UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA

TAINÁ ARRUDA DE AGUILAR

O USO DE REDES NEURAIS PARA CLASSIFICAÇÃO DE GÊNEROS LITERÁRIOS

Belo Horzionte
Outubro de 2020

TAINÁ ARRUDA DE AGUILAR

O USO DE REDES NEURAIS PARA CLASSIFICAÇÃO DE GÊNEROS LITERÁRIOS

Monografia apresentada para conclusão do curso de graduação em Engenharia Mecânica da UFMG, como parte dos requisitos necessários à obtenção do título de Engenheira Mecânica.

Orientador: **Prof. Dr. Eduardo José Lima II**
Universidade Federal de Minas Gerais

Belo Horzionte
Outubro de 2020

Dedico esse trabalho ao meu pai porque não tinha como dedicar para outra pessoa.

AGRADECIMENTOS

Definitivamente fazer um TCC é uma tarefa muito difícil e sozinha, eu possivelmente não estaria nem perto de terminar. Para minha sorte eu tive pessoas incríveis que me apoiaram e que merecem um espacinho aqui:

- Obviamente eu vou começar citando o meu pai, cujo trabalho na delegacia de julgamento da receita me serviu de inspiração para o projeto desenvolvido, sem ele nem tema eu teria. E muito provavelmente nem um TG, já que ele foi meu maior motivador a escrever, corrigindo meus erros de português (inclusive, se tiver algum erro nessa página já peço desculpas, mas eu não podia dar spoilers) e escutando todas as minhas tentativas de explicações sobre o assunto que foram fundamentais para que eu organizasse a confusão de ideias da minha cabeça. Se esse trabalho hoje faz sentido, foi graças a ajuda dele.
- Minha mãe eu agradeço por todo o suporte durante o curso, levando comida durante a minhas reuniões infinitas de PJ e dando suporte nas minhas empreitadas nos festivais de espetinho do vôlei. O Vitor também por ser o melhor irmão que eu podia querer, preocupado com a minha formatura e sempre me defendendo em qualquer assunto.
- Agradeço ao meu orientador Eduardo, que definitivamente é um professor diferenciado. São poucos os que, além de ensinar bem, ainda tomam o tempo para se preocupar com o aluno e realmente escutá-lo. Em todas as matérias suas que fiz, eu sempre quis ser uma aluna melhor, estudar além daquilo que era dado em sala de aula (nem sempre eu tive sucesso). Enfim, muito obrigada por acreditar no meu trabalho.
- A UFMG eu agradeço pelas oportunidades que eu tive e as pessoas que ela me deu a oportunidade de conhecer. Ter passado pela PJ foi algo que me mudou completamente e foi meu primeiro desafio da vida adulta. E principalmente, me trouxe as amigas mais incríveis que eu podia encontrar nessa engenharia (MecGirls S2). Obvio que ainda vou citar o Grifo. Minha experiência na atlética foi a mais intensa de todas. Tivemos muitas alegrias, ganhamos muitos campeonatos e também perdemos alguns, fiz infinitas amizades, tive algumas decepções, passei raiva até não poder mais, perdi um tanto considerável de horas de sono. Mas fazer o que, não tem como não dizer que não valeu a pena. O Grifo vai ter sempre um lugarzinho no meu coração.

"Big Data processes codify the past. They do not invent the future. Doing that requires moral imagination, and that's something only humans can provide. We have to explicitly embed better values into our algorithms, creating Big Data models that follow our ethical lead (Cathy O'Neil, 2016)."

RESUMO

Este trabalho tem como objetivo realizar uma análise de informações de livros e desenvolver um modelo computacional para classificação de livros em gêneros literários utilizando redes neurais recorrentes. Para tal, obteve-se um conjunto de dados composto pela informação de mais de 50 mil livros retiradas do site Goodreads, embora apenas 30 mil estivessem elegíveis para uso. Todo trabalho foi desenvolvido em Python. Foi aplicada uma rede neural recorrente utilizando a biblioteca TensorFlow e a API Keras, com a descrição como *input*. Foi considerado que cada livro poderia pertencer a apenas um gênero e o problema foi tratado como uma classificação multi classe. Por fim, o modelo foi comparado com algoritmos usuais para esse tipo de problema: regressão logística, máquinas de vetores de suporte e Naïve Bayes.

Palavras-chave: Aprendizado de Máquina. Estudos de gênero. LSTM.

ABSTRACT

This present project goals is making a data analysis of book's information and creating a computer model that is capable of classifying these books in genre. To do so, information of over 50 thousand books were collected from Goodreads, albeit only 30 thousand were able to be used for classification. All scripts were written in Python. It used a recurrent neural network and the TensorFlow library and its API Keras, with book's descriptions as inputs. This problem was considered as multi classification task in which each book could only belong to a single genre. Finally, the model was compared to well-known classification algorithms such as logistic regression, support vector classifiers and Naïve Bayes.
Keywords: Machine Learning. Genre Theory. LSTM.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

RNN	Redes Neurais Recorrentes (do inglês: *Recurrent Neural Network*)
MSE	Soma dos Quadrados dos Resíduos (do inglês: *Mean Squared Error*)
JSON	*JavaScript Object Notation*
API	*Application Programming Interface*
XML	*Extensible Markup Language*
HTML	Linguagem de Marcação de Hipertexto (do inglês: *HyperText Markup Language*)
NLP	Processamento de Linguagem Natural (do inglês: *Natural Language Processing*)
ISBN	*International Standard Book Number*
CE	*Cross-Entropy Loss*
LSTM	*Long Short-Term Memory*
SVC	*Support Vector Classifier*
RSS	Soma dos Quadrados do Erro (do inglês: *Residual Sum of Squares*

LISTA DE SÍMBOLOS

σ	Função Sigmoide
β	Parâmetros de ajuste do modelo de regressão
χ^2	Distribuição qui-quadrado

SUMÁRIO

1 INTRODUÇÃO

O gênero de uma obra (livro, filme ou música) é uma forma de categorização. Funciona como um guia inicial quando não dispomos de um sistema de recomendação mais sofisticado, como por exemplo o utilizado pela Netflix. Se a classificação estiver correta, o leitor encontrará mais facilmente obras que lhe agradem. Por outro lado, a classificação errada de um livro pode trazer problemas, pois dificulta o alcance de seu público alvo, que as vezes nem chega a saber da existência do livro, ou que decide não o ler por não se identificar com o gênero do livro. E isso pode resultar, inclusive, em dificuldade de vendas e críticas negativas.

1.1 **Objetivo**

Esse trabalho tem como objetivo avaliar o uso de redes neurais recorrentes para classificação de livros com base na descrição, bem como compará-lo com modelos utilizados atualmente no processamento de linguagem natural, fornecendo uma alternativa para os modelos atuais. A comparação será feita não apenas em relação a resultados práticos avaliando métricas como acurácia e matrizes de confusão, mas avaliando pontos como facilidade de uso e interpretabilidade.

1.2 **Metodologia**

O trabalho pode ser dividido em três fases principais: a coleta dos dados, a análise e preprocessamento e a etapa de treinamento propriamente dita. Os dados foram extraídos do site GoodReads, uma rede social de livros criada pela Amazon, utilizando técnicas de *web-scrapping* amplamente populares. O dados foram analisados e preprocessados em um Jupyter Notebook, utilizando diversas técnicas de análise exploratória, demonstradas por gráficos e imagens que ilustram esse processo no item 3.2. O modelo foi treinado no ambiente Google Colaboratory. A descrição foi codificada utilizando o modelo *open-source* de representação de palavras GloVe e a arquitetura de rede neural utilizada foi muitos para um, com duas camadas e uma camada final de classificação. Todo o código foi feito em Python utilizando o *framework* Keras. O resultado foi comparado com os seguintes métodos de classificação: Naïve Bayes, regressão logística e máquinas de vetores de suporte.

2 REVISÃO BIBLIOGRÁFICA

2.1 Estudos de Gênero

Apesar do conceito de gênero ter sido amplamente estudado nos últimos quarenta anos, o termo gênero ainda está envolto em discordâncias. Um dos questionamentos é sobre a definição do gênero apenas como uma função para classificar e ordenar experiências, eventos e ações que ele representa, ou se é capaz de também gerar aquilo que ele representa culturalmente(BAWARSHI; REIFF, 2010). Em outras palavras: seria o gênero apenas um rótulo ou ele é capaz de gerar significado a uma obra?

O estudo do gênero como gerador de significado adiciona uma nova faceta ao processo de classificação, pois ao definir um livro, filme ou outra expressão cultural como de determinado gênero, altera-se também o significado da obra e como ela afeta a sociedade. Nesse caso, além de levar em conta aspectos formais, deve-se observar qual o objetivo de cada gênero.

Apesar disso, o gênero apenas como um rótulo de classificação ainda é objeto de estudo. Não existe consenso em como delimitar quais aspectos formais devem ser considerados ao classificar uma obra, muito menos regras que guiem na escolha do gênero de uma obra, embora as vezes seja possível chegar em algum senso comum em relação a um gênero especifico. Na verdade, a própria existência de um gênero concreto é questionada por alguns estudiosos, como em Feuer (1992), no qual o gênero é descrito como um conceito abstrato ao invés de algo que existe empiricamente no mundo.

Pensando no âmbito cinematográfico, Robert Stam exemplifica como diferentes fatores podem afetar a maneira de categorizar um filme:

> Enquanto alguns gêneros são baseados no conteúdo da historia (filmes de guerras), outros são emprestados da literatura (comedia, melodrama) ou de outros meios (musicais). Alguns são baseados nos atores (filmes de Asteire-Rogers) ou baseados no orçamento (*blockbusters*), enquanto outros são baseados no status do artista (filmes artísticos), identidade racial (cinema negro), localização (*western*) ou orientação sexual (cinema *queer*). (STAM, 2000)

A definição de gênero depende também do proposito de cada pesquisa, devendo estar relacionado com qual parte do fenômeno se quer destacar. Por exemplo, se o objetivo é estudar como os gêneros influenciam a interpretação do leitor, então não se deveria focar em distinções teóricas, mas sim em como o leitor identifica gêneros (CHANDLER, 1997).

2.2 **Machine Learning**

De acordo com Burkov (2019, p. 4), Machine learning é uma subárea da ciência da computação cujo objetivo é criar algoritmos a partir de uma quantidade de exemplos. Analisando essa frase, o primeiro passo é esclarecer o que é um algoritmo. Usando a definição de Sedgewick e Wayne (2011), um algoritmo é a descrição de um procedimento para resolver um problema, seja em linguagem natural ou por meio de um programa de computador que implemente o procedimento.

Mas nem sempre o problema que temos nos permite uma sequência de procedimentos ou regras bem definidas. Por exemplo, como definir quais e-mails serão considerados spans e quais não? Aqueles que tratam o remetente pelo nome próprio? Ou cuja mensagem tem "urgente" no assunto? Um humano facilmente consegue ler um e-mail e classificá-lo. Porém não conseguimos tão facilmente definir as regras que usamos para essa tomada de decisão. Assim, voltamos ao problema: se algoritmos são descrições de um procedimento e em alguns casos não é possível definir claramente esse procedimento, como fazer?

A área de Machine Learning alterou o paradigma de computação. Ao invés de escrever várias regras por meio de sequencias de *if-then-elses*, é criado um modelo capaz de aprender por meio de exemplos. Como exemplificado na Figura 1, o programador define um conjuntos de regras baseado nos *inputs*. Esse será o código do programa. Ao observar os *outputs*, poderá ser necessário que o programador faça alterações no programa. Uma metodologia extremamente útil quando, a partir dos *inputs*, pode-se determinar facilmente as regras.

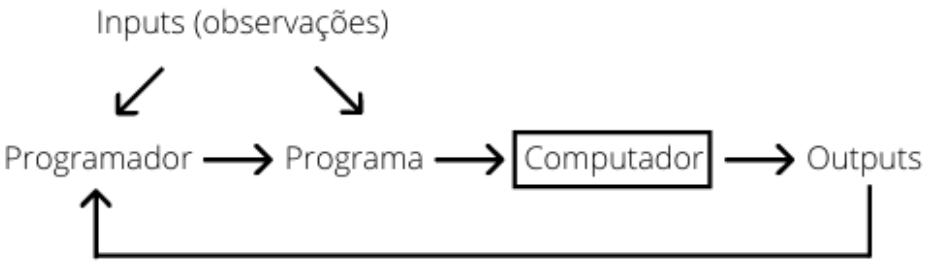

Figura 1 – Fluxo de informações no paradigma tradicional de computação

Quando as regras não estão tão claras, usar algoritmos de Machine Learning pode ser útil. O novo fluxo, visto na Figura 2, é mais simples. Consiste em fornecer os *inputs* e *outputs* ao computador para que esse gere o programa. Embora, na prática, não seja tão simples, pois

se deve escolher um programa flexível cujo comportamento é determinado por um número de parâmetros. Existem várias técnicas que podem ser utilizadas e escolher qual a mais adequada ao problema faz parte do estudo de Machine Learning.

Figura 2 – Fluxo de informações em um programa de Machine Learning

Uma comparação muito útil é a apresentada em Zhang et al. (2020). Os parâmetros do programa são como botões (como em uma televisão antiga, na qual se deve girar os botões para configurar a imagem e sincronizar em um canal). Esse botões podem ser ajustados para alterar o comportamento do programa. Ao ajustar os botões, o modelo estaria sendo programado. O cientista de dados é quem é responsável por realizar esses ajustes, buscando a melhora do modelo. O processo de trabalho do cientista de dados pode ser representado pelo diagrama na Figura 3, que se inicia com um modelo aleatório e básico que não é útil. O cientista obtém dados rotulados que permitem que os botões (parâmetros) sejam ajustados para aperfeiçoar o modelo em relação a esses exemplos, esse é o processo de treinamento do modelo. Esse processo é repetido até chegar em um resultado que seja bom o suficiente.

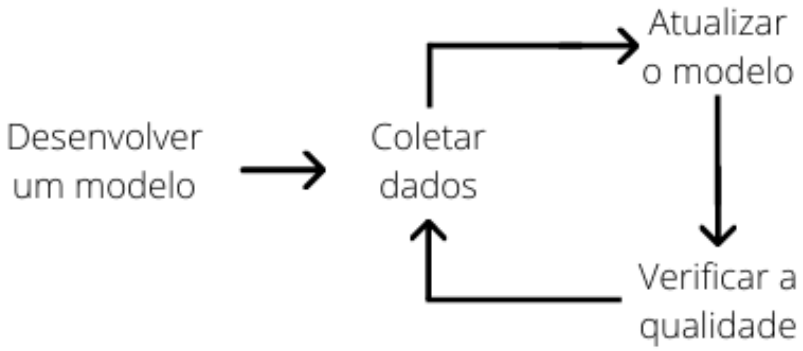

Figura 3 – Típico processo de trabalho de um cientista de dados

O fluxograma em 3 combina bastante com a seguinte definição:

> Dizem que um programa de computador aprendeu pela experiência com respeito a uma tarefa T e uma métrica de performance P, se a sua performance em T, medida por P, melhorar com a experiência E. (MITCHELL, 1997)

No caso a ser analisado no presente trabalho, os exemplos seriam as informações de livros contidos na base de treinamento e a tarefa T seria classificá-los em seus respectivos gêneros. Na seção 2.3 será discutido mais sobre a métrica de performance mencionada na citação acima.

De acordo com Géron (2017), Machine Learning é útil para:

- Problemas em que as soluções existentes requerem muitos ajustes manuais, ou uma grande lista de regras.
- Problemas complexos que não possuem uma boa solução usando um *approach* tradicional.
- Ambientes variáveis, sistemas de Machine Learning podem se adaptar a novos dados.
- Conseguir *insights* sobre problemas complexos e uma grande quantidade de dados.

Outro uso que merece ser destacado, embora esse também pudesse ser considerado uma mistura de alguns dos casos citados, são problemas em que não se consegue determinar uma resposta exata. E, portanto, não se pode definir um conjunto de regras para chegar a resposta. É possível, no máximo, definir um conjunto de regras que criam pesos, porém, nesse caso, o machine learning seria uma alternativa mais adequada.

2.3 **Loss Function**

Como mencionado na seção 2.2, deseja-se um modelo que se ajuste da melhor maneira possível aos dados. Para quantificar a qualidade do ajuste é necessário o conceito de *Loss Function*. A *Loss function* ou função de perda tem como objetivo estabelecer o quão bom ou ruim um modelo é. Lembrando do famoso aforismo de Box G. E. P.; Draper (1987), "todos os modelos estão errados, mas alguns são uteis", a utilidade de um modelo depende principalmente do contexto que está sendo trabalhado. A função perda ajuda a determinar, dentre possíveis modelos, quais podem ser uteis para a tarefa. Ter apenas uma única métrica de avaliação permite que o processo de iteração representado na Figura 3 seja realizado mais rapidamente, permitido que os modelos sejam ordenados de acordo com a sua performance e rapidamente seja decidido qual funciona melhor, como mencionado por Ng (2018, pg 20).

No geral, essa função depende dos parâmetros dos modelos e do *dataset* que está sendo utilizado e cada aplicação pode necessitar uma função de perda diferente. A definição da função de perda é um passo fundamental no processo de desenvolvimento de um projeto de Machine

Learning e se a função não representa bem os objetivos do projeto, muito possivelmente não se chegará a um modelo satisfatório.

Existem três tipos de funções perda padrão que são amplamente utilizadas em modelos preditivos a depender do tipo de tarefa que deseja-se realizar(BROWNLEE, 2017). Para problemas de regressão[1] a função mais comumente usada é o erro médio quadrático. Cuja formula está representada na equação 1.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) \tag{1}$$

Existem também as tarefas de classificação[2] que podem ser divididas em problemas binários[3] e multi-classe. Em problemas de classificação, a função de erro médio não é adequada. A função de perda precisa refletir as necessidades de um problema de classificação. Pensar em termos de número de erros (erro médio quadrático) não permite diferenciar situações como:

- Acertar com alta certeza
- Acertar com pouca certeza
- Errar com pouca certeza
- Errar com alta certeza

Assim o modelo não deve fornecer apenas a classe predita, mas também a certeza em relação a predição, ou seja, a probabilidade de cada classe. Aplicando a função de perda, um modelo que erra, mas com uma baixa confiança no valor predito deve ser menos penalizado que um modelo que erra com uma alta confiança ao prever um valor incorreto.

Uma função que atende esses requisitos é a *cross-entropy loss*, que possui variações tanto para o problema binário quanto para o problema multiclasses. A função binaria de cross entropy loss é a seguinte:

$$CE = \sum_{i=1}^{n} -(y_i \log(p_i) + (1 - y_i)\log(1 - p_i)) \tag{2}$$

em que $p_i = P(y_i = 1|X_i)$. Para usar a função de *cross-entropy loss* em um problema binário, é necessário representar as possíveis classes por $\{0,1\}$. Substituindo na equação 2, para um i em que $y = 1$, chega-se a seguinte equação:

$$CE_i = -log(p) \tag{3}$$

Ou seja, o logaritmo da probabilidade que fornecida pelo modelo de que a classe de y seja 1. Como $y = 1$, nesse caso, quanto maior p, melhor o modelo e menor a função perda[4].

[1] Tarefa em que se quer prever *output* que pode ter qualquer valor numérico especifico

[2] Tarefa em que se quer prever uma classe dentro um conjunto discreto de opções

[3] O *output* só pode assumir duas classes como {0,1}, {Sim, Não}, {Sucesso, Fracasso}

[4] Tipicamente quanto menor o valor da função perda, melhor o modelo

Pensando no caso inverso, um valor de i em que $y = 0$, chega-se à seguinte equação ao substituir y na equação 2:

$$CE_i = log(1-p) \tag{4}$$

Como a probabilidade de todos os eventos $P(y_i \in \{0,1\}|X_i) = 1$ e a probabilidade de ser 1 é igual a p, logo a probabilidade de $P(y_i \neq 1|X_i) = 1-p$ que também é a probabilidade $P(y_i = 0|X_i)$. Portanto quando maior for o valor da probabilidade $P(y_i = 0|X_i)$, menor o valor da minha função perda. Em outras palavras, um dos termos é relevante quando a classe é zero, enquanto o outro termo é relevante quando o valor da classe é um.

2.4 **Regressão Linear**

A regressão linear é um dos métodos de machine learning mais simples e também um dos mais populares. Sendo aqui descrito porque serve de ponto de partida de técnicas mais sofisticadas, inclusive redes neurais.

A regressão linear simples é um método direto para prever uma resposta quantitativa Y em função de apenas um *input* X. Para aplica-lo, é necessário fazer duas suposições acerca dos dados, de acordo com Zhang et al. (2020), são elas:

- A relação de X e Y é aproximadamente linear, com alguns ruídos
- Esses ruídos seguem uma distribuição gaussiana

A equação que rege o relacionamento é a seguinte:

$$Y \approx \beta_0 + \beta_1 X \tag{5}$$

Onde $\approx$ pode ser entendido como "é aproximadamente modelado como" (JAMES et al., 2014). As constantes β_0 e β_1 são respectivamente o intercepto e a inclinação no modelo. Porém, no presente trabalho, para possibilitar uma comparação direta com redes neurais, eles serão citados como bias e peso, respectivamente.

Antes de explicar o porque dos nomes, primeiro é interessante apresentar um novo modelo. A regressão linear múltipla assume que Y não é mais regido apenas por uma variável X, mas por um conjunto de variáveis X_p, que juntas compõem o Y. As mesmas suposições da regressão linear simples ainda se mantém:

- O relacionamento com cada variável é linear
- Os ruídos seguem uma distribuição gaussiana

A equação que rege o relacionamento é a seguinte:

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n \tag{6}$$

Essa equação representa um hiperplano em um espaço n-dimensional. O bias indica o quão deslocado esse plano está da origem do sistema de coordenadas cartesiano e os pesos determinam a influência de cada *feature* na previsão. Esse modelo realiza uma transformação afim[5] caracterizada por uma transformação linear das *features* por meio de uma soma ponderada e combinada com uma translação por meio do bias.

Para construir o modelo é necessário determinar valores para os pesos e o bias de modo que o modelo se ajuste da melhor maneira possível aos dados. E $\hat{\beta}_n$ representa a estimativa para o n-ésimo parâmetro, enquanto $\hat{y}$ representa o valor predito para y com base nos estimadores $\hat{\beta}_0$, $\hat{\beta}_1$,..., $\hat{\beta}_n$ encontrados.

Existem diversas maneiras de medir a qualidade do ajuste, ou o quão próximo o ponto predito está do valor real e com isso determinar os valores de $\hat{\beta}_0$, $\hat{\beta}_1$,..., $\hat{\beta}_n$. Porém o método mais comum envolve minimizar o critério dos mínimos quadrados. Esse método busca minimizar a soma dos quadrados do erro (RSS), que é definida pela equação 7

$$RSS = e_1^2 + e_2^2 + \cdot + e_n^2 \tag{7}$$

em que $e_n = y_n - \hat{y}$. Esse método permite obter uma solução analítica exata para o problema da regressão linear sem a necessidade de realizar um processo de otimização.

2.5 **Regressão Logística**

Quando se deseja estimar a probabilidade da ocorrência de um evento binário, a regressão linear, apesar de poder ser útil para classificação e testagem de hipóteses, não é a melhor alternativa (POHLMAN; LEITNER, 2003). Por exemplo, imagine se o objetivo fosse classificar se uma imagem é ou não um cachorro. Pode-se definir que, por exemplo, para valores abaixo de 0.5[6], a imagem não é um cachorro e acima disso, seria um cachorro. Porém o que um fornecido pelo modelo indica? Seria o resultado mais confiável se o número predito pelo modelo fosse 10 em comparação com 1? E se fosse 100 em comparação com 91? Qual a melhora do resultado? Esses são pontos que mostram como a regressão linear é falha em um caso de classificação. A

[5]Na geometria euclidiana, uma **transformação afim**, ou uma afinidade, é uma transformação geométrica que preserva linhas e paralelismo.

[6]Esse limite é arbitrário e foi escolhido apenas para exemplificar

alternativa é a regressão logística cujo valor retornado sempre estará entre 0 e 1, atendendo aos requisitos de uma probabilidade. Assim, o número fornecido pelo modelo tem um significado prático, representa uma probabilidade. Para tarefas de classificação, esse método pode ser considerado um dos mais básicos, porém entendê-lo é fundamental para auxiliar na compreensão futura de redes neurais.

A função usada na regressão logística é a função logística representada na equação 8.

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\ldots+\beta_n X_n}}{1+e^{\beta_0+\beta_1 X_1+\ldots+\beta_n X_n}} \tag{8}$$

Para ajustar a esse modelo (8), é utilizado o método método estimativo da probabilidade máxima. Ao contrário da regressão linear, com seus resíduos em distribuição normal, não é possível encontrar uma expressão em forma fechada para os valores dos coeficientes que maximize a função de probabilidade, fazendo que um processo iterativo seja usado (SHARDA; DELEN; TURBAN, 2019). A função logística sempre vai produzir uma curva em S, portanto independente dos valores de X, se obterá uma predição sensível (JAMES et al., 2014). É possível rescrever a equação 8 como:

$$\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X_1+\ldots+\beta_n X_n} \tag{9}$$

em que a quantidade $p(X)/[1-p(X)]$ é chamada de chance[7] e pode ter qualquer valor entre 0 e ∞. Ao aplicar o logaritmo dos dois lados, obtém-se:

$$log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0+\beta_1 X_1+\ldots+\beta_n X_n \tag{10}$$

O lado esquerdo da equação é chamado de *logit*. Como se pode observar, a regressão logística tem um *logit* que é linear em X. Assim, se usa um modelo linear para prever uma probabilidade, que pode ser mapeada a um rótulo de classe através de aplicação de uma regra de corte (que normalmente é 0.5) e qualquer registro com uma probabilidade maior do que o corte é classificado como 1 (BRUCE; BRUCE, 2019).

2.6 **Deep Learning**

Deep learning é um passo além dos modelos tradicionais de Machine Learning que exigiam uma maior intervenção do desenvolvedor. O modelo consiste em sucessivas transformações lineares e não lineares que distorcem o *input* até torna-lo linearmente separável.

[7] Em Probabilidade e Estatística, a chance (em inglês: *odds*) de ocorrência de um evento é a probabilidade de ocorrência deste evento dividida pela probabilidade da não ocorrência do mesmo evento

Para entender o conceito de redes neurais, será discutido o exemplo de predição do preço de venda de um imóvel a partir das seguintes informações: tamanho, número de quartos, CEP e renda média do bairro. A partir das variáveis de tamanho e numero de quartos é calculado uma nova variável, que indica o tamanho da família. Por meio do CEP, é definido uma variável que indica a acessibilidade da região. Por fim, com base no CEP e na renda média do bairro, é definido um indicador de qualidade escolar. Assim, a partir das quatro variáveis iniciais, foram construídas três novas variáveis (tamanho da família, acessibilidade e qualidade das escolas), um processo conhecido como *feature engineering*[8]. Essas três novas variáveis são usadas, por fim, para estimar o preço de venda da casa. Todo esse processo está diagramado na figura 4.

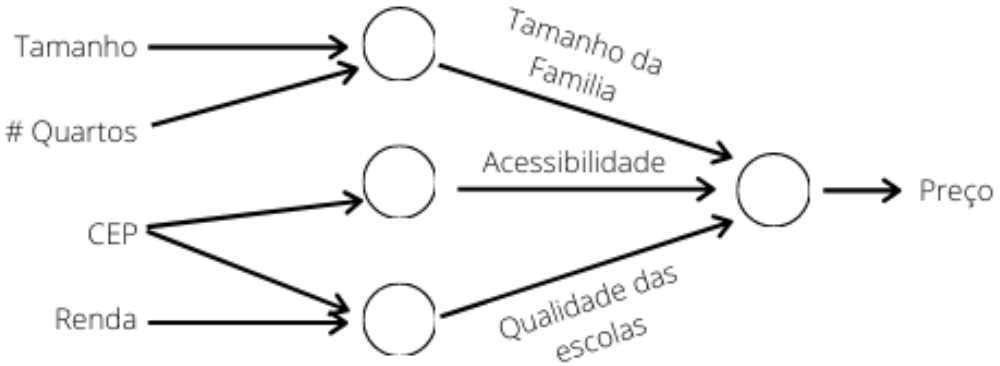

Figura 4 – Predição de preços de casas por meio de extração de *features*

A ideia de uma rede neural se assemelha a esse exemplo, a diferença é que cada *input* está conectado a cada um dos neurônios das camadas internas. E na verdade não existe uma interpretabilidade do que está sendo calculado em cada um dos neurônios. A Figura 5 é uma representação para esse caso.

O parte de redes do nome redes neurais vem do fato de todos os inputs estarem conectados a todos os neurônios de cada camada. O termo neural é derivado do fato que de que esse modelo buscava aproximar o funcionamento do cérebro, apesar de hoje em dia cada vez mais as arquiteturas serem apenas levemente baseadas na cérebro humano. Cada uma dessas redes pode ter várias camadas internas, essas camadas são conhecidas como camadas ocultas e deram origem ao termo aprendizagem profunda.

A escolha por uma representação profunda deriva da teoria dos circuitos: existem funções que se pode computar com uma pequena rede neural com L-camadas que redes mais

[8] ***Feature engineering*** é o processo de extrair *features* de dados não processados a partir de conhecimentos da aplicação e técnicas de mineração de dados

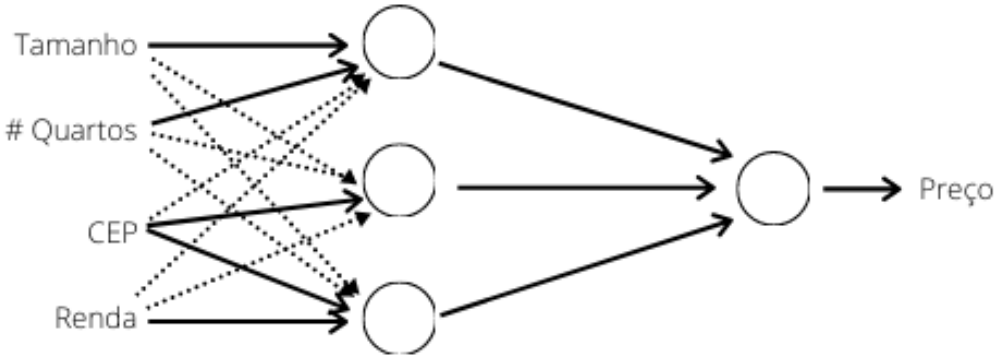

Figura 5 – Predição de preços de casas por meio de uma rede neural

rasas necessitariam de exponencialmente mais unidades ocultas para computar. (NG, 2017d)

A Figura 6 representa uma rede neural básica com duas camadas, três *inputs*, quatro neurônios na primeira camada e um único neurônio na segunda camada. Cada neurônio possui um peso *w* para cada *input* e um bias *b* compartilhado entre todos os inputs. Cada neurônio pode ser considerado uma unidade de processamento não linear para extração e transformação de características.(DENG; YU, 2014)

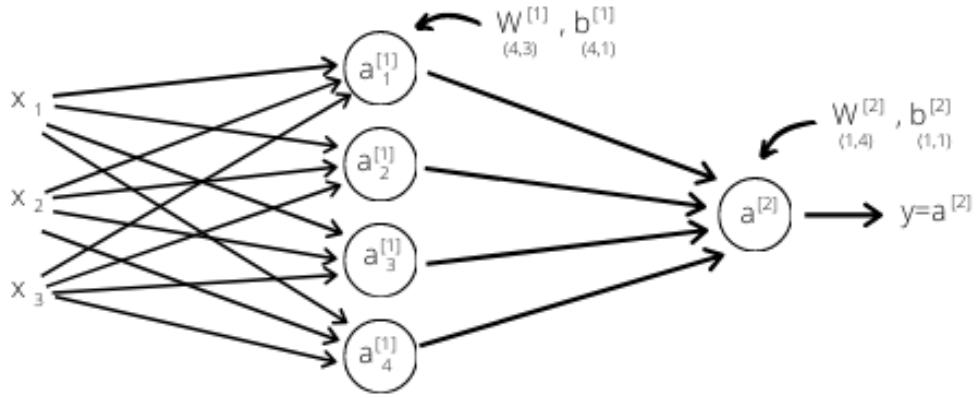

Figura 6 – Rede neural com duas camadas

Em cada neurônio, o *input* é multiplicado pelo seu respectivo peso e depois os valores são somados. Isso também pode ser representado como o produto escalar de dois vetores. Depois o bias é adicionado. Essa é a parte linear da função, mas para que a função consiga ser mais flexível, é necessário introduzir algumas não linearidades e isso é feito pela função de ativação. Basicamente esse valor de z que foi calculado é imputado em alguma função pré definida e esse é o valor que é passado para a próxima camada da rede neural. Esse processo está diagramado

na Figura 7.

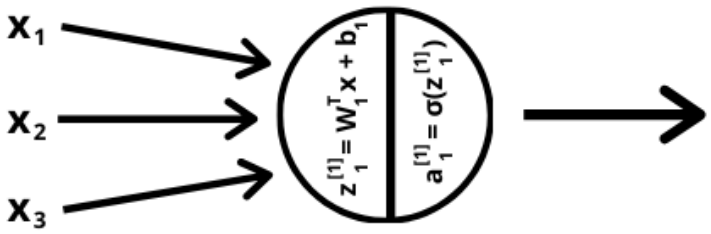

Figura 7 – Funcionamento de um neurônio

As equações 11 e 12 representam propagação para frente de informações na camada l de uma rede neural qualquer. Os valores são representados como matriz, pois assim é possível aproveitar melhor os recursos computacionais que linguagens como Python oferecem.

$$Z^{[l]} = W^{[l]}A^{[l-1]} + b[l] \tag{11}$$

$$A^{[l]} = g^{[l]}(Z[l]) \tag{12}$$

A função $g(z)$ é conhecida como função de ativação. Na Figura 8 tem-se as principais funções de ativação usadas em aprendizagem profunda. A função sigmoide é utilizada principalmente na última camada de redes neurais e são usadas para prever resultados que indicam a probabilidade de cada classe. Porém ela apresenta problemas como saturação dos gradientes, convergência lenta e um *output* não centrado em zero. A função tangente hiperbólica é preferível em relação a função sigmoide por ter melhor performance durante o treino, porém ela não conseguiu resolver o problema de gradientes desaparecendo(NWANKPA et al., 2018). Além disso, para valores muito grandes ou pequenos, a inclinação da curva é próxima de zero, diminuindo a velocidade de treinamento. A função ReLU é a função com maior velocidade de aprendizado, de acordo com LeCun, Bengio e Hinton (2015) e é também a função de ativação mais usada e de maior sucesso. Um detalhe interessante é que nesse caso, a derivada é sempre um ou zero[9]. Um problema da função ReLU é que ela facilmente sobre-ajusta a função. A última função, é a Leaky ReLU, uma versão de ReLU que introduz uma pequena inclinação na parte negativa, fazendo que sempre haja atualização dos parâmetros em todo o processo.

[9]No ponto zero a derivada não é definida, mas as chances de que o valor obtido sejam exatamente zero, são muito pequenas

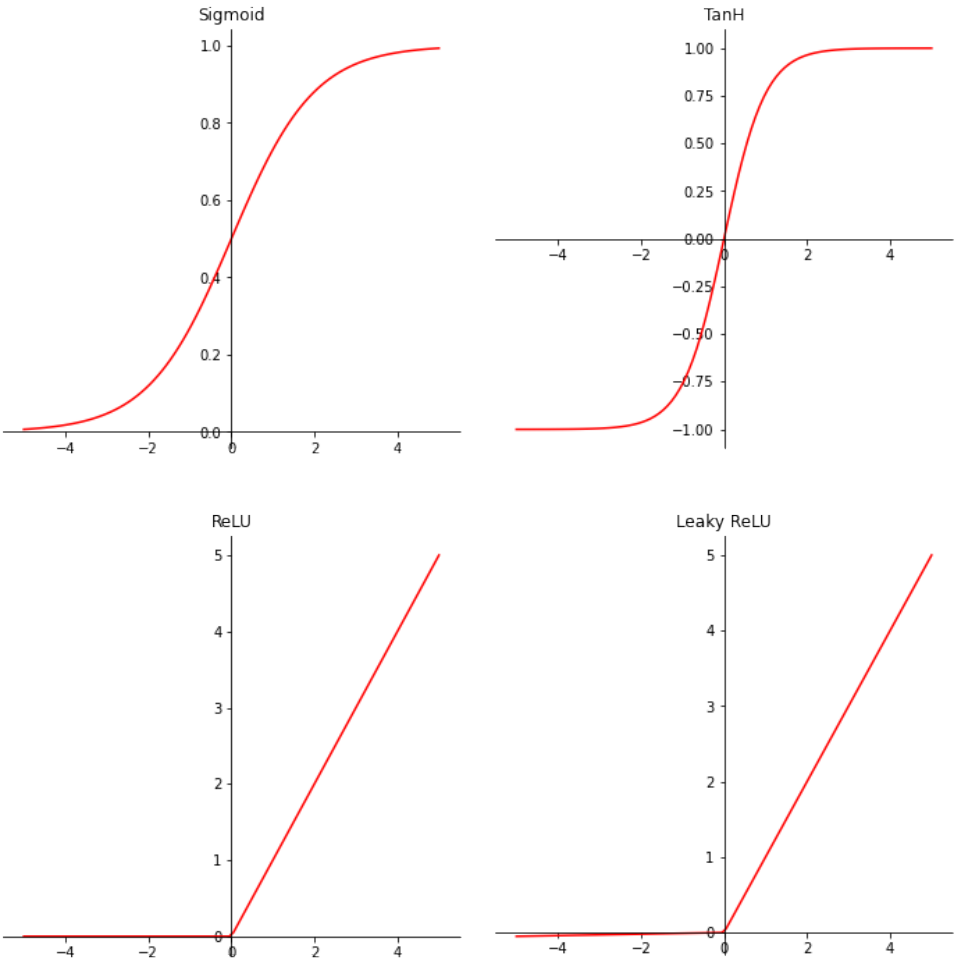

Figura 8 – Funções de ativação comuns

Um ponto importante sobre as redes neurais é que a informação não flui apenas do *input* para o *output*. Os parâmetros W e b são iniciados de maneira aleatória e no início não são nem um pouco úteis. Assim, é necessário que exista um guia de como ajustá-los. O algoritmo de *backpropagation* é justamente esse guia. De maneira intuitiva, a ideia desse algoritmo é responder a pergunta: dado o erro obtido (função perda) como ajustar os parâmetros de modo que o resultado seja um pouquinho melhor?

Em uma definição mais formal, o algoritmo de *backpropagation* computa o gradiente da função perda em relação aos pesos e bias de uma maneira computacionalmente eficiente, permitindo seu uso para o treinamento de redes neurais.

2.7 **Redes Neurais Recorrentes**

Redes neurais recorrentes ou RNNs são uma família de redes neurais utilizadas para processar dados sequenciais. De acordo com Ng (2017c), elas são uma alternativa às redes

neurais tradicionais para suprir dois problemas desse tipo de modelo:

- O modelos tradicionais não performam bem com *inputs* e *outputs* de tamanhos diferentes em cada um dos exemplos.
- Não existe compartilhamento de *features* aprendidas em diferentes posições do texto. Ou seja, os pesos aprendidos em parte do modelo não são reaproveitados ao longo da arquitetura da rede neural. A mesma palavra é tratada de maneira diferente caso esteja na primeira posição do texto, ou na centésima posição.[10]

Graças a avanços nas arquiteturas e modos de treinar RNNs, esse modelo tem provado ser muito útil em tarefas como prever o próximo carácter em um texto ou a próxima palavra em uma sequência e inclusive em tarefas mais complexas. Por exemplo, depois de ler uma sentença uma palavra por vez, uma rede *encoder* pode ser treinada de modo que seu vetor no estado final seja uma boa representação do pensamento expressado na sentença (LECUN; BENGIO; HINTON, 2015).

A arquitetura padrão de uma RNN pode ser representada pela Figura 9, em que cada X representa o vetor correspondente a cada palavra, como ilustrado na Figura 10. Essa figura ilustra várias das características interessantes desse tipo de rede neural. Primeiro, os pesos se mantém constantes durante toda a sequência, existindo um compartilhamento de pesos entre as camadas. O número de camadas sequenciais corresponde ao número de entradas, ou no caso de um texto, o número de palavras. Outro ponto, é que cada camada da sequência t utiliza não apenas o *input* x_t correspondente para prever y_t, como também o valor de a_{t-1} que é passado da camada anterior. A predição não depende apenas do valor atual como também de todos os valores anteriores (em teoria).

Pensando em uma RNN usada para processar textos, existem duas opções, cada carácter como um *input* único ou cada palavra como *input*, essa escolha depende da aplicação. A Figura 10 representa a divisão no caso de cada palavra sendo usado como *input*. Na verdade, não é a palavra que alimenta a rede neural, mas sim sua representação vetorial, que pode usar tanto uma representação do tipo *one hot encoding* quanto outra representação mais complexa.

As transformações que ocorrem em cada uma das camadas podem ser representadas pelas equações em 13 e 14. Observa-se que existem diversos conjuntos de pesos e bias em

[10] Uma outra arquitetura que também consegue suprir esse *gap* são as redes neurais convolucionais, que são tradicionalmente usadas para processamento de imagens e video, mas que já foram usadas com sucesso em tarefas de processamento de linguagem natural. As redes neurais convolucionais possuem filtros, que podem ser considerados como detectores de *features* que são compartilhados entre os atributos do *input*, permitindo que o for aprendido em uma parte do *input*, no caso, um texto, seja usado em outros momentos.

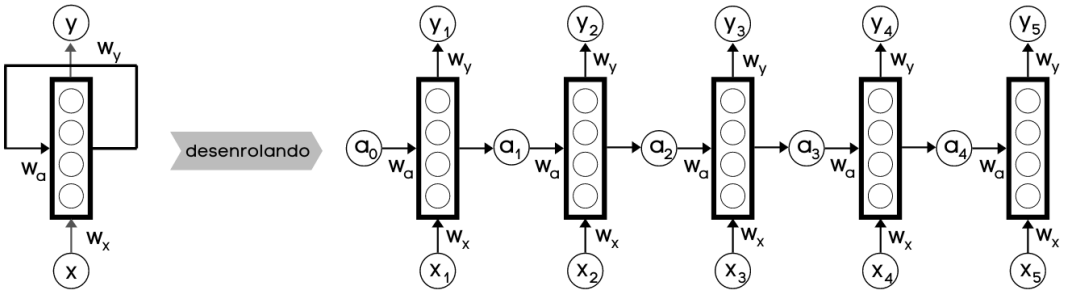

Figura 9 – Esquematização de uma RNN básica

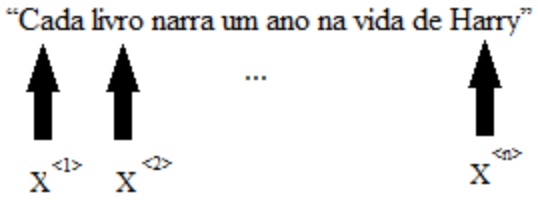

Figura 10 – Inputs em uma RNN

cada uma das camadas que são responsáveis por diferentes transformações. W_{aa} por exemplo é quem transforma a ativação proveniente da camada anterior, enquanto W_{ax} transforma o *input* da camada atual. Ao adicionar b_a aos dois valores calculados, obtém-se a ativação $a^{<t>}$ que será tanto passada para a próxima camada como usada para calculas o *output* dessa camada. W_{ya} é quem irá transformar a ativação no valor final calculado por essa unidade.

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}X^{<t>} + b_a) \tag{13}$$

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y) \tag{14}$$

Ter uma predição que depende tanto da palavra atual quanto das palavras anteriores, que seriam o contexto, é um grande ganho. Porém esse modelo ainda possui algumas fraquezas em sua arquitetura:

- Não leva em consideração os valores posteriores
- Memória curta

Na teoria, RNN grandes o suficiente deveriam ser capazes de gerar sequencias de complexidade arbitraria. Na pratica, RNNs padrão não são capazes de guardar informações sobre

inputs passados por muito tempo (KOLEN; KREMER, 2001). Um dos problemas disso, de acordo com Graves (2013), é que isso pode levar a instabilidade na geração de sequências, pois se as previsões são feitas apenas com base nos últimos inputs, o modelo tem pouca oportunidade de se recuperar dos erros do passado. Uma memória mais longe poderia ter um efeito estabilizador, pois o modelo poderia usar informações antigas para fazer as previsões.

2.7.1 *Long Short Term Memory* - LSTM

De modo a sanar o problema da memória curta, Hochreiter e Schmidhuber (1997) sugeriram a célula LSTM. Desde esse primeiro artigo, LSTMs foram alteradas e popularizadas por muitos pesquisadores (GERS; SCHMIDHUBER, 2000; GERS, 2001). Aqui será apresentado uma LSTM com um portão de esquecimento (*forget gate*), já que normalmente é assim descrito na literatura, embora seja possível encontrar variações em relação ao modelo apresentado neste trabalho. As equações que descrevem o LSTM são:

$$\tilde{c}^{<t>} = \tanh(w_c[a^{<t-1>}, x^{<t>}] + b_c) \tag{15}$$

$$\Gamma_u = \sigma(w_u[a^{<t-1>}, x^{<t>}] + b_u) \tag{16}$$

$$\Gamma_f = \sigma(w_f[a^{<t-1>}, x^{<t>}] + b_f) \tag{17}$$

$$\Gamma_o = \sigma(w_o[a^{<t-1>}, x^{<t>}] + b_o) \tag{18}$$

$$c^{<t>} = \Gamma_u \cdot \tilde{c}^{[t]} + \Gamma_f \cdot c^{<t-1>} \tag{19}$$

$$a^{<t>} = \Gamma_o \cdot \tanh(c^{<t>}) \tag{20}$$

Cada célula LSTM possui um valor $c^{<t>}$ que representa seu estado, o termo $\tilde{c}^{<t>}$ pode ser considerado um candidato a substituir esse valor. Os termos indicados por Γ são conhecidos como *gates* ou portões e são os que definem se realmente ocorrerá essa substituição. Cada um desses portões varia entre 0 e 1 pois são calculados usando uma função sigmoide, indicando o quanto de informação será passada para a próxima célula LSTM. Um valor de zero significa que nada passará, enquanto um valor de um indica que tudo vai passar. Γ_u é o portão de atualização

que indica se o valor de $\tilde{c}^{<t>}$ será adicionado ao novo valor de $c^{<t>}$. Já Γ_f é o portão de esquecimento que indica se o valor de $c^{<t-1>}$ será esquecido (NG, 2017b).

A Figura 11 representa as operações que ocorrem dentro de uma célula LSTM. Como dito por Ola (2015), o estado da célula é como uma esteira transportadora, que vai até o final da cadeia com apenas umas poucas interações lineares é muito fácil que a informação passe inalterada. A parte de remover ou adicionar informações é controlada pelos portões, como dito anteriormente.

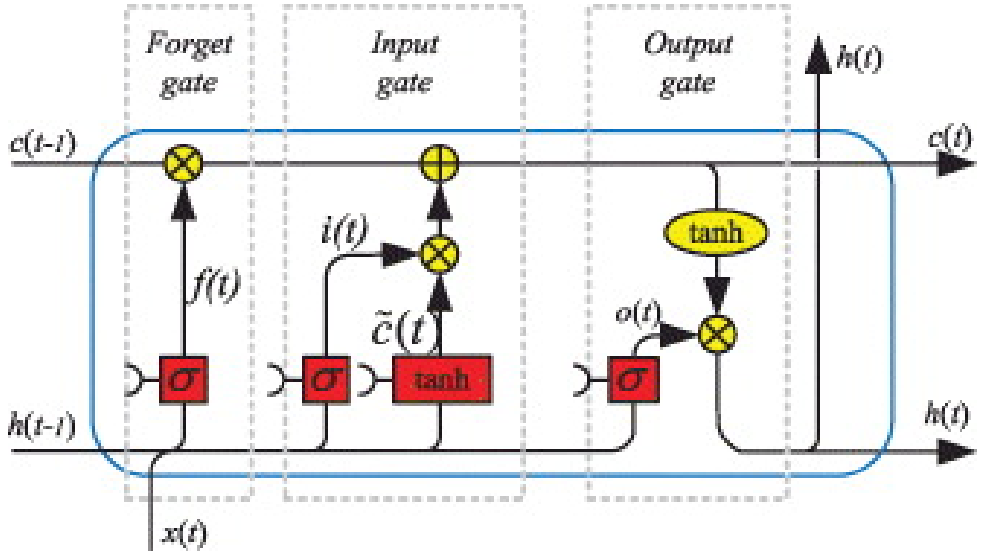

Figura 11 – Arquitetura LSTM com portão de esquecimento

2.7.2 Arquiteturas de RNNs

Os modelos apresentados na Figura 9 funcionam apenas para problemas em que T_x seja equivalente a T_y, o que nem sempre é o caso. Assim temos outras alternativas ao modelo padrão de RNN que podem ser usadas de acordo com o problema escolhido. Na Figura 12 são apresentados outros tipos de arquiteturas para RNNs.

O primeiro modelo, *one to one*, representa uma rede neural normal, descrita na seção 2.6. Outro tipo de arquitetura é *one to many* que começa a partir de um *input* único e normalmente a predição gerada é utilizada como input da próxima camada. Um exemplo de uso seria a geração de música.

A arquitetura *many to one*, que é a que será usada no presente trabalho, recebe vários *inputs* para no final gerar um único *output* que em geral será uma classe. Um exemplo de uso dessa arquitetura é para análise de sentimentos.

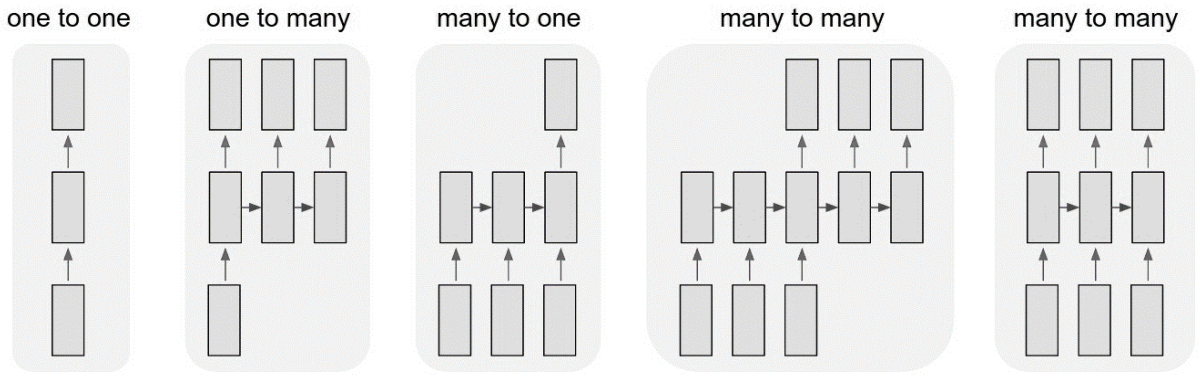

Figura 12 – Tipos de arquiteturas para RNN

Finalmente, a arquitetura *many to many* pode ter dois formatos, o primeiro já exemplificado em que $T_x = T_y$, porém também é possível existir um modelo *many to many* em que $T_x \neq T_y$. Nesse caso a primeira parte é chamada de *encoder* e a segunda de *decoder*. Esse modelo pode ser usado para tradução automática(NG, 2017a).

3 METODOLOGIA

Neste capitulo estão descritos todo o processo de obtenção, análise e transformação dos dados, assim como a metodologia adotada nos experimentos e a parametrização dos modelos.

3.1 Obtenção dos dados

A base utilizada foi obtida por meio de uma técnica *web-scrapping* da rede social GoodReads[1], cujo foco são os livros. Cada livro possui uma página de onde é possível retirar diversas informações tais como autor, idioma, número de páginas e descrição. A raspagem de dados foi facilitada por existir uma API[2] do próprio site, permitindo que os dados fossem facilmente acessados. Para acessar a API é necessário primeiro criar uma conta no site e obter uma chave de desenvolvedor (um requisito do método a ser utilizado). O método book.show permite obter informações sobre livros com base no id Goodreads do livro. Apesar da documentação da API informar que é possível obter respostas tanto em XML[3] quanto JSON[4], nas datas de realização das extrações, o método que retornava JSON não estava disponível, rasão pela qual foi usada a resposta XML. A resposta XML inclui informações de um livro como reviews populares, título, autor, etc. É importante destacar que o acesso é apenas aos metadados que são de propriedade do site do Goodreads, portanto alguns livros podem ter todas as informações presentes no site, mas não disponibilizadas na API, já que o site Goodreads não possui a licença para distribuir alguns metadados via API.

Importante detalhar o modo como foram selecionados os livros a serem utilizados na análise. A primeira tentativa consistiu em criar um *scrapping* com o objetivo de armazenar *links* existentes no site. Foi criado um *script* utilizando a linguagem Python[5]. A partir da página inicial <https://www.goodreads.com/> foram recolhidos todos os *links* presentes no código de cada página e armazenado em uma base de dados usando o SQLite[6]. Foi escolhido utilizar um banco da dados para garantir que as extrações pudessem ser realizadas em ocasiões diferentes.

---

[1] <https://www.goodreads.com/>

[2] *Interface de Programação de Aplicações* <https://pt.wikipedia.org/wiki/Interface_de_programaç~ao_de_aplicaç~oes>

[3] *Extensible Markup Language* <https://pt.wikipedia.org/wiki/XML>

[4] *JavaScript Object Notation* <https://pt.wikipedia.org/wiki/JSON>

[5] **Python** é uma linguagem de programação de alto nível, interpretada, de *script*, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte.

[6] **SQLite** é uma biblioteca em linguagem C que implementa um banco de dados SQL embutido.

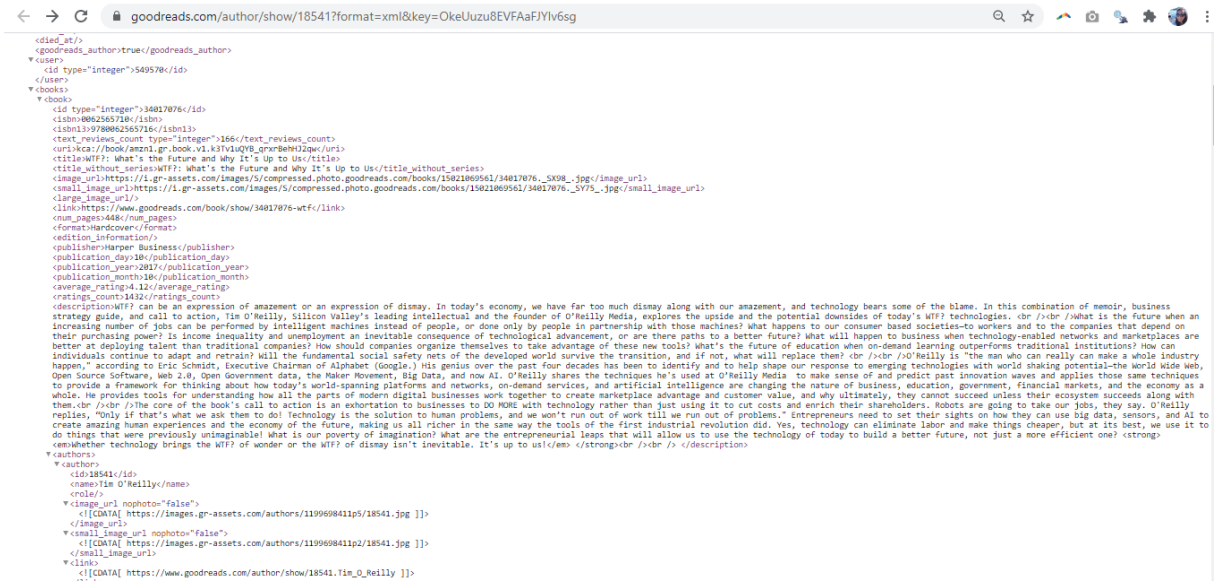

Figura 13 – Objeto XML contendo as informações do livro

O SQLite foi utilizado por já existir o módulo em versões do Python superiores a 2.5 sem a necessidade de instalação de dependências adicionais.

Os próximos *links* a serem raspados foram selecionados por meio de uma consulta ao armazenamento da base de dados e selecionando o primeiro item marcado como não recuperado. No total foram raspadas cerca de 2.600 páginas, coletando cerca de 80.000 links, dos quais 10% correspondiam a endereços de livros, que foram identificados pelo caminho */book/show/*. O caminho das páginas livro seguem o padrão */book/show/[id]-[title]*. Assim, por meio de uma expressão regular, foi extraído o id Goodreads necessário para a utilização do método da API *book.show*. Alguns *links* correspondiam a mesma id, pois foram coletados tanto *links* da página principal do livro quanto de itens como dúvidas ou *reviews*, que seguem o mesmo caminho de */book/show/*, então no total obteve-se pouco mais de 8.000 *links* válidos para realizar a extração de dados dos livros.

Isso resultou em uma base muito desbalanceada, com alguns gêneros com muito mais exemplares, o que poderia ocasionar em um enviesamento das predições em favor de gêneros com mais quantidade. Portanto, foi necessário pensar em uma alternativa para obter uma base mais equilibrada.

Assim, ao invés de buscar aleatoriamente, optou-se por uma nova abordagem mais direcionada. Dessa vez o objetivo foi coletar todos 52 mil livros presentes na lista *Best Books Ever*[7]. Esse é um número muito elevado de páginas a serem buscadas, então os métodos utilizados

[7] <https://www.goodreads.com/list/show/1>

até o momento para realizar a raspagem não estavam satisfatórios, pois demoravam mais de uma hora para cada mil livros extraídos. Foi necessário, portanto, encontrar uma solução mais adequada.

O pacote Scrapy[8] foi a solução encontrada e permitiu que todos os dados fossem extraídos em cerca de 15 horas. A ferramenta foi escolhida pela facilidade de uso, já que as únicas configurações necessárias foram o *parser* HTML da página e o tempo de pausa entre cada uma das consultas, o restante já é configurado pelo próprio projeto do Scrapy. As informações coletadas foram:

- Titulo
- ISBN[9]
- Autores
- Editora
- Idioma
- Avaliação
- Número de avaliações
- Número de reviews
- Número de páginas
- Descrição
- Ano de Publicação
- Estantes[10]
- Contagem de estantes[11]

Assim como os *links*, as informações extraídas de cada livro foram armazenadas em um banco de dados. Porém agora existe uma relação mais complexa entre as informações, sendo necessário definir uma arquitetura para otimizar o armazenamento. O diagrama na Figura 14 esquematiza o modo como os dados foram armazenados.

[8] Um *framework open source* para extrair dados de sites <https://scrapy.org/>

[9] *International Standard Book Number* <https://pt.wikipedia.org/wiki/International_Standard_Book_Number>

[10] Estantes (ou *Shelves*) são a maneira que os livros são organizados no Goodreads. Uma estante pode tanto representar um gênero, quanto classificações como: lido, quero ler e outros...

[11] Cada usuário pode adicionar um livro em quantas estantes quiser. A contagem indica quantas vezes um livro foi adicionado em uma estante especifica

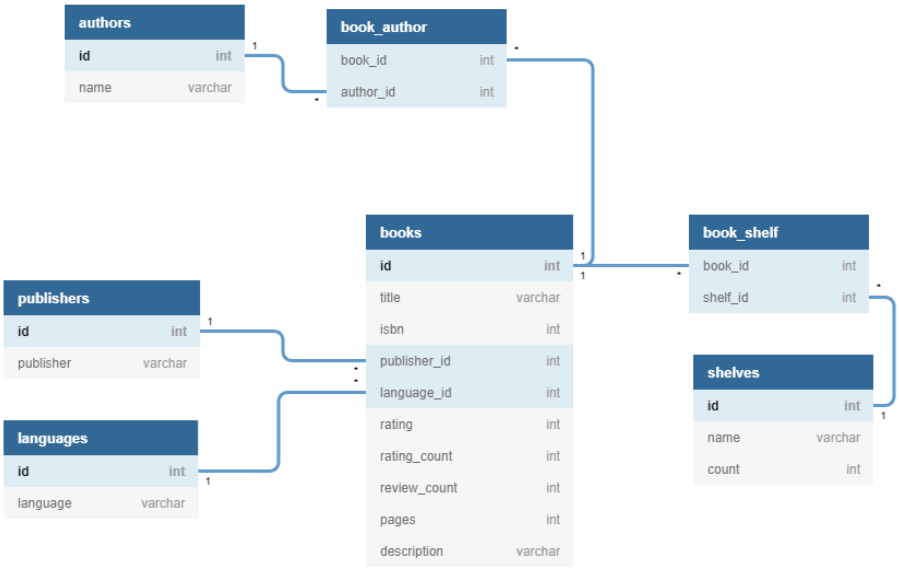

Figura 14 – Diagrama do banco de dados utilizado

3.2 **Análise e transformações dos dados**

Antes de começar qualquer analise foi feito um pré-tratamento na base, retirando livros que não estavam na língua inglesa[12] ou não possuíam descrição, já que foi definido que a classificação seria feita a partir da descrição. Optou-se por fazer a análise com livros em inglês, pois o estudo de processamento de linguagem natural está muito mais consolidado em aplicações utilizando o idioma inglês. Ao final, restaram o total de 41.677 livros para análise.

Importante ressaltar que a substituição da base inicial por uma maior, foi feita no início do processo de análise. Devido a natureza exploratória do trabalho, essas mudanças de escopo não foram consideradas como um ponto negativo, mas sim como uma maneira de avaliar diferentes resultados sobre outras perspectivas. Foi evitado mudanças apenas com o objetivo de distorcer o problema e facilitar de maneira excessiva a classificação.

3.2.1 Definição de Gênero

A escolha dos gêneros foi feita com base em dois pontos: popularidade do gênero (medida com base no número de vezes que os livros foram classificados com aquele gênero) e de modo a maximizar a separação entre os gêneros. Então, por exemplo, o gênero de romance histórico não foi considerado por possivelmente ter um grande número de livros também classificados

[12] Alguns dos livros são traduções, porém as informações eram referentes sempre a versão em inglês

como romance ou ficção histórica.

Como explicitado na seção 2.1, não há uma definição precisa do conceito de gênero. No presente trabalho, optou-se por utilizar o conceito de gênero apenas como rótulo, sem considerar questões como objetivo, por ser um primeiro estudo. Nessa pesquisa, objetiva-se usar o gênero como uma ferramenta de classificação para o usuário, sendo assim, resolveu-se utilizar o conceito de gênero definido pelo próprio usuário. Assim, um livro era considerado como pertencente a um gênero com base na quantidade de classificações do livro em determinada estante do Goodreads, classificação esta que depende unicamente do usuário.

O primeiro passo foi definir quais eram os gêneros mais populares. A Tabela 1 mostra a distribuição dos livros por gênero. Ficção é um gênero muito amplo, caso fosse escolhido, só faria sentido em um contexto de classificação binária entre ficção e não ficção, o que novamente voltaria para o contexto de classes desbalanceadas (28.886 livros de ficção contra 6.973 livros de não-ficção). O mesmo se aplica para não ficção, assim esses gêneros foram descartados da análise. Audiolivros é um tipo de gênero referente ao formato e não ao conteúdo do livro. Esse tipo de análise não faz sentido no contexto do trabalho, pois o objetivo é indicar se a descrição fornece pistas do gênero do livro pensado como uma maneira de categorizar assuntos, assim foi descartado. Adulto e jovem adulto são gêneros que dizem respeito ao público para qual o livro se destina, portanto é possível existir um livro de mistério para adultos, assim como um livro de mistério para jovens adultos. Portanto, no contexto do presente trabalho, foram descartados.

Tabela 1 – Quantidade de livros nos 10 gêneros mais populares.

Ficção	28.886
Fantasia	14.272
Romance	14.115
Jovem Adulto	10.770
Contemporâneo	9.365
Adulto	8.514
Mistério	7.754
Audiolivro	7.661
Ficção Histórica	7.182
Não Ficção	6.973

AO final, restaram apenas os seguintes gêneros:

- Fantasia
- Ficção Histórica
- Mistério

- Romance
- Contemporâneos

Para determinar qual era a melhor maneira de definir o problema (uma tarefa de classificação com um ou com múltiplos rótulos) foi feita uma análise exploratória inicial com o objetivo de determinar se era comum livros terem mais de um gênero. Os dados estavam dispostos em uma tabela com a informação de quantas pessoas haviam classificado e o valor máximo foi definido padrão. A partir disso, os outros valores foram divididos de modo que cada livro tivesse uma coluna para cada um dos gêneros, sendo que uma sempre havia o valor de 1 e às outras correspondiam a fração relativa ao valor máximo. A Figura 15 exemplifica o cálculo.

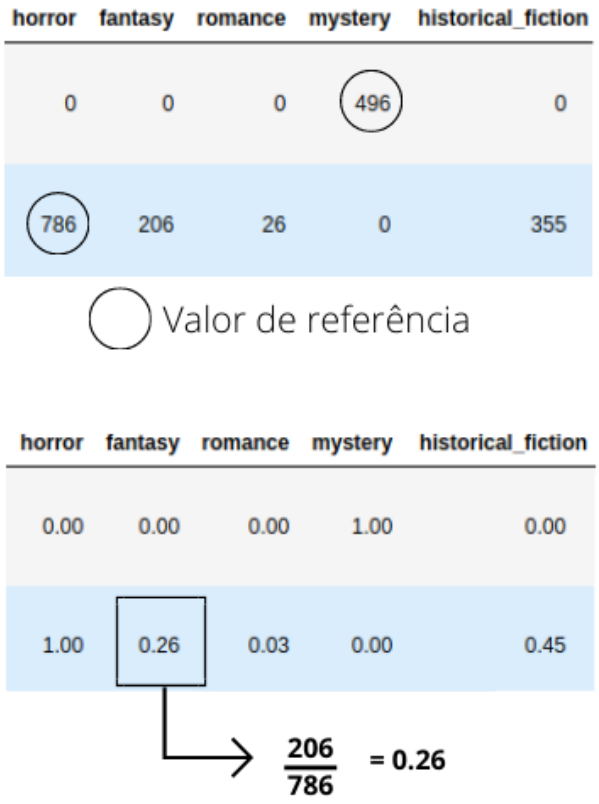

Figura 15 – Processo de cálculo do valor correspondente ao gênero

Depois disso, foi analisado o número de votos recebido pelo gênero principal, com o objetivo de avaliar se haveria livros com poucos votos, o que diminuiria a certeza daquela classificação.

A Tabela 2 permite visualizar algumas medidas de dispersão para o valor de referência. Assim, percebe-se que ao ampliar o número de livros na base foi necessário introduzir livros cuja classificação não era aparentemente tão confiável. Analisando melhor os dados do primeiro

Tabela 2 – Dispersão do número de votos máximos no gênero principal.

	1ª Análise	2ª Análise
Média	2046	425
Mínimo	95	2
25%	637	23
50%	1093	88
75%	2100	345
Máximo	62178	62178

quartil da 2ª análise, concluí-se que 10% da base possui menos de 4 votos por classificação. Apesar disso, em ambos os casos considerou-se que não havia a necessidade de fazer algum corte, sendo apenas um ponto de atenção ao analisar os resultados.

A partir disso, pode-se definir como gênero principal, aquele cujo valor na tabela é 1 (em outras palavras, o gênero com maior numero de votos). Isso permitiu determinar quantos livros de cada gênero estão sendo avaliados. A distribuição está representada no gráfico 16.

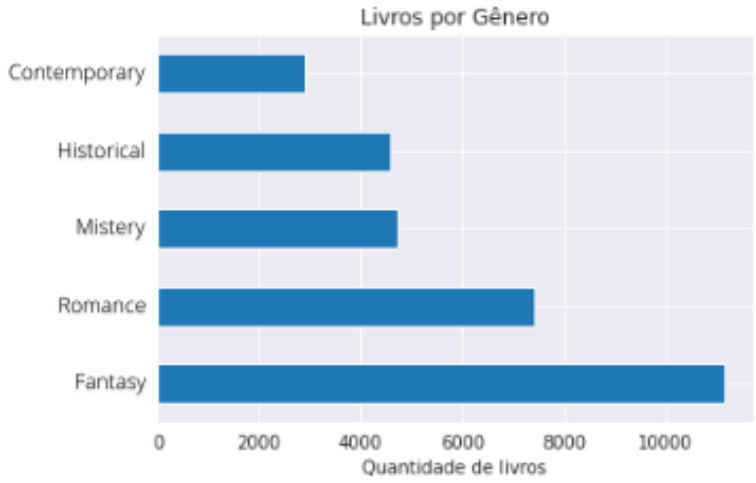

Figura 16 – Gráfico de quantidade de livros por gênero com contemporâneo

Com isso percebeu-se que o gênero contemporâneo era desnecessário nas análises, pois a quantidade de livros com o gênero principal contemporâneo era muito baixa. Optou-se então por excluir o gênero contemporâneo. Ressaltando que as quantidades na Tabela 1 representam quantidade de livros que foram marcados em cada gênero, mas não necessariamente que são os gêneros principais.

A nova distribuição está na figura 17. Alguns livros foram redistribuídos entre os gêneros restantes, outros porém foram excluídos da análise. Apesar de na seção 3.2 ser mencionado que

se utilizou 41.677 para a análise, após a definição de gêneros a serem classificados, o número foi reduzido para 29.143 livros, pois cerca de 12.000 livros não eram classificados em nenhum dos gêneros definidos. Esses 29.143 livros foram posteriormente divididos em uma base de treinamento e uma base de teste.

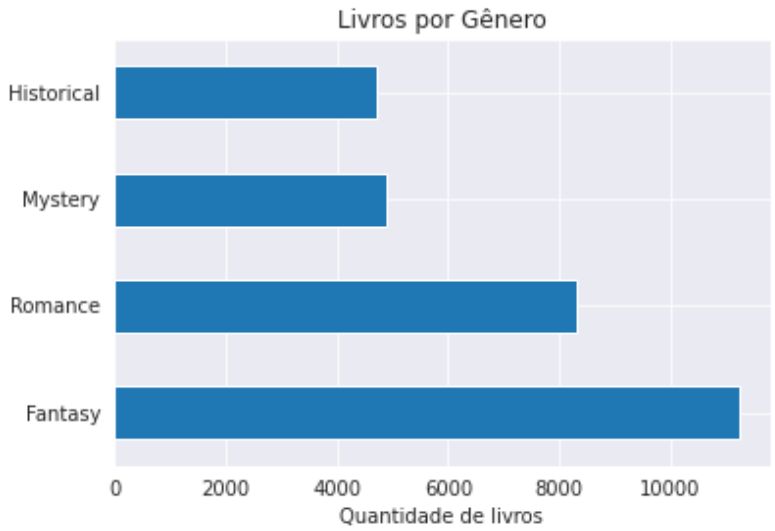

Figura 17 – Gráfico de quantidade de livros por gênero sem contemporâneo

Outro ponto que merece se destacado é que foi avaliada a quantidade de livros cujo número de votos por gênero principal era igual ao de outro gênero (tendo assim dois candidatos a gênero), para verificar se era um número relevante. Porém foram pouquíssimos os livros nessa situação, menos de 0.5% da base, que, em sua grande maioria, já estavam dentro do grupo de livros com menos de 4 votos.

Por fim, foi analisada a matriz de correlação entre os gêneros, para verificar se é válido manter todos ou se existe gêneros que estão relacionados. A correlação indica quão relacionada duas variáveis são. O coeficiente de correlação $\rho_{X,Y}$ entre duas variáveis X e Y ,com médias μ_x e μ_y e desvios padrões σ_X e σ_Y pode ser calculado por meio da Equação 21. O coeficiente de correlação pode variar de -1 a +1, sendo que -1 indica uma correlação inversa perfeita e +1 indica uma correlação perfeita. Do mesmo modo, valores próximos de 0 indicam pouca correlação.

$$\rho_{X,Y} = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} \tag{21}$$

Podemos ver pela Tabela 3 que o coeficiente de correlação é muito baixo, não passando de 0.36 em valores absolutos para nenhum dos valores. Isso significa que o número de votos que

cada gênero teve é independente entre si, fortalecendo a hipótese de que se deve manter todos os gêneros na análise.

Tabela 3 – Correlação entre gêneros.

	Fantasia	Ficção Histórica	Mistério	Romance
Fantasia	1.00	-0.36	-0.32	-0.34
Fic. Histórica	-0.36	1.00	-0.14	-0.25
Mistério	-0.32	-0.14	1.00	-0.30
Romance	-0.34	-0.25	-0.30	1.00

3.2.2 Preprocessamento das Descrições

Como a descrição do livro será a informação usada como preditor na rede neural foi necessário dedicar um tempo à sua análise. Antes de usar a descrição em um RNN, é necessário garantir que todas tenha o mesmo formato e comprimento. Usar *inputs* com um comprimento fixo facilita o treinamento, já que os pesos permanecem mais estáveis. Assim, é necessário primeiro determinar qual o comprimento ideal. Para isso foram analisados os gráficos 18 e 19.

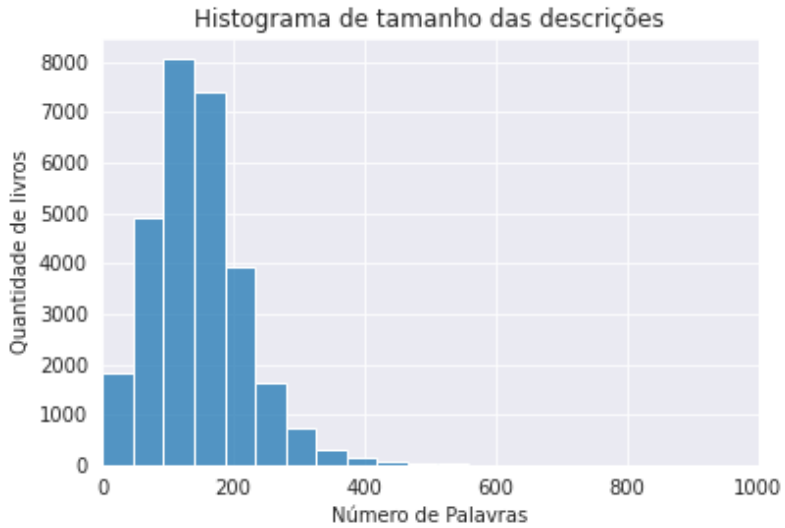

Figura 18 – Histograma do tamanho das descrições

Como podemos ver no gráfico 18, o histograma possui uma leve cauda para a direita, com um pico por volta de 200 palavras. Foi plotado também a distribuição cumulativa na Figura 19, para auxiliar na determinação do comprimento ideal. Escolheu-se por fim o valor 200.

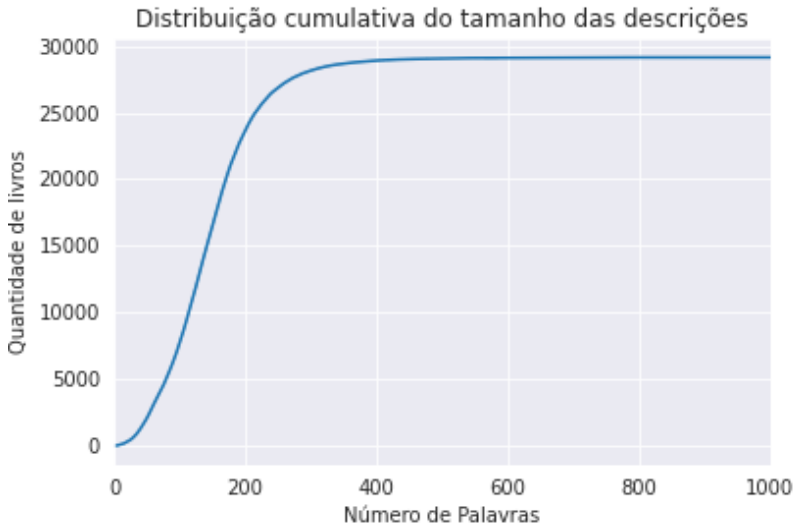

Figura 19 – Gráfico da distribuição cumulativa do tamanho das descrições

3.2.3 Treinamento da Rede Neural

Para realizar o treinamento da rede neural, foi escolhida a plataforma do Google Colaboratory (ou "Colab"), um serviço de nuvem gratuito e hospedado pelo Google. O *Colab* permite que qualquer pessoa possa executar códigos em python pelo navegador, não exigindo setup prévio e ainda provendo acesso a recurso de computação como GPUs, o que acelera bastante o processo de treinamento. Por ser gratuito, ele possui algumas limitações, como flutuações nos limites de uso, que são atualizados conforme a demanda. Um dos pontos negativos é que os tipos de GPUs disponíveis variam com o tempo, por se tratar de um recurso gratuito. Um notebook pode rodar por apenas 12 horas, porém no caso desse experimento, essa limitação não foi um impedimento.

Escolheu-se utilizar o TensorFlow[13] e sua API de alto nível, Keras[14] para realizar a criação e treinamento do modelo, por possuir uma interface fácil de lidar, sem abrir mão do desempenho já que os aplicativos são rodados em C++[15] e CUDA[16].

Como o rótulo e a descrição em mãos, deve-se prepará-los para alimentar o modelo.

[13]TensorFlow é uma biblioteca de código aberto para aprendizado de máquina aplicável a uma ampla variedade de tarefas. É um sistema para criação e treinamento de redes neurais.

[14]O Keras é uma biblioteca de rede neural de código aberto escrita em Python. Ele é capaz de rodar em cima de TensorFlow, Microsoft Cognitive Toolkit, R, Theano, ou PlaidML . Projetado para permitir experimentação rápida com redes neurais profundas, ele se concentra em ser fácil de usar, modular e extensível.

[15]C++ é uma linguagem de programação compilada multi-paradigma e de uso geral. Sendo bastante usada por seu grande desempenho e base de utilizadores.

[16]CUDA é uma plataforma de computação paralela, criada para que desenvolvedores possam usar de forma mais precisa e livre o alto potencial de processamento paralelo proporcionado por uma placa de vídeo.

O primeiro passo é a divisão em base de treinamento e validação, que foi feita com a função *train_test_split* da biblioteca de machine learning Scikit-Learn[17], dividindo a base em 80% treinamento e 20% teste.

Para funcionar como output do modelo, o rótulo passou por dois preprocessamentos. A primeira tranformação consistiu em renomear cada categoria como um número, transformação também conhecida como *label encoding* que pode ser feita com a classe Tokenizer do Keras. Em seguida, foi utilizado o *one-hot encoding*[18], convertendo o vetor com n observações de m classes diferentes em uma matriz binária $n \times m$, onde 1 representa um valor afirmativo e 0 o negativo.

As descrições também passaram por etapas de preprocessamento, antes de serem utilizadas como *inputs*. O primeiro passo foi utilizar a classe Tokenizer do Keras, para converter o texto em um vetor de números inteiros em que cada número representa uma palavra. O Tokenizer foi ajustado somente com base nos valores de treinamento, para evitar algum tipo de vazamento de dados, quando o modelo é exposto involuntariamente a parte da base de validação, o que resulta em acurácia da base de validação superestimada. Em seguida, tanto a base de treinamento, quanto a de validação foram transformadas utilizando o Tokenizer previamente ajustado. Por fim, como mencionado em 3.2.2, é recomendado que os *inputs* possuam todos o mesmo comprimento, assim foi utilizado o método *pad_sequences* que adiciona zeros ao inicio do vetor, caso seja necessário, com um valor máximo de comprimento (o valor é truncado caso exceda o máximo), resultando em uma matriz de n observações com um comprimento igual para todas.

A API do Keras fornece várias camadas já prontas, auxiliando na construção do modelo. Existem duas maneiras de compor modelos com Keras, composição sequencial e funcional. No caso do modelo utilizado, utilizou-se o composição sequencial em que modelos pre definidos são empilhados em um *pipeline* linear de camadas como em uma pilha ou em uma fila (GULLI; PAL, 2017).

Uma vez instanciado o modelo, ele é construído pela adição de camadas. A primeira camada é uma camada de *embeddings* cujo objetivo é transformar os índices dos vetores que são fornecidos à rede neural em vetores densos de tamanho fixo. Essa camada possui o argumento *trainable* passado como *False*. Isso significa que os parâmetros dela não são alterados durante o processo de treinamento. Como pesos, escolheu-se usar vetores pré-treinados, disponibilizados

[17] A Scikit-Learn é uma biblioteca de aprendizado de máquina de código aberto para a linguagem de programação Python.

[18] A frase *one hot encoding* vem da terminologia de circuitos digitais, em que descreve configurações de circuitos nos quais apenas uma parte pode ser positiva (*hot*)

no site <https://nlp.stanford.edu/projects/glove/>. Optou-se pelo modelo feito com base na Wikipedia 2014, que possui um vocabulário de 400.000 palavras e com 300 dimensões.

Nas camadas LSTM é necessário definir o número de unidades em cada um dos momentos da LSTM. Esse é um hiperparâmetro do modelo. Por meio de sucessivas iterações, foi possível definir valores que atendessem melhor a tarefa.

A camada de *dropout* aleatoriamente zera o valor o *input* de uma unidade com uma frequência definida pelo parâmetro passado na hora de criar a camada. No caso de uma camada criada com um parâmetro de 0.5, por exemplo, em 50% das vezes o *input* será zerado. Lembrando que essa camada só age durante o treinamento e não para realizar as predições(CHOLLET et al., 2015). O objetivo é adicionar uma regularização, tentando evitar o *overfit*.

Por fim, a ultima camada é uma camada densa que implementa as equações 11 e 12. Como a ativação dessa camada é uma função *softmax*, pode-se considerar que o resultado seria a probabilidade de que a descrição correspondesse a um livro de determinada classe. Portanto, para fazer a predição, basta selecionar o gênero correspondente ao maior valor do vetor de saída[19].

O código abaixo foi utilizado para a criação do modelo:

```
model = Sequential()
model.add(Embedding(vocab_size, 300, weights=[embedding_matrix],\
    input_length=200, trainable=False))
model.add(LSTM(50, return_sequences=True))
model.add(Dropout(0.3))
model.add(LSTM(30))
model.add(Dense(4, activation 'softmax'))
adam = Adam(lr=0.005)
model.compile(loss='categorical_crossentropy', optimizer=adam,\
    metrics=['accuracy'])
```

Na Figura 20 é possível ver os parâmetros em cada uma das camadas. E a Figura 21 ilustra como cada camada está interligada nessa arquitetura.

[19]Isso também está implementado no método *predict_classes* da instância de um modelo no Keras

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 750, 300)          30483300
_________________________________________________________________
lstm_5 (LSTM)                (None, 750, 50)           70200
_________________________________________________________________
dropout_2 (Dropout)          (None, 750, 50)           0
_________________________________________________________________
lstm_6 (LSTM)                (None, 30)                9720
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 124
=================================================================
Total params: 30,563,344
Trainable params: 80,044
Non-trainable params: 30,483,300
```

Figura 20 – Resumo do modelo criado

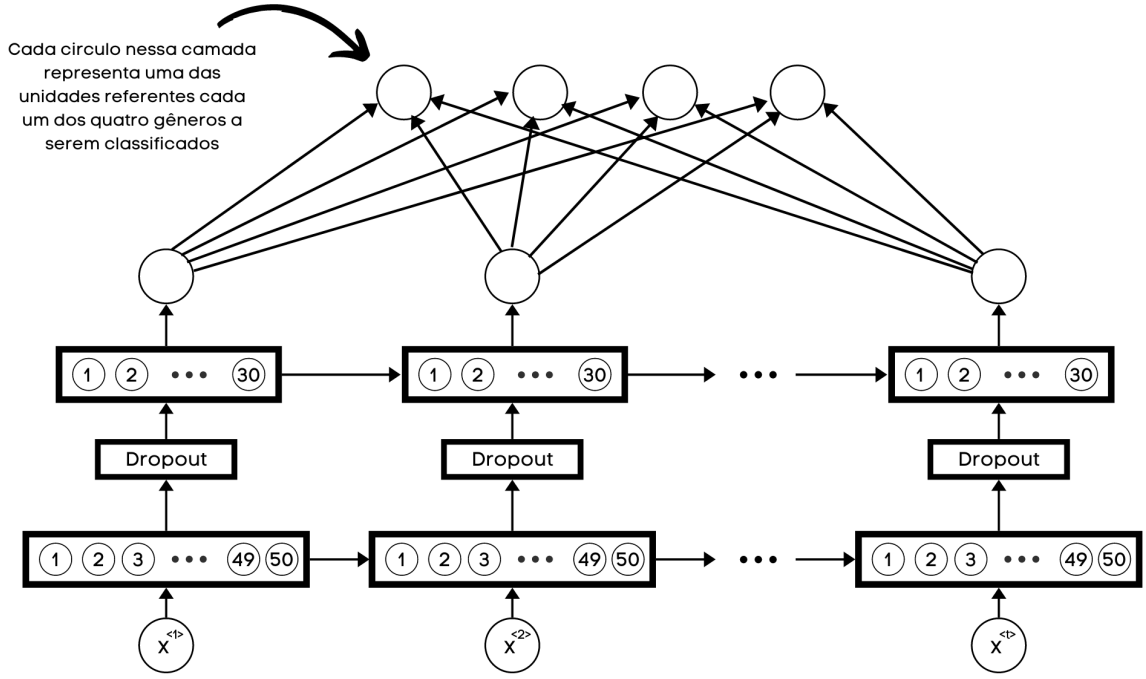

Figura 21 – Esquematização do modelo criado

3.2.4 Modelos para Comparação

Como forma de avaliar o desempenho do modelo por redes neurais, ele foi comparado com três modelos[20] amplamente usados em tarefas de classificação. A biblioteca Scikit-Learn, que já havia sido usada em outras partes do projeto, foi a escolhida para a implementação dos algoritmos de comparação. Os algoritmos utilizados foram os seguintes:

- **Regressão Logística:** Utilizou-se um regularizador do tipo L2 que adiciona o quadrado

[20] Inicialmente eram quatro modelos, mas o *random forest* apresentou um resultado horrível, classificando todos os livros como fantasia. Assim se optou por não adicioná-lo aos resultados

do valor dos pesos na função de custo e o parâmetro de regularização C com o valor de 1. O algoritmo para resolver o problema de otimização foi o *lbfgs*. Para que ele convergisse, foi necessário alterar o parâmetro *max_iter* para 200.

- **Máquina de Vetores de Suporte:** Foi usado um modelo com um *kernel* linear. Utilizou-se um regularizador do tipo L2 que adiciona o quadrado do valor dos pesos na função de custo e o parâmetro de regularização C com o valor de 1. A função perda utilizada foi a *squared hinge*. A estrategia usada para classificação multi-classes foi um contra o resto.
- **Naïve-Bayes:** Foi utilizado um classificador para modelos multinomiais. O parâmetro alfa para a suavização foi ajustado com o valor de 1. Como as classes estavam desbalanceadas, optou-se por usar o parâmetro *fit_prior* como *False*, ou seja, as probabilidades de cada classe foram baseadas em uma distribuição uniforme.

Todos os modelos foram ajustados utilizando atributos que foram preprocessados da mesma maneira, porém diferentes do utilizado na rede neural. Primeiro, não existia a necessidade de truncar o texto para que tivesse a mesma quantidade de palavras. Na verdade, o que é fornecido aos modelos é a frequência de cada palavra no documento. O *pipeline* de processamento pode ser visto abaixo.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer


count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(X_train)
X_test_counts = count_vect.transform(X_test)
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
X_test_tfidf = tfidf_transformer.transform(X_test_counts)
```

4 Resultados e Discussão

Ao término dos experimentos, obteve-se um modelo computacional para classificação de gênero literário utilizando redes neurais recorrentes. Assim como três outros modelos para *benchmarking*. A RNN poderia ser aperfeiçoada para se obter resultados melhores. Nesse capítulo serão apresentados os resultados obtidos e as análises do mesmo.

4.1 Resultados dos experimentos

Foram ajustados quatro modelos diferentes, descritos nas seções 3.2.3 e 3.2.4. A matriz de confusão de cada modelo pode ser vista na Figura 22.

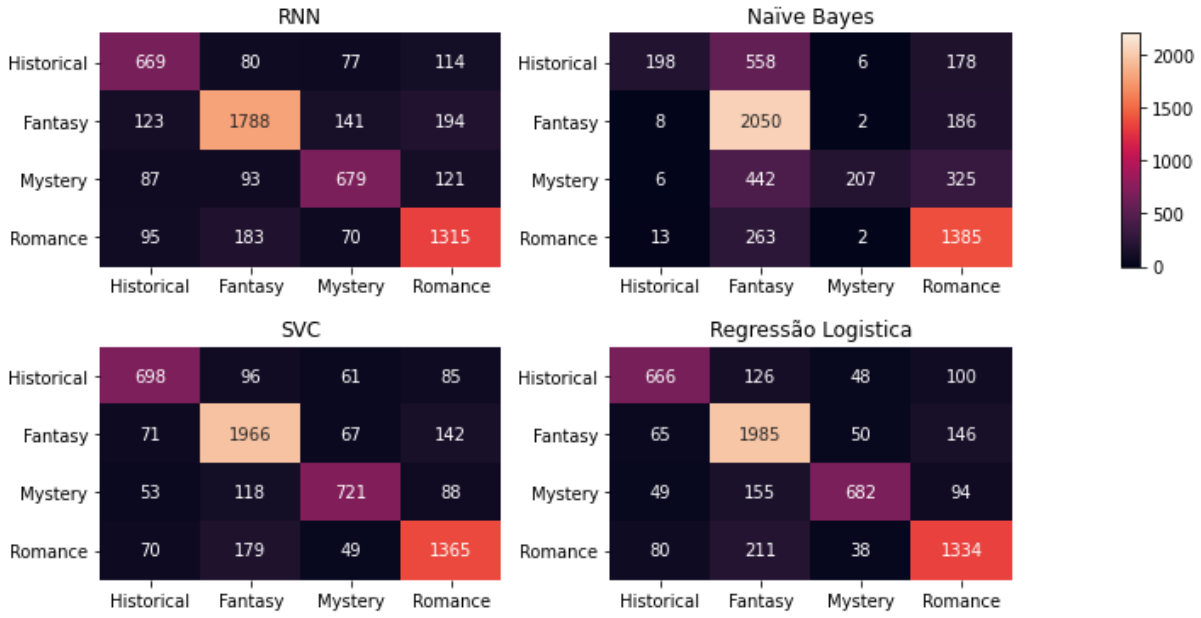

Figura 22 – Resumo do modelo criado

A Tabela 4 mostra o resultado dos experimentos utilizando acurácia como métrica.

Tabela 4 – Resultados dos modelos.

Modelo	Acurácia
RNN	77,8%
Naïve Bayes	65,9%
SVC	81,5%
Regressão Logística	80,1%

A rede neural apresentou resultado pouco inferior aos modelos de regressão logística e máquinas de vetores de suportes, sendo consideravelmente superior ao Naïve Bayes.

4.1.1 Análise de Erros

Além da métrica geral de erro, é necessário saber onde os erros estão ocorrendo. A Figura 22 já ajuda a ter algumas percepções, porém os valores são absolutos e calcular percentuais não é tão intuitivo. Um visual mais adequado está apresentado na Tabela 5, onde estão descritos por gênero o percentual de livros classificados em cada um dos gêneros. Assim, as colunas da tabela somam o total de 1.

Tabela 5 – Percentual de erro por gênero.

Gênero Previsto	**Gênero Atual** Fantasia	Ficção Histórica	Mistério	Romance
Fantasia	0.86	0.12	0.17	0.15
Fic. Histórica	0.04	0.72	0.05	0.05
Mistério	0.04	0.07	0.71	0.05
Romance	0.06	0.08	0.08	0.74

Como era de se esperar, a classe com o maior número de observações obteve o melhor resultado. Livros de fantasia são previstos como livros de fantasia em 86% das vezes. Dependendo do contexto, isso pode mostrar um classificador enviesado, ou poderia ser considerado apenas um classificador muito bom em prever esse tipo de gênero. Esse é um ponto a ser avaliado de acordo com o contexto de negócio que o problema está inserido. Um ponto que poderia ser avaliado é se livros de fantasia realmente representam a maior parte do conjunto de livros, ou se é algo relacionado a amostra escolhida. Isso também serve como guia para definir se esse desbalanceamento de classes deveria ser corrigido.

Uma possível fonte de erro poderia ser o truncamento da descrição do livro, conforme discutido na seção 3.2.2. Pois para algumas observações, parte da informação foi perdida. Para avaliar se existiu algum impacto, comparou-se a distribuição de número de palavras tanto para livros classificados incorretamente quanto para livros classificados corretamente. Pela Figura 23, é possível observar que as duas distribuições são muito parecidas, portanto o truncamento não pode ser considerado uma fonte de erro.

Na seção 3.2.1 foi mencionado que a inclusão de mais livros, poderia afetar a qualidade dos dados e, como consequência, a qualidade do modelo. Já que alguns dos livros adicionados

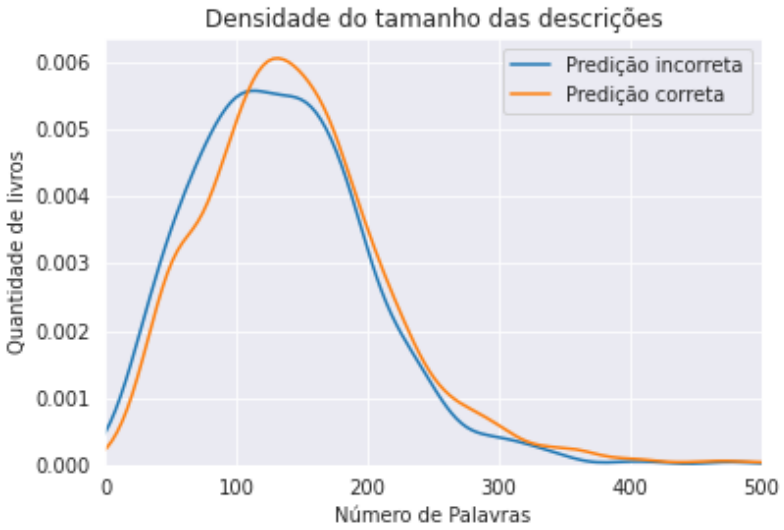

Figura 23 – Variação do número de palavras na descrição em predições corretas e incorretas

tinham poucos votos no gêneros principal. A Figura 24 confirma essa teoria, pois a distribuição de votos no gênero principal é bastante diferente nos casos de acerto e de erro na base de teste. Livros classificados de maneira incorreta, tem uma maior probabilidade de possuírem menos votos por gênero principal. Três possíveis hipóteses são:

- O livro na verdade pertencia a um outro gênero que foi descartado no processo de análise.
- O livro foi classificado de maneira incorreta, resultando em uma baixa popularidade e portanto possuindo um número baixo de votos.
- A descrição não se adequava ao gênero do livro, não atraindo o interesse de possíveis leitores e portanto possuindo um número baixo de votos.

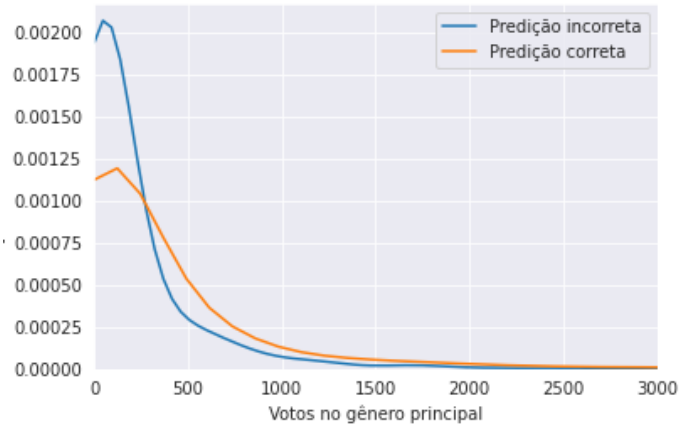

Figura 24 – Variação do número de votos em predições corretas e incorretas

4.2 **Considerações sobre o uso de RNNs**

Atualmente, redes neurais são modelos amplamente estudados e vem ganhando força em diversas aplicações. Os *frameworks* atuais, como o Keras, facilitam bastante a construção de modelos. Com apenas dez linhas de código, já se tem uma arquitetura tão complexa como a de uma RNN, com mais de uma camada, fazendo uso de camadas de dropout e o uso dos otimizadores considerados estado da arte no assunto, como o Adam. Porém, o fato de ser tão simples, faz com que o usuário não necessariamente tenha todo o conhecimento desejável para fazer um bom uso da aplicação. RNNs são modelos complexos e a realização desse trabalho, permitiu a comprovação das dificuldades enfrentadas na melhoria desse tipo de modelo.

Um dos pontos de dificuldade é que as redes neurais recorrentes não possuem uma maneira fácil de interpretabilidade do resultado, ao contrário dos outros modelos, que, por exemplo, podem utilizar um seletor de atributos para computar o teste χ^2 e gerar de modo simples uma lista com as palavras e combinações de palavras mais relevantes na classificação de cada gênero. Isso gera uma maior confiança na classificação, pois pela relação de termos em cada lista, consegue-se avaliar se aquilo está de acordo com o contexto esperado de cada gênero. No caso de RNNs, isso não é possível de ser feito. O fato de ser um modelo do tipo caixa preta dificultou o entendimento dos motivos de erro do modelo e, como consequência, dificultou a busca de alternativas para melhorá-lo.

Além disso, não existem práticas tão bem documentadas e definidas em relação ao processo de seleção de arquitetura e seleção dos hiperparâmetros do modelo, muitas vezes sendo um processo de tentativa e erro, em que o próximo passo era determinado basicamente pela intuição relacionada ao funcionamento desse tipo de modelo.

Um dos facilitadores foi que, ao contrário de aplicações práticas de redes neurais, o modelo em questão treinava relativamente rápido (20 segundos por *epochs*[1]), permitindo a testagem de muitas arquiteturas diferentes.

Um detalhe interessante é que o uso de uma arquitetura com apenas uma camada LSTM gerava mais sobre-ajuste na base do que o modelo final, com duas camadas de LSTM e uma camada de *dropout*. Isso indica que o efeito regularizador do *dropout* foi mais predominante do que a complexidade de uma camada adicional.

[1] Um *epoch* indica que o modelo realizou uma iteração completa sobre toda a base de treinamento

5 Conclusões

Nesse capítulo, estão descritas sugestões de temas que podem ser desenvolvidos a partir do que foi apresentado no trabalho, bem como considerações importantes acerca do mesmo.

5.1 Trabalhos Futuros

Após a análise dos resultados, percebe-se que existem muitas aplicações e pesquisas que podem ser desenvolvidas baseadas nesse trabalho. Estão listadas algumas sugestões:

- **Sistemas de classificação específicos:** No trabalho, foram classificados livros em geral, de gêneros que podem ser considerados amplos e com várias subdivisões. Uma sugestão de estudo seria verificar como essa metodologia se aplicaria em contextos mais especificas. Por exemplo, classificação de processos na área do judiciário, para fazer uma triagem dos processos, facilitando o julgamento. Ou até mesmo classificação de mercadorias na Receita Federal a partir de um texto de descrição do produto.
- **Algoritmo de recomendação de livros:** Já que a descrição pode ser utilizada para classificar os livros em gêneros, ela também poderia ser utilizada como base em um algoritmo de recomendação, sob a premissa que pessoas tendem a gostar de um determinado gênero de livro.
- **Avaliar o impacto de classificações erradas no contexto literário:** Como falado em 1, a venda de um livro pode ser impactada por sua classificação de gênero. Assim, seria interessante avaliar, do ponto de vista do negócio, o que uma classificação errada representa.

5.2 Considerações Finais

Redes neurais podem ser uma alternativa para tarefas de classificação de texto, como demonstrado nesse trabalho. No contexto do trabalho, o desempenho do modelo de rede neural foi pouco abaixo de dois dos modelos (regressão logística e máquinas de suporte de vetores) e superior ao Naïve Bayes. O resultado próximo aos dois modelos com melhor desempenho é um sinal positivo para futuras pequisas sobre rnn em contextos parecidos. Porém, como destacado em 4.2, outros fatores além das métricas devem ser levados em consideração, tais como a falta de interpretabilidade e a falta de boas-práticas bem definidas para esse tipo de modelo.

Referências

BAWARSHI, A.; REIFF, M. **Genre: An Introduction to History, Theory, Research, and Pedagogy**. Parlor Press, 2010. (Reference Guides to Rhetoric a). ISBN 9781602351714. Disponível em: <https://books.google.com.br/books?id=9xbASAAACAAJ>. Citado na página 2.

BOX G. E. P.; DRAPER, N. R. **Empirical Model-Building and Response Surfaces**. [S.l.]: John Wiley Sons, 1987. Citado na página 5.

BROWNLEE, J. **Long Short-Term Memory Networks With Python: Develop Sequence Prediction Models with Deep Learning**. Machine Learning Mastery, 2017. Disponível em: <https://books.google.com.br/books?id=m7SoDwAAQBAJ>. Citado na página 6.

BRUCE, A.; BRUCE, P. **Estatística Prática para Cientistas de Dados**. Alta Books, 2019. ISBN 9788550810805. Disponível em: <https://books.google.com.br/books?id=b0mvDwAAQBAJ>. Citado na página 9.

BURKOV, A. **The Hundred-Page Machine Learning Book**. Andriy Burkov, 2019. ISBN 9781999579517. Disponível em: <https://books.google.com.br/books?id=0jbxwQEACAAJ>. Citado na página 3.

CHANDLER, D. An introduction to genre theory. 01 1997. Citado na página 2.

CHOLLET, F. et al. **Keras**. 2015. <https://keras.io>. Citado na página 30.

DENG, L.; YU, D. **Deep Learning: Methods and Applications**. [S.l.], 2014. Disponível em: <https://www.microsoft.com/en-us/research/publication/deep-learning-methods-and-applications/>. Citado na página 11.

FEUER, J. Genre study and television. **Channels of discourse, reassembled**, Chapel Hill, NC: The U of North Carolina P, v. 2, p. 138–160, 1992. Citado na página 2.

GÉRON, A. **Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems**. O'Reilly Media, 2017. ISBN 9781491962299. Disponível em: <https://books.google.com.br/books?id=I6qkDAEACAAJ>. Citado na página 5.

GERS, F. Long short-term memory in recurrent neural networks (doctoral dissertation). **Beuth Hochschule für Technik Berlin**, 2001. Citado na página 16.

GERS, F. A.; SCHMIDHUBER, J. Recurrent nets that time and count. In: IEEE. **Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium**. [S.l.], 2000. v. 3, p. 189–194. Citado na página 16.

GRAVES, A. Generating sequences with recurrent neural networks. **CoRR**, abs/1308.0850, 2013. Disponível em: <http://arxiv.org/abs/1308.0850>. Citado na página 16.

GULLI, A.; PAL, S. **Deep Learning with Keras**. Packt Publishing, 2017. ISBN 9781787129030. Disponível em: <https://books.google.com.br/books?id=20EwDwAAQBAJ>. Citado na página 29.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Computation**, v. 9, n. 8, p. 1735–1780, 1997. Disponível em: <https://doi.org/10.1162/neco.1997.9.8.1735>. Citado na página 16.

JAMES, G. et al. **An Introduction to Statistical Learning: with Applications in R**. Springer New York, 2014. (Springer Texts in Statistics). ISBN 9781461471370. Disponível em: <https://books.google.com.br/books?id=at1bmAEACAAJ>. Citado 2 vezes nas páginas 7 e 9.

KOLEN, J.; KREMER, S. **A Field Guide to Dynamical Recurrent Networks**. Wiley, 2001. ISBN 9780780353695. Disponível em: <https://books.google.com.br/books?id=NWOcMVA64aAC>. Citado na página 16.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015. Citado na página 12.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature Cell Biology**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, maio 2015. ISSN 1465-7392. Citado na página 14.

MITCHELL, T. **Machine Learning**. McGraw-Hill Education, 1997. (McGraw-Hill international editions - computer science series). ISBN 9780070428072. Disponível em: <https://books.google.com.br/books?id=xOGAngEACAAJ>. Citado na página 5.

NG, A. **Different types of RNNs**. 2017. <https://www.coursera.org/learn/nlp-sequence-models/lecture/BO8PS/different-types-of-rnns>. Accessed 10/12/20. Citado na página 18.

NG, A. **Long Short Term Memory (LSTM)**. 2017. <https://www.coursera.org/learn/nlp-sequence-models/lecture/KXoay/long-short-term-memory-lstm>. Accessed 10/12/20. Citado na página 17.

NG, A. **Recurrent Neural Network Model**. 2017. <https://www.coursera.org/learn/nlp-sequence-models/lecture/ftkzt/recurrent-neural-network-model>. Accessed 10/12/20. Citado na página 13.

NG, A. **Why deep representations?** 2017. <https://www.coursera.org/learn/neural-networks-deep-learning/lecture/rz9xJ/why-deep-representations>. Accessed 10/12/20. Citado na página 11.

NG, A. **Machine Learning Yearning**. [S.l.: s.n.], 2018. <https://www.deeplearning.ai/machine-learning-yearning/>. Citado na página 5.

NWANKPA, C. et al. **Activation Functions: Comparison of trends in Practice and Research for Deep Learning**. 2018. Citado na página 12.

OLA, C. **Understanding LSTM Networks**. 2015. <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. Accessed 10/12/20. Citado na página 17.

POHLMAN, J. T.; LEITNER, D. A comparison of ordinary least squares and logistic regression. In: . [S.l.: s.n.], 2003. Citado na página 8.

SEDGEWICK, R.; WAYNE, K. **Algorithms, 4th Edition:**. Addison-Wesley, 2011. ISBN 9780321573513. Disponível em: <https://books.google.com.br/books?id=GhsNBQAAQBAJ>. Citado na página 3.

SHARDA, R.; DELEN, D.; TURBAN, E. **Business Intelligence e Análise de Dados para Gestão do Negócio - 4.ed.** Bookman Editora, 2019. ISBN 9788582605202. Disponível em: <https://books.google.com.br/books?id=Qr6xDwAAQBAJ>. Citado na página 9.

STAM, R. **Introdução A Teoria Do Cinema**. PAPIRUS, 2000. (Campo imagético). ISBN 9788530807320. Disponível em: <https://books.google.com.br/books?id=1Rt6cQY6bOkC>. Citado na página 2.

ZHANG, A. et al. **Dive into Deep Learning**. [S.l.: s.n.], 2020. <https://d2l.ai>. Citado 2 vezes nas páginas 4 e 7.