Capítulo 1

PLN: Das Técnicas Tradicionais aos Modelos de Deep Learning

Rafael Anchiêta, Francisco A. R. Neto, Jeziel C. Marinho, Raimundo Moura

Abstract

With the massive amount of data generated daily on the Web, researchers in the field of Natural Language Processing have focused on extracting useful information from unstructured data. This volume of data makes it impractical for anyone to manually process them in order to extract meaningful information, i.e., feelings, opinions, irony, hate speech, fake news, and others. The main objective of this short course is to introduce principles, traditional techniques, and tools in the field of NLP, developing models for binary classification tasks. The course focuses on practical activities using the Python language and libraries such as: NLTK, SpaCy, and Scikit-Learn. In the final part, some topics about Deep Learning will be discussed, including the BERT language model.

Resumo

Com a imensa quantidade de dados gerados diariamente na Web, pesquisadores da área de Processamento de Linguagem Natural (PLN) têm buscado extrair informações úteis de dados não estruturados. Esse volume de dados torna impraticável para qualquer pessoa processá-los manualmente a fim de extrair informações significativas, i.e., sentimentos, opiniões, ironia, discurso de ódio, fake news, entre outros. O objetivo principal deste minicurso é apresentar princípios, técnicas tradicionais e ferramentas da área de PLN, desenvolvendo modelos para tarefas de classificação binária. O curso é focado em atividades práticas usando a linguagem Python e bibliotecas, como: NLTK, SpaCy e Scikit-Learn. Na parte final, alguns tópicos sobre Deep Learning serão discutidos, incluindo o modelo de língua BERT.

1.1. Introdução

Processamento de Linguagem Natural (PLN) é uma vertente da Inteligência Artificial (IA) que ajuda computadores a entender, interpretar e manipular a linguagem humana. Em

termos simples, [Sarkar 2019] define linguagem natural como sendo uma linguagem desenvolvida e evoluída por humanos por meio do uso natural e da comunicação, em vez de construir e criar a linguagem artificialmente, como uma linguagem de programação.

A Comissão Especial de Processamento de Linguagem Natural (CE-PLN) da Sociedade Brasileira de Computação (SBC), estabelece que a área de PLN, também denominada Linguística Computacional ou, ainda, Processamento de Línguas Naturais, busca investigar, propor e desenvolver formalismos, modelos, técnicas, métodos e sistemas computacionais para resolver problemas relacionados à automação da interpretação e da geração da língua humana, como o inglês ou o português. A CE-PLN também descreve que as principais aplicações envolvem áreas, tais como: Tradução Automática de Textos, Sumarização Automática, Ferramentas de Auxílio à Escrita, Perguntas e Respostas, Categorização Textual, Recuperação e Extração de Informação, Análise Morfo-sintática e Análise Semântica, e Análise de Sentimentos.

Em um curso realizado em 2012 na Universidade de Stanford[1], Dan Jurafsky e Chris Manning classificaram as tarefas de PLN em:

- **Resolvidas**: detecção de *spams*, *POS tagging*, reconhecimento de entidades nomeadas;
- **Bom progresso**: análise de sentimentos, resolução de correferência, desambiguação lexical de sentidos, análise sintática (*parsing*), tradução automática, extração de informações;
- **Difíceis**: perguntas e respostas, paráfrases, sumarização, diálogos.

Hoje, cerca de 10 anos após o curso, essa classificação ainda é considerada válida, em especial para o português, uma língua considerada de poucos recursos (*low-resource language*). No entanto, o Centro de Pesquisa para Inteligência Artificial Avançada no Brasil[2] possui um desafio denominado NLP2 para produzir recursos e levar o Processamento de Linguagem Natural em Português para o estado-da-arte nas pesquisas mundiais.

Destaca-se que o foco principal deste minicurso é a tarefa de Análise de Sentimentos e Mineração de Opinião, tendo como insumo básico descrições textuais. De modo geral, a tarefa de Análise de Sentimentos (AS) pode ser definida como qualquer estudo feito computacionalmente envolvendo opiniões, sentimentos, avaliações, atitudes, afeições, visões, emoções e subjetividade, expressos de forma textual [Liu 2012, Liu 2015]. A tarefa de AS pode ser estruturada em três etapas: i) identificar as opiniões expressas sobre determinado assunto ou alvo em um conjunto de documentos; ii) classificar a orientação semântica ou a polaridade dessa opinião em positiva ou negativa; e iii) apresentar os resultados de forma agregada e sumarizada.

É importante mencionar que a maioria das pesquisas da área de AS são baseadas no nível de palavra, através da exploração de padrões linguísticos em tuplas, como <Caracteristica; PalavraOpinativa>. No entanto, métodos baseados no nível de conceito

[1]Natural Language Processing. Lecture Slides. Disponível em: https://web.stanford.edu/ jurafsky/NLPCourseraSlides.html

[2]C4AI: http://c4ai.inova.usp.br/pt/home-2/

têm surgido e precisam ser melhor investigados. Além disso, o uso de *Deep Learning* como as Redes Neurais Convolucionais [Poria et al. 2016] tornou-se muito importante para a evolução das pesquisas na área.

Além da classificação de sentimentos, outras tarefas têm, recentemente, ganho ênfase pela comunidade científica, tais como: detecção de *fake news* e identificação de discurso de ódio, por exemplo. A primeira foca em identificar conteúdo enganoso divulgado principalmente na Web, enquanto a segunda enfatiza a identificação e o combate a um tipo de linguagem que não é aceitável e é prejudicial as pessoas. Assim como na tarefa de classificar sentimentos, os primeiros trabalhos que se propuseram a lidar com notícias falsas e discurso de ódio focaram em abordagens baseadas em léxico, ou seja, um vocabulário pré-definido para ajudar na classificação dessas tarefas [Meneses Silva et al. 2021, Fortuna and Nunes 2018]. Posteriormente, os métodos evoluíram para abordagens baseadas em *deep learning* e modelos de língua, alcançando resultados superiores [Leite et al. 2020, Kaliyar et al. 2021].

No âmbito da Universidade Federal do Piauí (UFPI), pesquisadores do Laboratório de Processamento de Linguagem Natural (LPLN) têm desenvolvido estudos com o uso de PLN, a saber: i) identificação de elementos da Linguagem de Modelagem Unificada (UML) a partir de descrições textuais escritas em Português do Brasil [Anchiêta et al. 2013]; ii) metodologia para auxiliar a escrita de documentos de especificação de requisitos de sistemas [Soares and Moura 2015]; iii) estudo comparativo sobre métodos estatísticos de extração características em descrições textuais, usados para classificação de sentimentos [Anchiêta et al. 2015]; iv) definição da abordagem TOP(X) para inferir os comentários mais úteis sobre produtos e serviços [Sousa 2015]; v) variações da abordagens TOP(X), explorando as variáveis de entrada reputação do autor, quantidade de túplas <característica; palavra opinativa> e riqueza do vocabulário, além de variações dos modelos matemáticos utilizados: Sistema Fuzzy e Redes Neurais Artificiais [Santos et al. 2021]; e vi) estudo de técnicas de PLN para a resolução de problemas de regulação médica em planos de saúde [Magalhães Jr et al. 2019], cuja ideia explorou a descrição textual de prescrições de exames médicos para introduzir um fator de confiança e auxiliar a definição das regulações a serem aprovadas automaticamente.

Neste contexto, o objetivo principal deste minicurso é apresentar princípios, técnicas e ferramentas de PLN para a criação de modelos voltados para a classificação textos. O curso apresenta uma discussão de conceitos, com foco em atividades práticas usando a linguagem Python e bibliotecas que auxiliam no processamento de língua natural, como: NLTK (*Natural Language Toolkit*) [Bird 2006], SpaCy[3] e Scikit-Learn [Pedregosa et al. 2011]. Ao longo do curso, os trechos de código discutidos mostram o resultado da execução usando a tag '>>>'. Na parte final, alguns tópicos sobre *Deep Learning* serão discutidos, incluindo o modelo de língua BERT [Devlin et al. 2019].

De maneira geral, a tarefa de **classificação** consiste em atribuir o rótulo correto para uma determinada entrada. Por exemplo, definir se o texto de uma notícia é ou não irônico, para o caso de *classificação binária*. Ressalta-se que em tarefas básicas, cada entrada é considerada isoladamente das outras, e o conjunto de rótulos é definido com antecedência. Ademais, a tarefa de classificação tem algumas variantes interessantes,

[3]https://spacy.io/

por exemplo, na *classificação multiclasse*, cada instância pode receber vários rótulos; na *classificação de classe aberta*, o conjunto de rótulos não é definido com antecedência; e na *classificação de sequência*, uma lista de entradas é classificada em conjunto.

Adicionalmente, um classificador é denominado supervisionado se for construído com base em conjunto de dados de treinamento contendo o rótulo correto para cada entrada. A estrutura usada pela classificação supervisionada é mostrada na Figura 1.1, com as etapas de treinamento e predição.

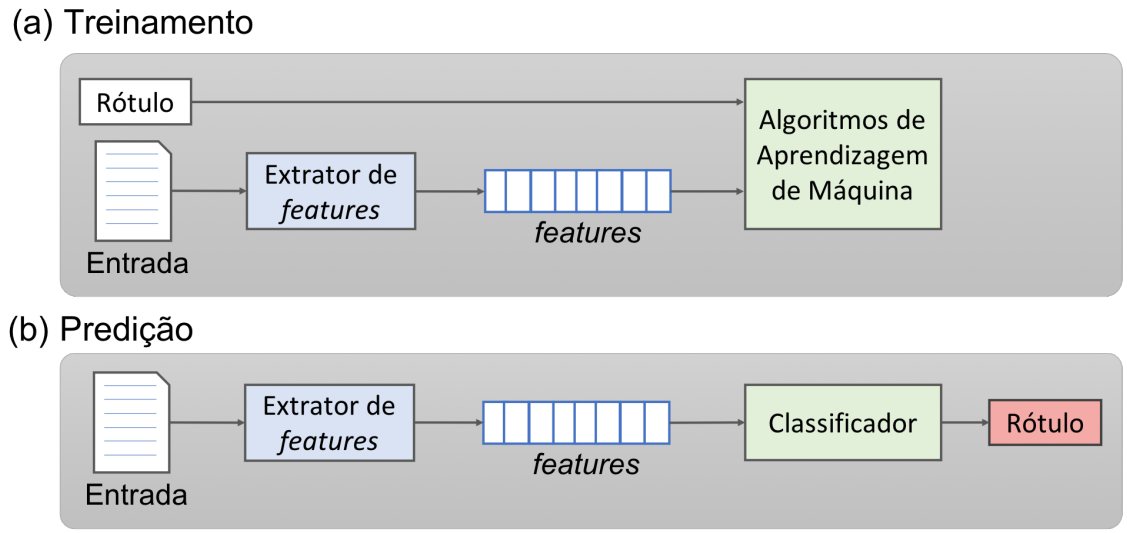


Figura 1.1: Classificação supervisionada. Adaptado de [Bird et al. 2009].

Observando a figura, percebe-se que durante a etapa de treinamento, um extrator de *features* é usado para converter cada texto de entrada em um vetor de *features*. Essas *features* devem capturar as informações básicas sobre cada texto e são usadas para classificá-lo. Já na etapa de predição, o mesmo extrator de *features* é utilizado para converter um novo texto no vetor de *features* correspondente e, em seguida, alimentar o classificador para prever o rótulo associado.

Além desta seção introdutória, o restante do minicurso será organizado em cinco seções: a seção 1.2 apresenta os principais conceitos e recursos usados na área de PLN, incluindo *Corpora* anotados e léxicos de várias tarefas, bem como ferramentas para análise de textos, como: *Part-Of-Speech Taggers* e *Stemmers*; a seção 1.3 discute modelos tradicionais usados no processo de classificação de textos; a seção 1.4 descreve brevemente dois modelos de *Word Embeddings*; a seção 1.5 apresenta modelos de *Deep Learning*, incluindo as redes neurais profundas e os modelos de língua (BERT); e, finalmente, a seção 1.6 conclui o trabalho e aponta perspectivas de trabalhos futuros.

Recomenda-se ainda três livros textos que podem ser usados como bibliografia básica em cursos da área: i) Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit [Bird et al. 2009], dos autores Steven Bird, Ewan Klein, and Edward Loper; ii) Inteligência Artificial - Uma Abordagem de Aprendizado de Máquina [Faceli et al. 2021], dos autores Katti Faceli, Ana Carolina Lorena, João Gama, Tiago Agostinho de Almeida e André C. P. L. F de Carvalho; e iii) Neural Network

Methods for Natural Language Processing [Goldberg 2017], do autor Yoav Goldberg.

Por fim, o procedimento de instalação das bibliotecas Python sugeridas está fora do escopo deste trabalho. No entanto, indicamos as seguintes urls `https://www.nltk.org/install.html`, `https://spacy.io/usage#installation` e `https://scikit-learn.org/stable/install.html`.

1.2. Recursos Léxicos

1.2.1. Conceitos e Definições

Esta subseção apresenta alguns termos, conceitos e definições amplamente utilizados nas diversas aplicações da área de PLN.

- **Corpus (plural: "Corpora"):** Um conjunto de dados linguísticos, sistematizados segundo determinados critérios, suficientemente extensos em amplitude e profundidade, de maneira que sejam representativos da totalidade do uso linguístico ou de algum de seus âmbitos, dispostos de tal modo que possam ser processados por computador. [Sanchez,1995] apud [Sardinha 2000];

- **Tokenização:** É o processo de separar um texto em *tokens* (palavras, números e símbolos). Alguns tokenizadores usam espaços, tabulações e quebras de linhas como separadores, enquanto que outros utilizam também os sinais de pontuação;

- **Normalização:** É o processo utilizado para garantir uma padronização mínima dos dados a serem processados. Por exemplo, conversão de todas as letras para minúsculas, correção de erros gramaticais comuns, remoção de imagens, URLs, *hashtags*, citações, espaços em excesso e pontuação duplicada, remoção de *stopwords* e letras repetidas, entre outros;

- **Stopwords:** São palavras que podem ser consideradas irrelevantes para o entendimento de um texto, ou seja, são palavras que geralmente têm pouco conteúdo lexical. Por exemplo, artigos, pronomes e preposições.

- **Stemmer:** É uma ferramenta utilizada para reduzir um termo ao seu radical através da remoção de desinências, afixos e vogais temáticas. *Stemming* é considerado um processo de normalização de texto;

- **Lematizer:** É uma ferramenta usada para agrupar diferentes inflexões e variantes de um termo para que possam ser analisadas como um único item, o lema. A lematização é uma operação mais poderosa e leva em consideração a análise morfológica das palavras, sendo que o lema deve obrigatoriamente existir em um dicionário;

- **Etiquetador (POS Tagger):** É uma ferramenta usada para atribuir a classe gramatical a cada uma das palavras de um texto, baseado tanto na sua definição quanto em seu contexto (palavras adjacentes) [Jurafsky and Martin 2009];

- **Analisador Sintático (Parser):** É uma ferramenta usada para analisar os modos de combinação de regras gramaticais (i.e., a função da palavra na oração), com a finalidade de gerar uma árvore que represente a estrutura sintática da oração analisada.

1.2.2. Corpora no NLTK

O NLTK disponibiliza diversos *Corpora* e coleções de livros, que podem ser baixados para a sua máquina, logo após a instalação do NLTK. Para mais informações, veja o capítulo 1 do livro NLTK [Bird et al. 2009], disponível em `https://www.nltk.org/book/ch01.html`. Aqui, discutiremos dois *Corpora* normalmente usados em aplicações para a língua portuguesa.

1.2.2.1. MacMorpho Corpus

Este *Corpus* foi desenvolvido pelo NILC/USP e contém 1 milhão de palavras etiquetadas com a classe gramatical. Normalmente, ele é usado para treinar etiquetadores (*POS Tagger*) para serem utilizados em aplicações. O código Python abaixo é usado para carregar o MacMorpho em memória e imprimir as palavras iniciais do *Corpus* com as respectivas etiquetas, sendo a etiqueta (tag) 'N' para substantivo e a tag 'V' para verbo.

```
from nltk.corpus import mac_morpho
print(mac_morpho.tagged_words())
>>> [('Jersei',  'N'), ('atinge', 'V'), ('média', 'N'), ...]
```

1.2.2.2. Stopwords Corpus

Este *Corpus* contém 2400 palavras consideradas *stopwords* em 11 línguas diferentes. O código abaixo é usado para carregar o recurso para a língua portuguesa em memória e comando *print(sw[0:30])* mostra as trinta primeiras palavras.

```
from nltk.corpus import stopwords
sw = stopwords.words('portuguese')
print(sw[0:30])
>>> ['de','a','o','que','e','é','do','da','em','um','para',
'com','não','uma','os','no','se','na','por','mais','as',
'dos','como','mas','ao','ele','das','à','seu','sua']
```

1.2.2.3. Carregando o seu próprio Corpus

O NLTK possui diversos métodos que podem ser usados para acessar informações de um *Corpus*. A Tabela 1.1 mostra algumas dessas funcionalidades.

A classe *PlaintextCorpusReader* é usada para acessar os métodos citados acima, considerando uma coleção de textos quaisquer. O código abaixo é usado para carregar os arquivos '.txt' existentes na pasta './corpus/'. O resultado da execução do trecho de código mostra que existem dois arquivos na pasta, além das palavras iniciais do *Corpus*.

```
from nltk.corpus import PlaintextCorpusReader
local = "./corpus/"
```

Tabela 1.1: Funcionalidades básicas do NLTK para acessar *Corpora*.

Métodos	**Descrição**
fileids()	os arquivos do *Corpus*
categories()	as categorias no caso de *Corpora* categorizados
words()	as palavras de todo o *Corpus*
words(fileid=[f1,f2,f3])	as palavras dos arquivos especificados
words(categories=[c1,c2,c3])	as palavras das categorias especificadas
sents()	as sentenças de todo o *Corpus*

```
corpusIronia = PlaintextCorpusReader(local,".*\.txt")
print(corpusIronia.fileids())
print(corpusIronia.words())
>>> ['tweetsIronia.txt', 'tweetsNaoIronia.txt']
>>> ['Quando', 'cheguei', 'a', 'Montemor', ',', 'gozavam',
 ↪  ...]
```

Existem outras formas de trabalhar com dados textuais na linguagem Python, com destaque para o uso da biblioteca Pandas[4]. O código abaixo é usado para carregar as informações de *tweets* sobre ironia, disponíveis no arquivo 'ironia.csv'. O comando *dados.info()* retorna as colunas de um registro do arquivo e o comando *dados['text']* retorna as informações do campo 'text' dos registros iniciais e finais.

```
import pandas as pd
file = "./corpus/ironia.csv"
dados = pd.read_csv(file, encoding='utf-8', decimal='.',
    sep=';', error_bad_lines=False)
print(dados.info())
print(dados['text'])
```

1.2.3. Etiquetadores

O NLTK possui quatros classes de etiquetadores (POS Taggers) que podem ser usados em aplicações de PLN, a saber: *DefaultTagger*, *UnigramTagger*, *BigramTagger* e *TrigramTagger*. A classe *DefaultTagger* atribui uma única etiqueta para todas as palavras. A classe *UnigramTagger* treina o modelo considerando as etiquetas de palavras únicas, enquanto as classes *BigramTagger* e *TrigramTagger* treinam os modelos considerando as etiquetas de sequências de duas e três palavras, respectivamente. O código abaixo é usado para treinar e testar três etiquetadores, usando as sentenças do *Corpus Mac_Morpho*, na proporção de 90%/10%. A parte final do código mostra-se um exemplo de etiquetagem de uma sentença, usando o *BigramTagger* que obteve 85,5% de acurácia.

```
import nltk
from nltk.corpus import mac_morpho
```

[4]https://pandas.pydata.org/docs/index.html#

```
# 10% para teste e 90% para treino
prop = int(0.1 * len(mac_morpho.tagged_sents()))
treino = mac_morpho.tagged_sents()[prop:]
teste = mac_morpho.tagged_sents()[:prop]
# Etiquetador Padrão
etiq1 = nltk.DefaultTagger('N')
print("BASIC Tagger: ", etiq1.evaluate(teste))
#Etiquetador UNIGRAM
etiq2 = nltk.UnigramTagger(treino, backoff=etiq1)
print("UNIGRAM Tagger: ", etiq2.evaluate(teste))
#Etiquetador BIGRAM
etiq3 = nltk.BigramTagger(treino, backoff=etiq2)
print("BIGRAM Tagger: ", etiq3.evaluate(teste))
frase = nltk.word_tokenize("O governo não pode dar educação
 ↪  porque a educação derruba o governo")
print(etiq3.tag(frase), end='\n')
>>> BASIC Tagger:  0.2079790656025594
>>> UNIGRAM Tagger:  0.837488915549528
>>> BIGRAM Tagger:  0.8548849825256899
>>> [('O', 'ART'), ('governo', 'N'), ('não', 'ADV'),
 ('pode', 'VAUX'), ('dar', 'V'), ('educação', 'N'),
 ('porque', 'KS'), ('a', 'ART'), ('educação', 'N'),
 ('derruba', 'V'), ('o', 'ART'), ('governo', 'N')]
```

1.2.4. Normalizadores

O processo de normalização de textos pode ser entendido como alguns passos de pre-processamento, incluindo a correção de erros gramaticais, transformação do texto para letras minúsculas, remoção de letras e/ou palavras duplicadas, remoção de acentos e *stopwords*, uso de *stemming* e/ou *lemas*, entre outros.

Os erros gramaticais podem ser identificados através do uso do corretor gramatical CoGrOO [Kinoshita et al. 2007], que verifica a colocação pronominal, concordância nominal e verbal, concordância entre sujeito e verbo, uso de crase, regência nominal e verbal, além de outros erros comuns da língua portuguesa escrita.

Com relação a *Stemmers*, o NLTK inclui diversos algoritmos facilmente disponíveis, por exemplo o Porter e Lancaster stemmers seguem suas próprias regras para remover afixos. O código abaixo é usado para instanciar e aplicar esses *stemmers* em uma determinada frase. Neste exemplo, a versão do *Porter Stemmer* utilizada é para a língua inglesa e, praticamente, não atua em palavras da língua portuguesa. Porém, existe o projeto *PTStemmer - A stemming toolkit for Portuguese in Python*[5], que implementa versões dos *Stemmers* Orengo, Porter e Savoy.

```
import nltk
texto="O governo não pode dar educação porque a educação
 ↪  derruba o governo."
```

[5]https://github.com/lsdr/ptstemmer

```
tokens = nltk.word_tokenize(texto)
porter = nltk.PorterStemmer()
rslp = nltk.stem.RSLPStemmer()
outPorter=[]
outRSLP=[]
for tok in tokens:
    outPorter.append(porter.stem(tok))
    outRSLP.append(rslp.stem(tok))
print(outPorter, '\n', outRSLP)
>>> ['O', 'governo', 'não', 'pode', 'dar', 'educação',
 'porqu', 'a', 'educação', 'derruba', 'o', 'governo']
>>> ['o', 'govern', 'não', 'pod', 'dar', 'educ', 'porqu',
 'a', 'educ', 'derrub', 'o', 'govern']
```

1.2.5. Analisadores Sintáticos (*Parsing*)

A classe *RegexpParser* do pacote *nltk.chunk* permite explorar estruturas gramaticais de textos. Além dessa classe, o NLTK possui diversas outras para tratar gramáticas livres de contexto (GLC)[6]. O código abaixo mostra uma gramática relativamente simples para identificar sintagmas nominais (SN) em textos, composto de um artigo seguindo de substantivo. Destaca-se que o etiquetador (*etiq3*) e o objeto *tokens* são os mesmos definidos nos exemplos anteriores.

```
from nltk.chunk import RegexpParser
tags = etiq3.tag(tokens)
analiseGramatical = RegexpParser(r"""
    SN:{<ART><N>}
    }<V>{
    """)
arvore = analiseGramatical.parse(tags)
arvore.draw()
```

Como resultado da execução do código tem-se a Figura 1.2. Assim, com o uso de gramáticas é possível encontrar elementos textuais nas descrições. Essa técnica é conhecida na literatura como identificação de padrões linguísticos, e tem sido explorada com sucesso na área de análise de sentimentos e mineração de opiniões sobre produtos ou serviços. [Anchiêta et al. 2013] usaram alguns padrões para identificar atributos e métodos em descrições de casos de uso da UML e [Sousa 2015] também utilizou padrões linguísticos para identificar aspectos de um produto e as palavras opinativas sobre tais aspectos. Na gramática, ele utilizou substantivos, adjetivos, verbos e advérbios para identificar esses elementos.

[6]GLC é uma gramática formal onde todas as regras de produções são da forma $A \rightarrow \alpha$, sendo A um símbolo não-terminal e α um regra de produção composta por terminais e não-terminais

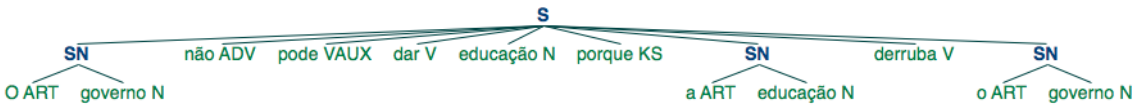


Figura 1.2: Árvore sintática parcial.

1.2.6. Outros Recursos

Existem algumas alternativas à biblioteca NLTK, como, por exemplo, as ferramentas spaCy[7], nlpnet [Fonseca and Rosa 2013], stanza [Qi et al. 2020], NLPyPort [Ferreira et al. 2019], entre outras. Aqui, focaremos na spaCy, pois ela é uma ferramenta robusta com modelos treinados para várias tarefas de PLN.

O spaCy é uma ferramenta que tem suporte a mais de 60 línguas, possuindo modelos treinados para tarefas de reconhecimento de entidades nomeadas, etiquetação morfossintática (*tagger*), análise de dependência (*dependency parsing*), lematização, entre outras. Para o português, o spaCy possui três modelos treinados que variam em tamanho e acurácia, quanto menor o modelo, menor a acurácia, são eles: `pt_core_news_sm`, `pt_core_news_md` e `pt_core_news_lg`, onde o primeiro é o menor e último o maior. Para mais detalhes sobre esses modelos, sugere-se a páginal oficial da ferramenta[8].

Um dos usos mais comuns da ferramenta é a extração de informações morfológicas, morfossintáticas e sintáticas de um texto. O código em Python abaixo ilustra um exemplo dessa tarefa. Neste exemplo, `token.text` é o texto original da sentença, `token.lemma_` é a forma canônica da palavra, `token.morph` é a informação morfológica da palavra `token.pos_` é a etiqueta morfossintática da palavra, `token.dep_` é a relação de dependência entre as palavras, `token.is_alpha` informa se a palavra é um texto e `token.is_stop` indica se a palavra é uma *stopword*.

```
import spacy

nlp = spacy.load("pt_core_news_sm")  # modelo
doc =  nlp("Justiça aceita denúncia")  # sentença
for token in doc:
    print(token.text, token.lemma_, token.morph,
     ↪  token.pos_, token.dep_, token.is_alpha,
     ↪  token.is_stop)

>>> "Justiça", "Justiça", "Gender=Fem|Number=Sing", "NOUN",
 ↪  "nsubj", "True", "False"
>>> "aceita", "aceito",
 ↪  "Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin",
 ↪  "VERB", "ROOT", "True", "False"
>>> "denúncia", "denúncia", "Gender=Fem|Number=Sing",
 ↪  "NOUN", "obj", "True", "False"
```

[7] https://spacy.io/
[8] https://spacy.io/models/pt

Outro uso comum da ferramenta spaCy é para a tarefa reconhecimento de entidades nomeadas cujo objetivo é identificar entidades nomeadas dentro e um conjunto de categorias pré-definidas, tais como: Pessoa, Localização, Organização, entre outras. O código abaixo exemplifica essa tarefa.

```
import spacy

nlp = spacy.load("pt_core_news_sm")   # modelo
doc =  nlp("Quixadá é uma bela cidade.")   # sentença
for ent in doc.ents:
    print(ent.text, ent.label_)

>>> 'Quixadá', 'LOC'
```

No exemplo, `ent.text` é o texto original e `ent.label_` é o rótulo da entidade reconhecida. Aqui, o spaCy identificou que a palavra `Quixadá` é uma localização.

Além dos exemplos acima, o spaCy permite criar regras a fim de melhorar ou adaptar o resultado obtido pelos modelos. No que segue, serão apresentados alguns algoritmos tradicionais de aprendizado de máquina aplicados em tarefas de PLN.

1.3. Classificação: Algoritmos tradicionais

A tarefa de **classificação de Textos** consiste em atribuir o rótulo de classe para uma determinada entrada. Por exemplo, definir se o texto de um *tweet* ou notícia é ou não irônico; se uma mensagem possui ou não discurso de ódio; decidir qual é o tópico de um artigo de notícias (esporte, tecnologia, política, ...), entre outros.

A seguir, apresenta-se um classificador para determinar se o texto de um *tweet* é ou não irônico, baseado no modelo mostrado na Figura 1.1. Os dados sobre os *tweets* irônicos e não irônicos foram disponibilizados pelos organizadores da tarefa IDPT (*Irony Detection in Portuguese*), que faz parte do evento IberLEF 2021 (*Iberian Languages Evaluation Forum*). A partir desses dados organizou-se dois arquivos: 'tweetsIronia.txt' e 'tweetsNaoIronia.txt', contendo 12.807 e 2.476 mensagens, respectivamente.

1.3.1. Classificador de *Tweets* Irônicos

O desenvolvimento de um classificador pode ser dividido nos seguintes passos:

1. **Carregar dados:** Inicialmente, deve-se carregar os dados na memória, atribuindo o rótulo de cada classe. Na parte final do código, o comando 'y, text = zip(*dados)' permite separar as classes e os textos nos objetos 'y' e 'text', respectivamente:

   ```
   dados = []
   with open('db/tweetsIronia.txt') as f:
       d = [l.strip().split('\n') for l in f.readlines()]
   for t in d:
       dados.append(['1']+t)
   ```

```
with open('db/tweetsNaoIronia.txt') as f:
    d = [l.strip().split('\n') for l in f.readlines()]
for t in data:
    dados.append(['0']+t)

y, text = zip(*dados)
print(collections.Counter(y))
>>> Counter({'1': 12807, '0': 2476})
```

2. **Definir os conjuntos de treino e teste:** O segundo passo é separar os dados nos conjuntos de treinamento e teste. Isso é feito através do método 'train_test_split' da biblioteca Sklearn, conforme comandos abaixo. No caso, a proporção utilizada para treinar e testar os dados foi de (75% / 25%). O parâmetro 'stratify' permite que os dados sejam divididos de forma estratificada, considerando os rótulos da classe:

```
text_train, text_test, y_train, y_test =
 ↪  train_test_split(text, y, random_state=1,
 ↪  test_size=0.25, stratify=y)
```

3. **Vetorização dos dados:** As bibliotecas de aprendizagem de máquina esperam entrada na forma de arrays de floats, com dimensão fixa. Então, é necessário fazer a conversão dos textos em tais representações. Essa forma de representação é chamada *Bag of Words*, em que um texto é transformado em uma lista de tokens e, posteriormente, no array de floats, considerando o vocabulário dos textos. Os seguintes comandos fazem a conversão dos textos para arrays de floats, usando o método 'TfidfVectorizer'. No entanto, existem outros métodos de vetorização, a saber: *HashingVectorizer* e *CountVectorizer*. O método *TfidfVectorizer* considera os tokens como palavras (words), elimina as *stopwords* presentes no objeto 'sw' e usa no máximo 1500 *features* como dimensão do array.

```
sw = stopwords.words('portuguese')
vet = TfidfVectorizer(analyzer="word", stop_words=sw,
 ↪  max_features=1500)
vet.fit(text_train)
X_train = vet.transform(text_train)
X_test = vet.transform(text_test)
print("Treino/Teste: ", X_train.shape, X_test.shape)
>>> Treino/Teste: (11462, 1500) (3821, 1500)
```

No exemplo, os objetos 'X_train' e 'X_test' são do tipo *'scipy.sparse.csr.csr_matrix'*, chamada *matriz documento-termo*, que é uma matriz esparsa da forma (nrAmostras, nrFeatures). Destaca-se que o conjunto de *features* representa o vocabulário dos dados e pode ser visualizado através do comando `print(vet.vocabulary_)`.

4. **Treinar o classificador:** A biblioteca *Sklearn* possui diversos algoritmos tradicionais para a tarefa de classificação, incluindo as máquinas de vetores de suporte

(do inglês: *Support Vector Machines - SVM*), *Naïve Bayes* e as árvores de decisão (do inglês: *Decision Tree*). A fundamentação matemática para o entendimento dos algoritmos está fora do escopo deste trabalho. Aqui, vamos utilizar o método 'SVC', cuja implementação é baseada no algoritmo 'libsvm'. Os comandos a seguir são usados para instanciar e treinar esse algoritmo. Para conhecimento geral dos algoritmos disponíveis na biblioteca *Sklearn*, sugere-se a documentação oficial [9].

```
svc = SVC()
svc.fit(X_train, y_train)
```

5. **Avaliar o modelo:** O passo final do processo é avaliar o desempenho do algoritmo. Existem várias métricas que são usadas para avaliar as fases de treinamento e teste dos classificadores. Quando as classes são balanceadas, normalmente utiliza-se a Acurácia, que consiste na razão entre o número de predições verdadeiras e o número total de amostras. Para problemas de classificação binária e multiclasse, a matriz de confusão é utilizada para identificar a performance do modelo para cada classe. A matriz fornece quantas amostras de cada classe foram classificadas corretamente e quantas foram confundidas com outras classes. O código abaixo mostra a Acurácia do modelo para os dados de treino e teste, além de apresentar a matriz de confusão do modelo para os dados de teste.

```
print("Acc (Treino): %.2f"%svc.score(X_train, y_train))
print("Acc (Teste): %.2f"%svc.score(X_test, y_test))
print("Matriz de Confusão:")
print(confusion_matrix(y_test, svc.predict(X_test)))
>>> Acc (Treino): 0.98
>>> Acc (Teste): 0.96
>>> Matriz de Confusão:
>>> [[ 480  139]
     [  31 3171]]
```

Observa-se que, a Acurácia do modelo foi de 98% para os dados de treinamento e 96% para os dados de teste. A matriz de confusão segue a seguinte forma:

```
[[ TN, FP]
 [ FN, TP]]
```

sendo:

- **TN (*True Negative*):** A quantidade de rótulos da classe negativa que o modelo previu corretamente como negativo.
- **FP (*False Positive*):** A quantidade de rótulos que originalmente pertencem a classe negativa, mas o modelo previu como positivo.
- **FN (*False Negative*):** A quantidade de rótulos que originalmente pertencem a classe positiva, mas o modelo previu como negativo.

[9] https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

- **TP (*True Positive*):** A quantidade de rótulos da classe positiva que o modelo previu corretamente como positivo.

A biblioteca *Sklearn* possui também um relatório de classificação que fornece as métricas de **precisão, cobertura, medida-f e a quantidade de amostras de cada classe (*support*)**. Essas medidas são definidas da seguinte forma:

- **Precisão (P):** Número de amostras de uma determinada classe que são realmente daquela classe. $P = TP/(TP+FP)$.
- **Cobertura (Recall ou R):** Número de predições corretas de uma determinada classe. $R = TP/(TP+FN)$.
- **Medida-F (F1-Score ou F1):** É a média harmônica entre a precisão e a cobertura. $F1 = 2*(P*R)/(P+R)$.

O código abaixo é utilizado para imprimir o relatório da classificação do exemplo em questão e a Figura 1.3 mostra o resultado da precisão, cobertura e medida-f de cada classe considerando os dados do conjunto de teste. O relatório mostra que a medida-f da classe de ironia foi de 97% e para a classe não ironia foi inferior (apenas de 85%). Isto aconteceu devido a baixa cobertura desta segunda classe.

```
print("Relatorio de Classificacao")
print(classification_report(y_test,
 ↪  svc.predict(X_test)))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.94      0.78      0.85       619
           1       0.96      0.99      0.97      3202

    accuracy                           0.96      3821
   macro avg       0.95      0.88      0.91      3821
weighted avg       0.95      0.96      0.95      3821
```

Figura 1.3: Saída do relatório de classificação.

Para o caso de classe desbalanceadas, a curva ROC (do inglês: *Receiver Operating Characteristic*) ajuda a entender a performance do modelo. A Curva ROC funciona com a saída da função de predição, definindo diferentes valores (*threshold*) para descobrir diferentes taxas de falsos positivos (FP) e verdadeiros positivos (TP) de acordo com o *threshold*. O código a seguir é usado para plotar a curva ROC para o classificador SVC, considerando o conjunto de teste utilizado no exemplo. A Figura 1.4 mostra a representação gráfica resultante.

```
df = svc.decision_function(X_test)
fpr, tpr, thresholds = roc_curve(y_test, df,
 ↪  pos_label='1')
acc = svc.score(X_test, y_test)
auc = roc_auc_score(y_test,
 ↪  svc.decision_function(X_test))
```

```
with plt.style.context(('ggplot','seaborn')):
    plt.figure(figsize=(8,6))
    plt.scatter(fpr, tpr, c='blue')
    plt.plot(fpr, tpr, label="Acuracia:%.2f AUC:%.2f" %
     ↪  (acc, auc), linewidth=2, c='red')
    plt.xlabel("Taxa FP")
    plt.ylabel("Taxa TP")
    plt.title('Curva ROC')
    plt.legend(loc='best');
```

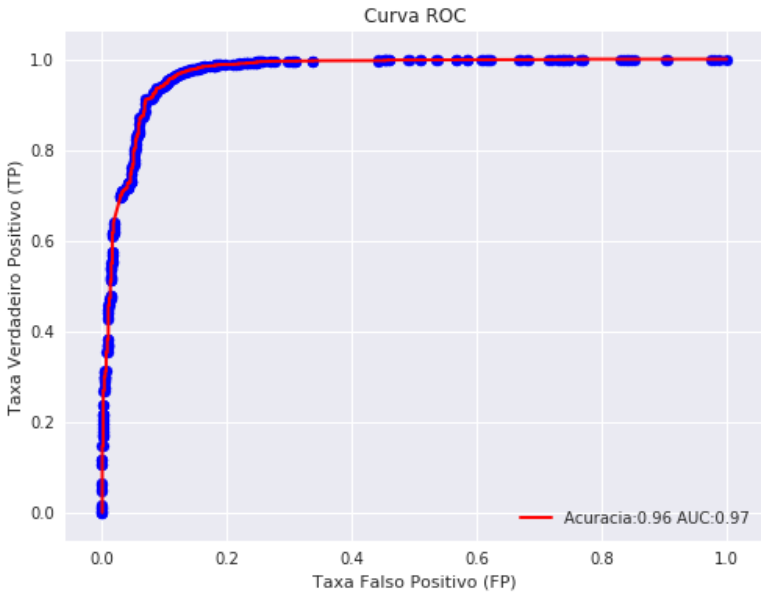


Figura 1.4: Curva ROC

Destaca-se que uma curva ROC é uma demonstração bidimensional da performance de um classificador. Para comparar classificadores é preciso reduzir a curva ROC a um valor escalar. Normalmente, usa-se o valor da área abaixo da curva (do inglês: *Area Under the Curve - AUC*). No exemplo, o valor AUC foi de 97%.

1.4. Word Embeddings

Embora as técnicas de vetorização de palavras descritas anteriormente (*bag of words*, *TF-IDF*) se mostrem simples e eficientes, elas possuem sérias limitações. Conforme [Mikolov et al. 2013a], quando aplicadas em grandes volumes de dados estes modelos apresentam matrizes de alta dimensão, que afetam o custo computacional ao serem processadas. Aliado a este fator, é comum encontrar a característica de esparsidade, que é a ausência do valor do grau de importância de uma determinada célula da matriz devido ao fato daquele termo não estar presente no documento associado. Em determinados documentos algumas palavras podem não estar presentes, e desta forma na matriz *termo x documento* o respectivo grau de importância é representado com o valor '0'. Por fim,

outra importante desvantagem é ausência de informações semânticas pois estes modelos ignoram o contexto e a ordem das palavras nos documentos [Manning et al. 2008].

Diferente dos modelos clássicos de Recuperação da Informação, as *Word Embeddings* representam cada uma das palavras como um vetor n-dimensional contínuo de números reais, de forma que esta representação é capaz de capturar relações de sintáticas e semânticas (e.g. similaridades). Essas relações são derivadas das posições destas palavras em um espaço vetorial, i.e., se estas palavras estão em regiões próximas, é possível computar a distância entre os vetores (*e.g.,* distância cosseno) e encontrar relações de similaridade entres elas[Mikolov et al. 2013b]. A Figura 1.5 apresenta uma abstração de um espaço vetorial onde é possível verificar a proximidade de palavras similares.

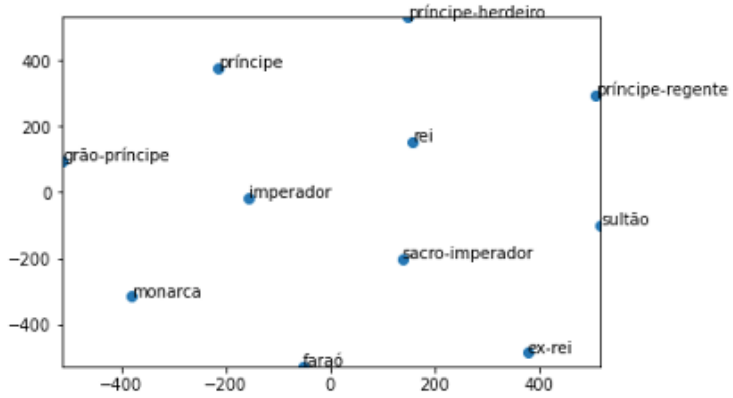


Figura 1.5: Representação abstrata de espaço vetorial: algoritmo Word2Vec

Como é possível verificar na figura, palavras como "rei", "sultão", "imperador"e "faraó"estão em regiões próximas no espaço vetorial, o que demonstra a característica de similaridade pois são termos sinônimos. Tais características são usadas para analisar diversas tarefas de PLN, como análise de sentimentos e detecção de discurso de ódio.

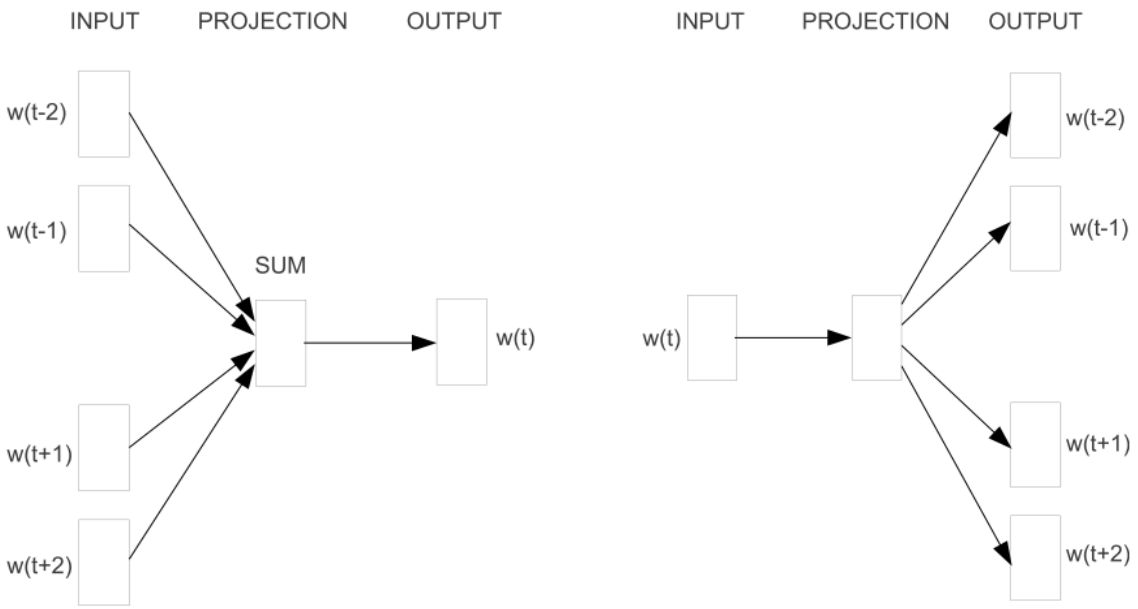
Alguns dos algoritmos mais utilizados na literatura de *Word Embeddings* são *Word2Vec* e *Paragraph Vector* (ou *Doc2Vec*). Ambos algoritmos são baseados em Redes Neurais Artificiais de duas camadas(camada de projeção e camada de saída), onde a partir de um *corpus* de entrada, ao gerarem vetores de palavras para cada um dos termos, buscam mapear estes vetores em um espaço vetorial onde é possível representar as palavras similares em vetores similares no espaço vetorial [Mikolov et al. 2013a].

1.4.1. Word2Vec

Conforme apresentado em [Mikolov et al. 2013a] [Mikolov et al. 2013b], o algoritmo *Word2vec* possui duas variações para o treinamento, são elas *Continous Bag of Words* (CBOW) e *Skip gram*. No algoritmo CBOW, a predição do termo central é dada a partir uma janela de palavras em um determinado contexto. A entrada da Rede Neural são as palavras (juntamente com seu vetor numérico) em uma dada janela, a camada oculta representa o número de dimensões na qual é desejado representar o termo central, que é o elemento da camada de saída. A Figura 1.6a apresenta uma ilustração deste algoritmo.

O funcionamento do algoritmo *Skip gram* é o oposto do CBOW. A partir de um

termo central, esta técnica busca predizer um conjunto de vetores de palavras dada uma janela de termos. Neste caso a camada de entrada é o termo central, a camada oculta também contém o número de dimensões na qual é desejado representar o termo central, e por fim a camada de saída apresenta o conjunto de termos em torno da palavra central. A Figura 1.6b mostra um exemplo deste algoritmo.



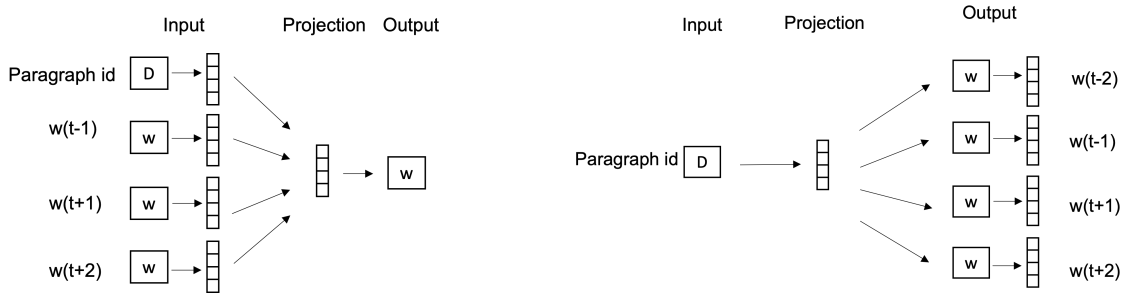
(a) Método CBOW (b) Método Skip-Gram

Figura 1.6: Algortimo Word2Vec: Variações para treinamento

1.4.2. Doc2vec

De acordo com [Le and Mikolov 2014], *Paragraph Vector* é um método não supervisionado que busca aprender um tamanho fixo de características a partir de diferentes formas textuais e.g., senteças, parágrafos ou documentos. Os autores apresentam duas abordagem deste algoritmo, *Distributed Memory* (DM) e *Distributed Bag of Words* (DBOW). O método DM, utiliza vetores de palavras junto a representação do vetor do parágrafo (sentença, ou documento) para a predição de um determinado termo dentro da sentença. Esta representação do vetor parágrafo atua como uma espécie de memória, sendo capaz de conceder informações contextuais à tarefa de predição. A Figura 1.7a apresenta o esquema do método DM. É possível perceber a semelhança ao método CBOW (Figura 1.6a), onde a diferença consiste no vetor *Paragraph Id* que é compartilhado pelos termos daquele contexto que está sendo utilizado no treinamento.

Ao contrário do método DM, a técnica DBOW utiliza somente as informações presentes no *Paragraph Id* (ver Figura 1.7b). Este método faz uma seleção aleatória de palavras na representação do vetor de parágrafos e usa estas informações para prever o termo central. Este método já se assemelha ao *Skip gram* (ver Figura 1.6b).



(a) Método DM (b) Método DBOW

Figura 1.7: Algortimo Paragraph Vector.

1.5. Algoritmos de Deep Learning

Apesar dos interessantes resultados alcançados pelos algoritmos tradicionais de aprendizado de máquina, eles sofrem com a dificuldade de cobrir todas as regularidades de uma língua através do desenvolvimento manual de *features* [Deng and Liu 2018]. Nesse contexto, os algoritmos de aprendizagem profunda (*deep learning)* são usados para preencher essa lacuna. *Deep learning* pode ser definido como um classe de técnicas de aprendizado de máquina que explora muitas camadas de processamento de informação não linear para extração e transformação automática de *features* para análise e classificação de padrões [Deng and Yu 2014]. Essa classe de algoritmos é baseada nas Redes Neurais e popularizou-se devido aos avanços tanto em *hardware* quanto em *software* e, principalmente, por causa dos resultados alcançados em diversas áreas. Aqui, serão apresentados dois tipos de algoritmos de aprendizagem profunda, Redes Convolucionais na subseção 1.5.1 e Redes Recorrentes na subseção 1.5.2.

1.5.1. Redes Convolucionais

As Redes Neurais Convolucionais (do inglês: *Convolutional Neural Networks - CNN*) foram introduzidas por [LeCun 1989] e são um tipo especializado de rede neural para processar dados que possuem uma topologia semelhante a uma grade, como as imagens.

Para a área de PLN essas redes foram inicialmente usadas para tarefas de anotação de papéis semânticos [Collobert et al. 2011], análise de sentimentos [Kim 2014] e classificação de questões [Kalchbrenner et al. 2014].

As principais operações de uma rede convolucional são a **convolução** e o ***pooling***. A **convolução** pode ser entendida como aplicação de uma função não linear sobre cada instância de uma janela deslizante de k-palavras sobre as sentenças. Essa função, chamada de filtro, transforma uma janela de k-palavras em um vetor de dimensão l. Cada dimensão corresponde a um filtro que captura propriedades importantes das palavras na janela [Goldberg 2017]. A Figura 1.8 mostra um exemplo de convolução sobre o texto "*Quixadá é bonita*". Como pode ser observado, o filtro é aplicado sobre as *words embeddings* para produzir um mapa de *features* que representa um k-grama específico.

O objetivo de camada de ***pooling*** é focar nas *features* mais importantes de uma sentença. Existem diversas operações de *pooling*: *max-pooling*, *average pooling*, *k-max*

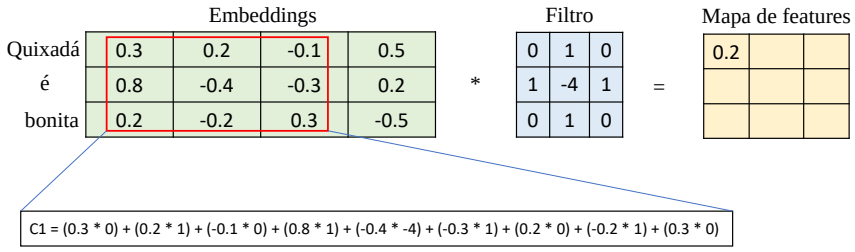


Figura 1.8: Exemplo de convolução sobre uma sentença.

pooling, entre outras [Goldberg 2017]. A mais comum é a *max-pooling* que pega o valor máximo em cada dimensão. Figura 1.9 exibe a operação de *max-pooling* para extrair os maiores valores da matriz, através de um filtro $2x2$ de passo 2.

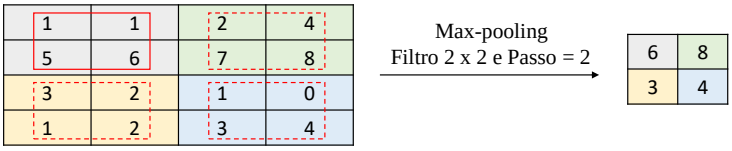


Figura 1.9: Exemplo da operação de max-pooling.

Pode-se perceber que a operação de *pooling* extraiu os maiores valores da matriz, baseado em um filtro $2x2$ de passo 2, resultando em uma matriz $2x2$. Diversos *frameworks* implementam essas operações, tais como: PyTorch[10], TensorFlow[11], Keras[12], entre outros. Esses *frameworks* são para a linguagem Python e eles abstraem as operações matemáticas por traz das redes neurais permitindo fazer uso das operações de convolução e *pooling* através de simples chamadas de métodos.

1.5.2. Redes Recorrentes

As Redes Recorrentes (do inglês: *Reccurent Neural Networks - RNNs*) foram introduzidas por [Rumelhart et al. 1986]. A principal característica dessa rede é trabalhar com dados sequenciais, como um texto. Assim como as redes convolucionais, as redes recorrentes são um tipo especializado de rede neural, no entanto elas são focadas em processar uma sequência de valores. Esse tipo de rede é bastante utilizado em tarefas de PLN, cuja entrada é uma sequência de texto e a saída também é uma sequência de texto, como a tradução automática.

De acordo com [Jurafsky and Martin 2009] uma rede recorrente é qualquer rede que contenha um ciclo em suas conexões de rede, ou seja, qualquer estrutura em que o valor de uma unidade seja direta ou indiretamente dependente de saídas anteriores como entrada. Na Figura 1.10, é exibido uma RNN que recebe uma entrada X no tempo t (X_t) e produz uma saída h_t. Essa saída e o próximo valor (X_{t+1}) são utilizados como entrada na próxima rede, produzindo a saída h_{t+1}. Esse processo continua até o final da sequência.

Tradicionalmente, uma rede recorrente percorre uma sequência da esquerda para a

[10] https://pytorch.org/
[11] https://www.tensorflow.org/
[12] https://keras.io/

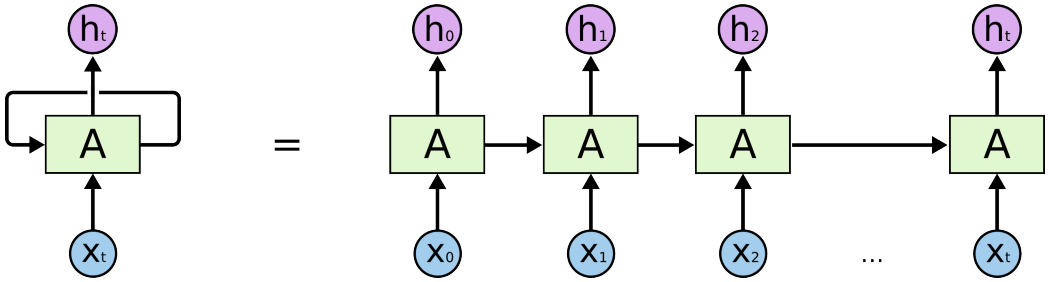


Figura 1.10: Estrutura de uma rede recorrente.

direita. No entanto, é possível percorrer de forma bidirecional: da esquerda para a direita e da direita para esquerda. Considerando a tarefa de etiquetagem gramatical, muitas vezes para se atribuir uma classe a uma palavra é necessário observar o contexto bidirecional da mesma. A Figura 1.11 exibe um exemplo desse tipo de rede.

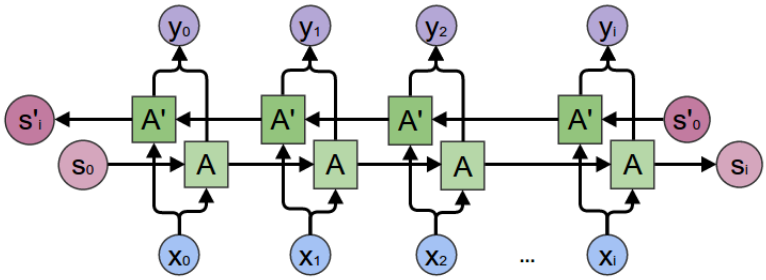


Figura 1.11: Estrutura de uma rede recorrente bidirecional.

A partir da figura acima, pode-se ver que o contexto está sendo analisado tanto de S_0 para S_i quanto de S'_0 para S'_i. Devido a capacidade de analisar o contexto de ambos os lados, as redes bidirecionais atingem resultados superiores as redes unidirecionais.

Além da bidirecionalidade, também é possível criar empilhar várias redes recorrentes, como apresentado na Figura 1.12. Nessa figura, a redes são unidirecionais, no entanto é possível criar redes recorrentes bidirecionais empilhadas. Nesse tipo de rede, a saída de cada unidade de processamento é a entrada para a rede recorrente no nível acima.

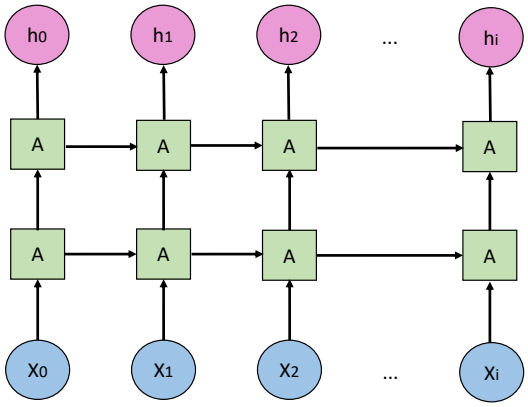


Figura 1.12: Exemplo de redes reccorentes empilhadas.

Existem dois tipos de redes recorrentes, *Long Short-Term Memory* (LSTM) e *Gated Reccurent Unit* (GRU). Essas redes foram criadas para resolver os problemas das RNNs convencionais que é o de "esquecer" informações ao longo do processamento em sentenças longas.

Assim como nas redes convolucionais, os *frameworks* PyTorch, TensorFlow e Keras também disponibilizam a implementação das redes recorrentes através de simples chamadas de métodos.

1.5.3. BERT

Bidirectional Encoder Representations from Transformer (BERT) [Devlin et al. 2019] é um modelo de língua baseado em *transformer*, que é um modelo de aprendizagem profunda que utiliza mecanismo de *self-attention* [Vaswani et al. 2017]. Modelo de língua é um modelo capaz de predizer a próxima palavra/sentença dado uma sequência prévia de palavras/sentenças. Por exemplo, a partir da sentença "*Tinha uma [MASK] no meio do caminho.*" um modelo de língua é treinado para predizer qual palavra deve substituir "[MASK]". Para esse exemplo, o BERT sugere que "[MASK]" deve ser substituída pela palavra "pedra". De modo simples, o mecanismo de atenção (*attention*) é um componente das redes profundas que é responsável em dar foco nos principais dados de entrada.

De forma geral, BERT é um modelo que utiliza vetores de sub-palavras mais *transformers*. Os vetores de sub-palavras possuem um contexto para cada vetor que comportam variações morfológicas diferentes das *embeddings* tradicionais. Os *transformers* não utilizam direcionalidade, ao invés disso eles utilizam o contexto inteiro da sentença de uma só vez. A arquitetura do BERT pode ser simplificada em duas partes, como exibido na Figura 1.13. No lado esquerdo, está o tokenizador que quebra as palavras em sub-palavras e o *encoder* responsável por produzir diferentes representações do texto de entrada. Ainda no lado esquerdo, o BERT é treinado em dados não rotulados, visando predizer a tag "[MASK]. Nessa tarefa, o BERT recebe um texto e, de forma aleatória, substitui algumas palavras por "[MASK]" a fim de posteriormente predizê-las. A partir desse treinamento em dados não rotulados, é possível fazer o treinamento em dados rotulados, ou seja, o *decoder* (lado direito). Esse treinamento é chamado de afinação (fine-tuning), pois o BERT vai ser ajustado para alguma tarefa específica, por exemplo, análise de sentimentos, tradução automática, sumarização automática, e assim por diante.

Essa capacidade de afinação e transferência de aprendizagem, uma vez que a partir do treinamento em dados não rotulados, O BERT é capaz é ser treinados em diversas tarefas de PLN e atingir resultados estado-da-arte é uma das grandes características desse modelo. Para mais detalhes sobre esse modelo de língua, sugere-se o seguinte guia ilustrado[13]. Os *frameworks* TensorFlow, PyTorch e keras possuem implementações do modelo para fácil uso, sendo a biblioteca HuggingFace[14] uma das mais famosas.

1.6. Conclusão

Conhecer várias técnicas de PLN é importante para desenvolver modelos de aprendizagem mais robustos e competitivos. Dessa forma, este capítulo apresentou uma visão geral

[13] `https://jalammar.github.io/illustrated-bert/`
[14] `https://github.com/huggingface/transformers`

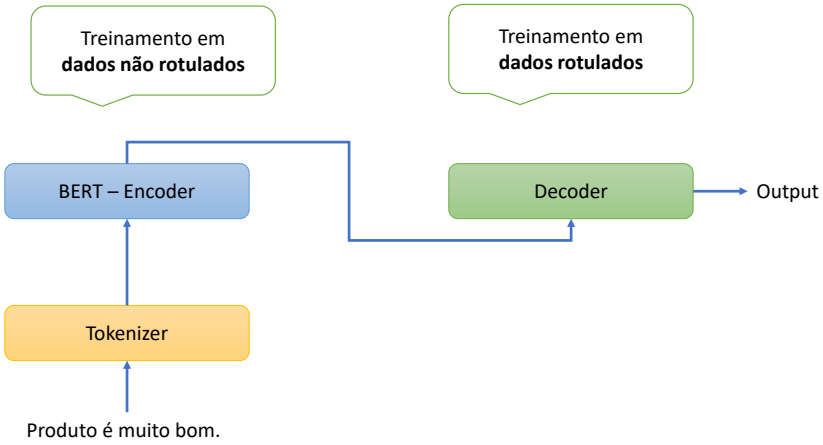


Figura 1.13: Arquitetura simplificada do modelo BERT.

sobre várias técnicas de Processamento de Língua Natural focadas na tarefa de classificação, iniciando com abordagens baseadas em *features*, *n*-gramas, léxicos passando por estratégias baseadas em *word embeddings* e finalizando em métodos baseados em aprendizado profundo e modelos língua.

A área de PLN tem focado principalmente em estratégias de aprendizado profundo e modelos de língua devido tanto a evolução do *software* quanto *hardware* e da disponibilidade de grandes *corpora*. A junção desses fatores proporcionou alcançar resultados impressionantes em diversas tarefas de PLN. Por fim, espera-se que este capítulo sirva como base e ajude no desenvolvimento de métodos cada vez mais robustos.

Referências

[Anchiêta et al. 2013] Anchiêta, R. T., de Sousa, R. F., and Moura, R. S. (2013). Using NLP techniques for identifying GUI prototypes and UML diagrams from use cases. In *Proceedings of the 25th International Conference on Software Engineering and Knowledge Engineering*, Boston, MA, USA.

[Anchiêta et al. 2015] Anchiêta, R. T., Ricarte Neto, F. A., de Sousa, R. F., and Moura, R. S. (2015). Using stylometric features for sentiment classification. In *Proceedings of the 16th International Conference on Intelligent Text Processing and Computational Processing*, Cairo, Egypt.

[Bird 2006] Bird, S. (2006). NLTK: The Natural Language Toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 69–72, Sydney, Australia. Association for Computational Linguistics.

[Bird et al. 2009] Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly.

[Collobert et al. 2011] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12.

[Deng and Liu 2018] Deng, L. and Liu, Y. (2018). *Deep Learning in Natural Language Processing*. Springer Singapore.

[Deng and Yu 2014] Deng, L. and Yu, D. (2014). Deep learning: methods and applications. *Foundations and trends in signal processing*, 7(3–4):197–387.

[Devlin et al. 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Minneapolis, Minnesota.

[Faceli et al. 2021] Faceli, K., Lorena, A. C., Gama, J., and Carvalho, A. C. P. L. F. D. (2021). *Inteligência artificial: uma abordagem de aprendizado de máquina*. LTC.

[Ferreira et al. 2019] Ferreira, J., Oliveira, H. G., and Rodrigues, R. (2019). Improving NLTK for processing Portuguese. In *8th Symposium on Languages, Applications and Technologies*, Coimbra, Portugal.

[Fonseca and Rosa 2013] Fonseca, E. R. and Rosa, J. L. G. (2013). Mac-morpho revisited: Towards robust part-of-speech tagging. In *Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology*, Fortaleza, Brazil. SBC.

[Fortuna and Nunes 2018] Fortuna, P. and Nunes, S. (2018). A survey on automatic detection of hate speech in text. *ACM Computing Surveys (CSUR)*, 51(4):1–30.

[Goldberg 2017] Goldberg, Y. (2017). *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers.

[Jurafsky and Martin 2009] Jurafsky, D. and Martin, J. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall series in artificial intelligence. Pearson/Prentice Hall.

[Kalchbrenner et al. 2014] Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Baltimore, Maryland.

[Kaliyar et al. 2021] Kaliyar, R. K., Goswami, A., and Narang, P. (2021). Fakebert: Fake news detection in social media with a bert-based deep learning approach. *Multimedia Tools and Applications*, 80(8):11765–11788.

[Kim 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar. Association for Computational Linguistics.

[Kinoshita et al. 2007] Kinoshita, J., Salvador, L., Menezes, C., and Silva, W. (2007). Cogroo - an openoffice grammar checker. In *Proc. of the 17th International Conference on Intelligent Systems Design and Applications*, Rio de Janeiro, Brazil. IEEE.

[Le and Mikolov 2014] Le, Q. and Mikolov, T. (2014). Distributed Representations of Sentences and Documents. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1188–1196, Bejing, China. PMLR.

[LeCun 1989] LeCun, Y. (1989). Generalization and network design strategies. Technical report, University of Toronto.

[Leite et al. 2020] Leite, J. A., Silva, D., Bontcheva, K., and Scarton, C. (2020). Toxic language detection in social media for Brazilian Portuguese: New dataset and multilingual analysis. In *Proc. of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, Suzhou, China.

[Liu 2012] Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis Lectures on Human Language Technologies*, 5(1):1–167.

[Liu 2015] Liu, B. (2015). *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge University Press.

[Magalhães Jr et al. 2019] Magalhães Jr, G. V., Vieira, J. P. A., Santos, R. L. S., Barbosa, J. L. N., Santos Neto, P., and Moura, R. (2019). A study of the influence of textual features in learning medical prior authorization. In *Proc. of the 32nd International Symposium on Computer-Based Medical Systems*, Córdona, Spain. IEEE.

[Manning et al. 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[Meneses Silva et al. 2021] Meneses Silva, C. V., Silva Fontes, R., and Colaço Júnior, M. (2021). Intelligent fake news detection: a systematic mapping. *Journal of applied security research*, 16(2):168–189.

[Mikolov et al. 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. In *Proc. of the 1st International Conference on Learning Representations Workshop papers*, Scottsdale, AZ, USA.

[Mikolov et al. 2013b] Mikolov, T., Yih, W.-t., and Zweig, G. (2013b). Linguistic regularities in continuous space word representations. In *Proc. of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Atlanta, Georgia.

[Pedregosa et al. 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12.

[Poria et al. 2016] Poria, S., Cambria, E., and Gelbukh, A. (2016). Aspect extraction for opinion mining with a deep convolutional neural network. *Knowledge-Based Systems*, 108(C):42–49.

[Qi et al. 2020] Qi, P., Zhang, Y., Zhang, Y., Bolton, J., and Manning, C. D. (2020). Stanza: A python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Online.

[Rumelhart et al. 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

[Santos et al. 2021] Santos, R. L. S., Sa, C. A., Sousa, R. F., Anchiêta, R. T., Rabelo, R. A. L., and Moura, R. S. (2021). Estimating importance from web reviews through textual description and metrics extraction. In *Natural Language Processing for Global and Local Business*. IGI Global.

[Sardinha 2000] Sardinha, T. B. (2000). Linguística de corpus: histórico e problemática. *DELTA: Documentação de Estudos em Lingüística Teórica e Aplicada*, 16:323–367.

[Sarkar 2019] Sarkar, D. (2019). *Text Analytics with Python. A Practitioner's Guide to Natural Language Processing*. APress.

[Soares and Moura 2015] Soares, H. A. and Moura, R. S. (2015). A methodology to guide writing software requirements specification document. In *Proceedings of the 2015 Latin American Computing Conference*, pages 1–11, Arequipa, Peru. IEEE.

[Sousa 2015] Sousa, R. F. (2015). Abordagem TOP(X) para inferir os comentários mais importantes sobre produtos e serviços. Master's thesis, Universidade Federal do Piauí.

[Vaswani et al. 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, Long Beach, USA.