pois se adaptam bem desde pequenos históricos de uso, até grandes sequências. Outra forma seria utilizando uma mescla desses dois tipos de algoritmos, com as cadeias de markov sendo utilizadas inicialmente no sistema, enquanto as redes recorrentes ainda não são treinadas.

## 6.2 Trabalhos Futuros

Como trabalhos futuros, pretende-se investigar alguns cenários e técnicas que, devido aos recursos computacionais não foi possível realizar. Dessa forma, pretende-se realizar esses mesmos experimentos em uma base de dados maior e mais balanceada, visto que a pouca quantidade de dados para alguns usuários limitou os modelos. Com isso, seria possível investigar com mais detalhes o decaimento da acurácia em *batches* maiores. Pretende-se também investigar como a ordenação das sequências com base nos eventos de abertura e fechamento da funcionalidade influência nas recomendações, visto que neste trabalho foi utilizado apenas o evento de ativação da janela da funcionalidade. Esse experimento é válido pois, em alguns sistemas, as telas das funcionalidades se sobrepõem, assim não faz sentido recomendar uma funcionalidade que já esta aberta.

Por fim, pretende-se utilizar técnicas de configuração das redes neurais de forma automatizada, como busca em grade, para encontrar parâmetros mais adequados e ver como isso influência no aprendizado. Além disso, pretende-se investigar o desempenho de outras redes neurais recorrentes aplicadas ao problema.

# Referências

ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Disponível em: <<https://www.tensorflow.org/>>. Citado na página 85.

AGGARWAL, C. C. Evaluating recommender systems. In: \_\_\_\_\_. *Recommender Systems: The Textbook*. Cham: Springer International Publishing, 2016. p. 225–254. ISBN 978-3-319-29659-3. Disponível em: <[https://doi.org/10.1007/978-3-319-29659-3\\_7](https://doi.org/10.1007/978-3-319-29659-3_7)>. Citado 2 vezes nas páginas 68 e 71.

AGGARWAL, C. C. An introduction to recommender systems. In: \_\_\_\_\_. *Recommender Systems: The Textbook*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2016. p. 377–408. ISBN 3319296574. Citado 8 vezes nas páginas 26, 27, 56, 57, 58, 59, 61 e 62.

AGHDAM, M. H. et al. Adapting recommendations to contextual changes using hierarchical hidden markov models. In: *Proceedings of the 9th ACM Conference on Recommender Systems*. [S.l.: s.n.], 2015. (RecSys '15), p. 241–244. Citado na página 27.

ARAZY, O.; KUMAR, N.; SHAPIRA, B. Improving social recommender systems. *IT professional*, IEEE, v. 11, n. 4, p. 38–44, 2009. Citado na página 60.

BEN-SHIMON, D. et al. Recommender system from personal social networks. In: *Advances in Intelligent Web Mastering*. [S.l.]: Springer, 2007. p. 47–55. Citado na página 60.

BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, Institute of Electrical and Electronics Engineers (IEEE), v. 5, n. 2, p. 157–166, mar 1994. Disponível em: <[https://doi.org/10.1109%2F72.279181](https://doi.org/10.1109/2F72.279181)>. Citado na página 50.

BLITZSTEIN, J. K.; HWANG, J. Introduction to probability. In: \_\_\_\_\_. first. Bosa Roca, United States: Taylor & Francis Inc, 2014. cap. Markov chains. Citado na página 37.

BOBADILLA, J. et al. Recommender systems survey. *Knowledge-Based Systems*, v. 46, p. 109 – 132, 2013. ISSN 0950-7051. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0950705113001044>>. Citado na página 56.

BURKE, R. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, v. 12, n. 4, p. 331–370, Nov 2002. ISSN 1573-1391. Disponível em: <<https://doi.org/10.1023/A:1021240730564>>. Citado na página 56.

BURKE, R. Hybrid web recommender systems. In: \_\_\_\_\_. *The Adaptive Web: Methods and Strategies of Web Personalization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 377–408. ISBN 978-3-540-72079-9. Disponível em: <[https://doi.org/10.1007/978-3-540-72079-9\\_12](https://doi.org/10.1007/978-3-540-72079-9_12)>. Citado 2 vezes nas páginas 56 e 57.

CHOLLET, F. et al. *Keras*. 2015. Disponível em: <<https://keras.io>>. Citado na página 85.

DEERWESTER, S. et al. Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, v. 41, n. 6, p. 391–407, 1990. Citado na página 55.

DESHPANDE, M.; KARYPIS, G. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, ACM New York, NY, USA, v. 22, n. 1, p. 143–177, 2004. Citado na página 72.

DESHPANDE, M.; KARYPIS, G. Selective markov models for predicting web page accesses. *ACM Trans. Internet Technol.*, Association for Computing Machinery, New York, NY, USA, v. 4, n. 2, p. 163–184, maio 2004. ISSN 1533-5399. Disponível em: <<https://doi.org/10.1145/990301.990304>>. Citado na página 75.

FOLTZ, P. W.; DUMAIS, S. T. Personalized information delivery: An analysis of information filtering methods. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 35, n. 12, p. 51–60, dez. 1992. ISSN 0001-0782. Disponível em: <<https://doi.org/10.1145/138859.138866>>. Citado na página 55.

FU, X.; BUDZIK, J.; HAMMOND, K. J. Mining navigation history for recommendation. In: *Proceedings of the 5th International Conference on Intelligent User Interfaces*. New York, NY, USA: Association for Computing Machinery, 2000. (IUI '00), p. 106–112. ISBN 1581131348. Disponível em: <<https://doi.org/10.1145/325737.325796>>. Citado 3 vezes nas páginas 26, 63 e 75.

GE, M.; DELGADO, C.; JANNACH, D. Beyond accuracy: Evaluating recommender systems by coverage and serendipity. In: . [S.l.: s.n.], 2010. p. 257–260. Citado na página 68.

GERS, F. A.; SCHMIDHUBER, J. A.; CUMMINS, F. A. Learning to forget: Continual prediction with lstm. *Neural Comput.*, MIT Press, Cambridge, MA, USA, v. 12, n. 10, p. 2451–2471, out. 2000. ISSN 0899-7667. Disponível em: <<https://doi.org/10.1162/089976600300015015>>. Citado na página 51.

GOLDBERG, D. et al. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 35, n. 12, p. 61–70, dez. 1992. ISSN 0001-0782. Disponível em: <<https://doi.org/10.1145/138859.138867>>. Citado na página 55.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <<http://www.deeplearningbook.org>>. Citado na página 50.

HE, Q. et al. Web query recommendation via sequential query prediction. In: *IEEE 25th International Conference on Data Engineering*. [S.l.: s.n.], 2009. p. 1443–1454. Citado na página 27.

HENSLEY, C. B. Selective dissemination of information (sdi): State of the art in may, 1963. In: *Proceedings of the May 21-23, 1963, Spring Joint Computer Conference*. New York, NY, USA: Association for Computing Machinery, 1963. (AFIPS '63 (Spring)), p. 257–262. ISBN 9781450378802. Disponível em: <<https://doi.org/10.1145/1461551.1461584>>. Citado na página 55.

HIDASI, B. et al. Session-based recommendations with recurrent neural networks. *CoRR*, v. 1511.06939, 2015. Citado 2 vezes nas páginas 27 e 70.

HOCHREITER, S. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, v. 91, n. 1, 1991. Citado na página 50.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural Comput.*, MIT Press, Cambridge, MA, USA, v. 9, n. 8, p. 1735–1780, nov. 1997. ISSN 0899-7667. Disponível em: <<https://doi.org/10.1162/neco.1997.9.8.1735>>. Citado na página 50.

JANNACH, D.; LERCHE, L.; JUGOVAC, M. Adaptation and evaluation of recommendations for short-term shopping goals. In: *Proceedings of the 9th ACM Conference on Recommender Systems*. New York, NY, USA: Association for Computing Machinery, 2015. (RecSys '15), p. 211–218. ISBN 9781450336925. Disponível em: <<https://doi.org/10.1145/2792838.2800176>>. Citado 2 vezes nas páginas 70 e 75.

JANNACH, D.; LUDEWIG, M. When recurrent neural networks meet the neighborhood for session-based recommendation. In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. [S.l.: s.n.], 2017. (RecSys '17), p. 306–310. Citado na página 27.

JÄRVELIN, K.; KEKÄLÄINEN, J. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, Association for Computing Machinery, New York, NY, USA, v. 20, n. 4, p. 422–446, out. 2002. ISSN 1046-8188. Disponível em: <<https://doi.org/10.1145/582415.582418>>. Citado 2 vezes nas páginas 73 e 74.

KOREN, Y.; BELL, R.; VOLINSKY, C. Matrix factorization techniques for recommender systems. *Computer*, IEEE, v. 42, n. 8, p. 30–37, 2009. Citado na página 59.

Li, S. et al. Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN. *arXiv e-prints*, p. arXiv:1803.04831, Mar 2018. Citado 2 vezes nas páginas 52 e 53.

LI, S. et al. Deep independently recurrent neural network (indrnn). *CoRR*, abs/1910.06251, 2019. Disponível em: <<http://arxiv.org/abs/1910.06251>>. Citado na página 53.

LIU, F.; LEE, H. J. Use of social network information to enhance collaborative filtering performance. *Expert Systems with Applications*, v. 37, n. 7, p. 4772 – 4778, 2010. ISSN 0957-4174. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0957417409011075>>. Citado na página 58.

LU, E. H.-C.; LIN, Y.-W.; CIOU, J.-B. Mining mobile application sequential patterns for usage prediction. In: IEEE. *2014 IEEE International Conference on Granular Computing (GrC)*. [S.l.], 2014. p. 185–190. Citado na página 82.

MATEJKA, J. et al. Communitycommands: Command recommendations for software applications. In: *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA: Association for Computing Machinery, 2009. (UIST '09), p. 193–202. ISBN 9781605587455. Disponível em: <<https://doi.org/10.1145/1622176.1622214>>. Citado na página 26.

MCCULLOCH, W.; PITTS, W. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, v. 5, p. 127–147, 1943. Citado na página 39.

MITCHELL, T. M. *Machine Learning*. 1. ed. USA: McGraw-Hill, Inc., 1997. ISBN 0070428077. Citado na página 31.

MOBASHER, B.; COOLEY, R.; SRIVASTAVA, J. Automatic personalization based on web usage mining. *Communications of the ACM*, ACM New York, NY, USA, v. 43, n. 8, p. 142–151, 2000. Citado 2 vezes nas páginas 26 e 63.

MOBASHER, B. et al. Using sequential and non-sequential patterns in predictive web usage mining tasks. In: IEEE. *2002 IEEE International Conference on Data Mining, 2002. Proceedings*. [S.l.], 2002. p. 669–672. Citado 2 vezes nas páginas 26 e 63.

MURPHY, K. P. *Machine Learning: A Probabilistic Perspective*. [S.l.]: The MIT Press, 2012. ISBN 0262018020. Citado 2 vezes nas páginas 37 e 38.

NATARAJAN, N.; SHIN, D.; DHILLON, I. S. Which app will you use next?: collaborative filtering with interactional context. In: ACM. *Proceedings of the 7th ACM conference on Recommender systems*. [S.l.], 2013. p. 201–208. Citado na página 82.

NILSSON, N. J. *Introduction to Machine Learning. An early draft of a proposed textbook*. [S.l.: s.n.], 1996. Citado na página 32.

PARK, D. H. et al. A literature review and classification of recommender systems research. *Expert Systems with Applications*, v. 39, n. 11, p. 10059 – 10072, 2012. ISSN 0957-4174. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0957417412002825>>. Citado 2 vezes nas páginas 27 e 56.

QUADRANA, M.; CREMONESI, P.; JANNACH, D. Sequence-aware recommender systems. *CoRR*, v. 1802.08452, 2018. Citado 9 vezes nas páginas 27, 57, 63, 64, 65, 67, 69, 70 e 71.

QUADRANA, M. et al. Personalizing session-based recommendations with hierarchical recurrent neural networks. In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. New York, NY, USA: Association for Computing Machinery, 2017. (RecSys '17), p. 130–137. ISBN 9781450346528. Disponível em: <<https://doi.org/10.1145/3109859.3109896>>. Citado na página 70.

RESNICK, P. et al. Grouplens: an open architecture for collaborative filtering of netnews. In: ACM. *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. [S.l.], 1994. p. 175–186. Citado 2 vezes nas páginas 55 e 58.

RESNICK, P.; VARIAN, H. R. Recommender systems. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 40, n. 3, p. 56–58, mar. 1997. ISSN 0001-0782. Disponível em: <<https://doi.org/10.1145/245108.245121>>. Citado na página 56.

RICCI, F.; ROKACH, L.; SHAPIRA, B. Introduction to recommender systems handbook. In: *Recommender systems handbook*. [S.l.]: Springer, 2011. p. 1–35. Citado 5 vezes nas páginas 27, 55, 56, 57 e 60.

RICCI, F.; ROKACH, L.; SHAPIRA, B. *Recommender systems handbook*. [S.l.]: Springer, 2011. Citado na página 26.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, v. 65, n. 6, p. 386–408, 1958. ISSN 0033-295X. Disponível em: <<http://dx.doi.org/10.1037/h0042519>>. Citado na página 40.

- RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning Representations by Back-propagating Errors. *Nature*, v. 323, n. 6088, p. 533–536, 1986. Disponível em: <<http://www.nature.com/articles/323533a0>>. Citado na página 41.
- RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3. ed. [S.l.]: Prentice Hall, 2010. Citado 2 vezes nas páginas 39 e 40.
- SALTON, G.; BUCKLEY, C. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, v. 24, n. 5, p. 513–523, 1988. ISSN 0306-4573. Disponível em: <<http://www.sciencedirect.com/science/article/pii/0306457388900210>>. Citado na página 55.
- SARWAR, B. M. et al. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In: *Proceedings of the fifth international conference on computer and information technology*. [S.l.: s.n.], 2002. v. 1, p. 291–324. Citado na página 58.
- SCHAFER, J. B.; KONSTAN, J.; RIEDL, J. Recommender systems in e-commerce. In: ACM. *Proceedings of the 1st ACM conference on Electronic commerce*. [S.l.], 1999. p. 158–166. Citado na página 56.
- SHALEV-SHWARTZ, S.; BEN-DAVID, S. *Understanding Machine Learning: From Theory to Algorithms*. USA: Cambridge University Press, 2014. ISBN 1107057132. Citado na página 32.
- SHANI, G.; GUNAWARDANA, A. Evaluating recommendation systems. In: \_\_\_\_\_. *Recommender Systems Handbook*. Boston, MA: Springer US, 2011. p. 257–297. ISBN 978-0-387-85820-3. Disponível em: <[https://doi.org/10.1007/978-0-387-85820-3\\_8](https://doi.org/10.1007/978-0-387-85820-3_8)>. Citado 3 vezes nas páginas 68, 71 e 73.
- SHANI, G.; HECKERMAN, D.; BRAFMAN, R. I. An mdp-based recommender system. *J. Mach. Learn. Res.*, v. 6, p. 1265–1295, 2005. Citado na página 27.
- SHARDANAND, U.; MAES, P. Social information filtering: Algorithms for automating "word of mouth". In: . [S.l.]: ACM Press, 1995. p. 210–217. Citado na página 58.
- SINHA, R. R.; SWEARINGEN, K. et al. Comparing recommendations made by online systems and friends. In: *DELOS*. [S.l.: s.n.], 2001. Citado na página 62.
- SMYTH, B. Case-based recommendation. In: \_\_\_\_\_. *The Adaptive Web: Methods and Strategies of Web Personalization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 342–376. ISBN 978-3-540-72079-9. Disponível em: <[https://doi.org/10.1007/978-3-540-72079-9\\_11](https://doi.org/10.1007/978-3-540-72079-9_11)>. Citado na página 59.
- SOHAIL, S. S.; SIDDIQUI, J.; ALI, R. Classifications of recommender systems: A review. *Journal of Engineering Science & Technology Review*, v. 10, n. 4, 2017. Citado na página 56.
- WU, Y. et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. Disponível em: <<http://arxiv.org/abs/1609.08144>>. Citado na página 47.

ZHANG, Y. et al. Sequential click prediction for sponsored search with recurrent neural networks. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2014. (AAAI'14), p. 1369–1375. Citado na página 27.

ZHAO, G. et al. Increasing temporal diversity with purchase intervals. In: *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: Association for Computing Machinery, 2012. (SIGIR '12), p. 165–174. ISBN 9781450314725. Disponível em: <<https://doi.org/10.1145/2348283.2348309>>. Citado na página 70.