Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Detecção de sarcasmo em redes sociais utilizando DeBERTa

Lucas Paiolla Forastiere

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Ricardo Marcondes Marcacini

São Paulo
2017

Resumo

Lucas Paiolla Forastiere. **Detecção de sarcasmo em redes sociais utilizando DeBERTa**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2017.

Elemento obrigatório, constituído de uma sequência de frases concisas e objetivas, em forma de texto. Deve apresentar os objetivos, métodos empregados, resultados e conclusões. O resumo deve ser redigido em parágrafo único, conter no máximo 500 palavras e ser seguido dos termos representativos do conteúdo do trabalho (palavras-chave). Deve ser precedido da referência do documento. Texto texto. Texto texto.

Palavras-chave: Palavra-chave1. Palavra-chave2. Palavra-chave3.

Abstract

Lucas Paiolla Forastiere. **Sarcasm detection in social media using decoding-enhanced BERT with disentangled attention**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2017.

Elemento obrigatório, elaborado com as mesmas características do resumo em língua portuguesa. De acordo com o Regimento da Pós-Graduação da USP (Artigo 99), deve ser redigido em inglês para fins de divulgação. É uma boa ideia usar o sítio www.grammarly.com na preparação de textos em inglês. Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text. Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text.

Keywords: Keyword1. Keyword2. Keyword3.

Lista de Abreviaturas

CFT	Transformada contínua de Fourier (*Continuous Fourier Transform*)
DFT	Transformada discreta de Fourier (*Discrete Fourier Transform*)
EIIP	Potencial de interação elétron-íon (*Electron-Ion Interaction Potentials*)
STFT	Transformada de Fourier de tempo reduzido (*Short-Time Fourier Transform*)
ABNT	Associação Brasileira de Normas Técnicas
URL	Localizador Uniforme de Recursos (*Uniform Resource Locator*)
IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo

Lista de Símbolos

ω	Frequência angular
ψ	Função de análise *wavelet*
Ψ	Transformada de Fourier de ψ

Lista de Figuras

Lista de Tabelas

Lista de Programas

Sumário

Capítulo 1

Introduction

Sarcasm detection is an important aspect of many natural language processing (NLP) systems, with many implications in natural language understanding, dialog systems, and data mining. However, sarcasm detection is difficult because it is infrequent in many conversations and, many times, it is difficult even for humans to discern.

Many studies have been made in the area and many datasets have been proposed with either *balanced* or *unbalanced* data. Also many of these datasets use humans to annotate sarcastic statements.

In this paper, we use the Self-Annotated Reddit Corpus (SARC), which is a large corpus for sarcasm detection created using Reddit posts to get labels automatically, to train, evaluate, and compare many NLP models.

Capítulo 2

Fundamentos e Trabalhos Relacionados

2.1 Detecção de Sarcasmo

De acordo com o dicionário Dicio, **sarcasmo** é uma zombaria que busca ofender, enquanto **ironia** é a ação de dizer o oposto do que se deseja expressar. Ainda segundo ele, a diferença entre esses dois termos se dá no fato de que sarcasmo é um dito ácido que pode ou não ser expresso por meio de uma ironia e essa, por sua vez, pode ou não ser utilizada para ofender. Dicio, 2022b; Dicio, 2022a

(R. Giora, On Irony and Negation)

(H. Paul Grice. 1975. Logic and Conversation)

(Irony and Sarcasm: Corpus Generation and Analysis Using Crowdsourcin)

O termo **detecção de sarcasmo** refere-se à determinação de se há ou não sarcasmo em uma porção de texto verbal. E o termo **detecção automática de sarcasmo** refere-se a resolução (ou tentativa de) resolver o problema mencionado utilizando métodos computacionais que automatizam essa decisão. Computacionalmente, podemos definir esse problema como uma **classificação binária** de texto, termo explicado mais a frente.

Entretanto, na literatura é bastante comum também se definir detecção de sarcasmo como a determinação de se há ou não sarcasmo ou ironia verbal em uma porção de texto verbal. Portanto, em geral, ao se falar de sarcasmo, a literatura engloba tanto sarcasmo quanto ironia como se fossem a mesma coisa. Este texto também não fará discriminação entre sarcasmo ou ironia.

Em geral, esse problema é difícil, pois, por vezes, nem humanos conseguem perceber essa figura de linguagem (e.g. "*Oba, hoje está tão ensolarado, que vontade de ir para a escola.*"). Além disso, o contexto tende a importar muito. Muitas características externas ao texto podem servir para descriminar se uma pessoa está ou não sendo sarcástica. Entre alguns exemplos estão intonação, locutor, interlocutor, conhecimento prévio sobre a fala, tempo e espaço em que se fala e elementos não verbais. Wallace *et al.*, 2014

2.2 Classificação binárias

O problema de detecção de sarcasmo pode ser tratado como uma classificação binária. Esse tipo de problema é muito estudado no campo do aprendizado de máquina e envolve classificar os elementos de um conjunto em dois grupos chamados de classes. Por classificar entende-se a determinação de um elemento do conjunto entre pertencente à primeira ou segunda classe.

No caso da detecção de sarcasmo, os elementos são os pedaços de texto e as duas classes são *ser sarcástico* e *não ser sarcástico*. Em termos matemáticos, tem-se um conjunto de exemplos denotado pela matriz X de dimensões $n \times m$, onde n é o número de exemplos e m é o número de características que se usa para descrever cada um desses exemplos. Cada linha de X é denotada por $x^{(i)}$ e chamada de *exemplo i* ou *instância i*.

A linha $x^{(i)}$ é um vetor de m posições em que cada posição é uma característica que descreve a entrada. Por exemplo, na detecção de sarcasmo, o primeiro valor pode ser a quantidade de palavras, o segundo valor pode ser a quantidade de caracteres, a terceira pode ser a soma de positividade do texto (definida por algum critério específico) e assim por diante. É importante que a mesma posição em diferentes instâncias tenha o mesmo significado. Portanto, caso se defina que a primeira posição represente o número de palavras, todas as instâncias devem seguir essa regra.

Cada instância i pertence ou à primeira classe ou à segunda, modela-se isso por um valor $y^{(i)}$ chamado de i-ésimo rótulo (do inglês, *label*). E escreve-se $y^{(i)} \in \{ 0, 1 \}$, onde cada valor representa uma das classes. No caso de sarcasmo, pode-se representar por *não sarcasmo* o valor 0 e *sarcasmo*, o valor 1. Com esses valores $y^{(i)}$ constrói-se um vetor y onde $y_i = y^{(i)}$.

Ao final, o problema é descobrir uma função f que mapeie bem X em y. Note, entretanto, que n, o número de exemplos, é possivelmente infinito, pois sempre se pode achar novos exemplos e testar se f os mapeia corretamente. Portanto, na prática, o valor de n é limitado às instâncias em um determinado *conjunto de dados* coletado de alguma forma (manual, semi-automática ou automática). Ou seja, tenta-se achar essa função f conhecendo parcialmente o conjunto das instâncias. Além disso, após encontrar a função f nesse conjunto limitado de dados, chamado de *dados de treinamento*, deseja-se utiliza-la para discriminar novos textos similares aos originais, mas para os quais não se sabe de antemão se são ou não sarcásticos. Esses novos dados são, em muitos casos, chamados de *dados de teste*.

Vale a pena, entretanto, mencionar que os dados de treinamento geralmente são divididos em duas ou três partições que são chamadas de *dados de treinamento*, *dados de validação* e *dados de teste*. Essa nomenclatura pode ser um pouco confusa, pois os termos acabam se repetindo. Portanto, daqui em diante se utilizará *dados rotulados* para os dados que se sabe a classe de antemão (ou seja, sabe-se se aquele texto é ou não sarcástico) e *dados não rotulados* para os dados novos em que se deseja aplicar f e não se sabe a classe de antemão.

2.2.1 Métricas de avaliação

Para saber se a função f mapeia bem X em y, é preciso definir uma **métrica de avaliação**, que é uma função matemática bem definida que indica quanto de erro se comete em determinado conjunto finito de exemplos X.

Dado um conjunto de entrada X com n instâncias, seja y o vetor que representa se cada instância é ou não sarcástica. Seja então f a função sob avaliação (ou seja, a função de modelagem que recebe os textos e determina se são ou não sarcásticos). Seja $\hat{y} = f(X)$ os resultados gerados por essa função.

Note que para cada exemplo i, podemos ter duas opções:

$$\begin{array}{ll} \hat{y}_i = y_i & \textit{acerto} \\ \hat{y}_i \neq y_i & \textit{erro} \end{array}$$

Além disso, note que o par $(y_i,\ \hat{y}_i)$ pode ter quatro valores:

$(y_i,\ \hat{y}_i)$	caso	nome em inglês	notação
$(1,\ 1)$	verdadeiro positivo	*true positive*	tp
$(0,\ 1)$	falso positivo	*false positive*	fp
$(0,\ 0)$	verdadeiro negativo	*true negative*	tn
$(1,\ 0)$	falso negativo	*false negative*	fn

Tabela 2.1: *Os nomes dados para os diferentes tipos de erros e acertos.*

Nesses quatro casos, dois correspondem a quando o modelo acerta, os verdadeiros positivos e verdadeiros negativos, e dois correspondem a erros, os falsos positivos e falsos negativos. Os falsos positivos são, em alguns contextos, chamados de erros tipo 1, e são cometidos quando uma instância é erroneamente classificada como positiva. Já os falsos negativos são também chamados de erros tipo 2, e são cometidos quando uma instância é erroneamente classificada como negativa.

Com esses valores em mente, pode-se definir algumas métricas que ajudam a perceber quão boa a função f é. Dessa forma, pode-se comparar duas funções f.

Acurácia Definida simplesmente como a quantidade total de acertos divido pela quantidade total de instâncias. Seja n' a quantidade de acertos, então:

$$\texttt{Acurácia} = \frac{n'}{n}$$

É comum também definir a acurácia em termos dos valores acima listados:

$$\texttt{Acurácia} = \frac{tp + tn}{tp + tn + fp + fn}$$

Note que $tp + tn$ representa justamente quando $\hat{y}_i = y_i$ e $fp + fn$, representa quando $\hat{y}_i \neq y_i$.

Precisão Definida como:

$$\texttt{Precisão} = \frac{tp}{tp + fp}$$

é a taxa de verdadeiros positivos em relação todas as instâncias preditas como positivas. Portanto, é a fração entre todas as instâncias previstas como positivas e as instâncias que realmente eram positivas. Ela pune o modelo quando ele comete erros do tipo 1, ou seja, quando classifica valores demais como positivos.

Revocação Muitas vezes chamada por seu nome em inglês, *recall*, é definida como:

$$\texttt{Revocação} = \frac{tp}{tp + fn}$$

é a taxa de verdadeiros positivos em relação a todos os positivos. Portanto, mede a fração de quantos positivos foram acertados em relação a todos os que existiam no conjunto observado. Ela pune o modelo quando ele comete erros do tipo 2, ou seja, quando ele deixa de classificar um valor como positivo.

A precisão e a revocação guardam uma relação chave entre si e geralmente se quer ter um bom balanço entre as duas. Note que o modelo deve achar um ponto ótimo entre classificar valores como 1 demais (baixa precisão) e de menos (baixa revocação). Observe essa relação a partir do seguinte exemplo.

Comece com a função

$$f\left(x^{(i)}\right) = 1 \quad \forall i \tag{2.1}$$

Como ela sempre considera a entrada como positiva, então os únicos casos existentes serão de verdadeiros e falsos positivos. Logo:

$$\texttt{Precisão} = \frac{tp}{tp + fp} = \frac{tp}{n}$$

e

$$\texttt{Revocação} = \frac{tp}{tp + fn} = \frac{tp}{tp + 0} = 1$$

Portanto, a revocação terá o maior valor possível, enquanto a precisão será exatamente a fração de valores positivos que o conjunto observado possui. Como, semanticamente, a revocação é penalizada sempre que um elemento é positivo, mas foi previsto como negativo, então faz sentido que se tenha uma revocação de 100%, já que não se comete esse tipo de erro.

Agora, tome a função

$$f\left(x^{(i)}\right) = 0 \quad \forall i \tag{2.2}$$

Como ela sempre considera a entrada como negativa, então os únicos casos existentes

serão de verdadeiros e falsos negativos. Logo:

$$\texttt{Precisão} = \frac{tp}{tp + fp} = \frac{0}{0 + fp} = 0$$

e

$$\texttt{Revocação} = \frac{tp}{tp + fn} = \frac{0}{0 + fn} = 0$$

Portanto, como a função nunca tenta marcar uma instância como positiva, ela nunca acerta os verdadeiros positivos e comete 100% de erro.

Métrica F_1 Por esses e outros motivos, é interessante agregar a precisão e revocação em um único valor, que permite fácil comparação entre dois modelos. É possível utilizar uma média simples desses dois valores, mas é muito mais comum a utilização da métrica F_1 (ou, do inglês, F_1 *score*).

Ela é definida como:

$$F_1 = 2 \cdot \frac{\texttt{precisão} \cdot \texttt{revocação}}{\texttt{precisão} + \texttt{revocação}}$$

Obter um valor F_1 alto significa que ambas precisão e revocação são altas.

2.3 Trabalhos Relacionados

Esta seção aborda os trabalhos e as diferentes abordagens realizados na área. Pode-se caracterizar os trabalhos por três principais tipos de abordagens: as baseadas em regras, baseadas em métodos de aprendizado e baseadas em contexto. Além disso, fala-se um pouco sobre as principais características extraídas dos textos que são utilizadas para melhorar a eficácia das soluções.

2.3.1 Abordagens Baseadas em Regras

Abordagens baseadas em regras são aquelas que usam regras fixas para determinar se uma sequência de palavras contém ou não ironia. Para criar essas regras, várias características do texto podem ser utilizadas, como as classes sintáticas das palavras, se são palavras de cunho positivo ou negativo, se há a presença ou não de certas palavras em uma ordem, entre qualquer outro tipo de regra proposta.

Essa abordagem é computacional, porque as regras formam um algoritmo que pode ser implementado por uma linguagem de computação. Dessa forma, qualquer sequência de palavras pode ser dada como entrada para o algoritmo e ele retornará se ele acredita que essa sequência contém ou não sarcasmo.

A principal vantagem desse tipo de abordagem é que ela, além de detectar o sarcasmo, ajuda a estudá-lo. Ao criar uma regra do tipo *se o texto possui essas características, então ele é sarcástico* se cria uma explicação para as principais características presentes em um texto sarcástico.

Entretanto, as metodologias utilizadas são bastante específicas para cada tipo de texto e cada conjunto de dados. As regras criadas para um determinado conjunto, por exemplo, da rede social Twitter podem não funcionar em outras redes sociais como o Facebook, o Instagram ou o WhatsApp, pois cada uma dessas redes sociais possui um contexto e formato de conversação muito diferente. Portanto, são modelos que nos permitem explicar o processo de decisão, mas não permitem fácil generalização.

Veale e Hao, 2010 investigam em seu artigo sequências da forma "*as * as a **" ("tão * quanto um(a) *"), consideradas como analogias entre o que os autores chamam de *base* (*ground*) e *meio* (*vehicle*). Eles utilizam a API do Google para coletar 45021 instâncias do padrão "*about as * as **" e filtram os resultados manualmente para chegar em 20299 instâncias que de fato são consideradas analogias. Eles então anotam manualmente os rótulos para essas instâncias e encontraram que 15502 casos (76%) são irônicos e apenas 4797 (24%) são não irônicos.

Então, dado uma sequência "*as * as a **", os autores utilizam mecanismos de busca na rede como a API do Google para distinguir entre três casos: os casos em que essa sequência nunca foi usada como "*about*"; aqueles que já foram, mas não frequentemente; e aqueles que são frequentemente usados com essa marcação. Segundo os próprios autores, essas três categorias fornecem, respectivamente, evidencia fraca contra ironia, evidencia fraca a favor da ironia e evidência forte para ironia. Ou seja, o caso em que se tem mais certeza de que é uma ironia é o caso em que a sequência é frequentemente utilizada junto com a palavra "*about*".

Assim sendo, Veale e Hao, 2010 criam uma sequência de nove passos para classificar uma frase desse tipo entre as classes *ironia* e *não ironia*. Esses passos são bastante claros e precisos, podendo ser, portanto, implementados por um programa de computador e caracterizando, por conseguinte, uma detecção automática de sarcasmo.

Nessas regras, os autores utilizam o fato de que analogias mais frequentemente utilizadas são menos prováveis de serem irônicas, pois a ironia utiliza bastante a criatividade do locutor para fazer analogias não usuais.

Devido à natureza das regras propostas pelos autores, eles conseguem medir a precisão e revocação obtidas por cada uma das regras. No geral, o modelo atinge uma acurácia de 88%, o que é um valor bastante alto para esse tipo de problema. Entretanto, note que os autores se limitaram a um escopo muito fechado (das frases do tipo "*as * as a **").

Maynard e Greenwood, 2014 exploram o uso de regras baseadas em *hashtags* presentes em postagens da rede social Twitter e sua aplicação para detecção de sarcasmo em um contexto mais geral de análise de sentimento.

Em seu artigo, eles utilizam um algoritmo de tokenização de *hashtags* para transformá-las em palavras com as quais eles conseguem extrair informação. No caso, eles utilizam as palavras contidas nas *hashtags* para avaliar se o sentimento é positivo ou negativo e inverter o sentido original da frase caso detectem sarcasmo nas *hashtags*. Por exemplo, a *hashtag #notreally* é tokenizada em *not* e *really* e identificada como uma *tag* sarcástica de acordo com regras definidas pelos autores.

Então, eles aplicam cinco regras que podem determinar o sentimento de uma postagem

como *negativo* ou *positivo*, ou então trocar o sentimento pré-determinado da postagem. Os autores, assim, fazem uso da detecção de sarcasmo utilizando *hashtags* para resolver um outro problema mais geral que é definir o sentimento predominante de um texto.

Como ponto negativo de sua técnica, está o fato de que nem sempre o tokenizador de *hashtags* funciona de forma adequada. Por exemplo, *#greatstart* pode ser dividida de duas formas: *greats* e *tart* ou *great* e *start*. Um ser humano provavelmente saberia que a segunda opção é a mais provável, mas o algoritmo não sabe fazer essa distinção e acaba ficando com o primeiro conjunto e palavras. Apesar disso, esse sistema de tokenização possui um $F1$ de $97,25\%$ e os autores obtiveram 91% de precisão e revocação na tarefa no conjunto de dados utilizado por eles.

Bharti *et al.*, 2015 apresentam duas abordagens baseadas em regras. A primeira é através da análise morfossintática (do inglês, *Part-of-Speech Tagging*) e criação de árvores sintáticas (do inglês, *parse-trees*) que identificam a positividade do sentimento e situação predominantes em *tweets*. A segunda é através da análise sintática de *tweets* que começam com interjeições.

Em sua primeira abordagem, os autores utilizam dicionários e algoritmos pré-criados para encontrar as classes morfológicas e sintáticas das palavras. Então, baseando-se nisso, utilizam um algoritmo criado por eles para encontrar um valor de positividade para o sentimento do texto e para a situação retratada e, caso seus valores tenham sinais contrários (sentimento positivo e situação negativa ou vice-versa), eles consideram o texto como sarcástico. Por exemplo, em "*I hate Australia in cricket, because they always win*" ("eu odeio a Australia no críquete, porque eles sempre ganham"), as palavras "*I hate*" apresentam um sentimento negativo, enquanto as palavras "*they always win*" retratam uma situação positiva, e, portanto, essa frase seria classificada como sarcástica em sua primeira abordagem.

Em sua segunda abordagem, os autores utilizam novamente algoritmos de *POS Tagging* pré-criados para achar as classes morfológicas e sintáticas de palavras em *tweets* que começam com interjeições, como *wow*, *oh*, *wow*, *aha*, *yay*, *yeah*, *nah*, etc. Em seu algoritmo, eles classificam como sarcásticos os textos que começam por interjeições e possuem um adjetivo ou advérbio imediatamente após a interjeição, ou então possuem, em alguma posição após a interjeição, ou um advérbio seguido de adjetivo ou um adjetivo seguido de um substantivo ou um advérbio seguido de um verbo. Essa abordagem é bastante interessante, pois ela é bastante simples e, ainda assim, consegue um ótimo resultado de 0.90 F_1 no subconjunto de *tweets* marcado com a hashtag *sarcasm*.

2.3.2 Conjuntos de Características

Antes de prosseguir com as abordagens, essa seção mostra um pouco dos principais conjuntos de características usados. Abordagens baseadas em regras utilizam determinadas características como principalmente as classes morfossintáticas das palavras. Entretanto, as abordagens listadas a seguir fazem uso de conjuntos muito maiores de características retiradas do texto e vários trabalhos foram realizados para expandir a quantidade de características possíveis de se utilizar.

Como dito anteriormente, ao criar a matriz de exemplos de entrada X, cada coluna

representa uma característica do texto. Dessa forma, o texto fica agregado em um conjunto de valores numéricos que são acessíveis para a máquina utilizar e comparar. Ao invés de se comparar dois texto distintos com números de caracteres e palavras diferentes, utiliza-se um algoritmo para extrair características (do inglês, *features*) comuns entre os textos e compará-los por meio dessas características.

Um exemplo é o método chamado *bag-of-words* (em tradução literal, "sacola-de-palavras") que cria um vetor com, por exemplo, 512 posições de valores naturais onde cada posição do vetor representa quantas vezes uma determinada palavra apareceu no texto. Cada posição do vetor é associada a uma palavra e, muitas vezes, utiliza-se uma outra posição especial para denotar qualquer palavra que não está no dicionário de palavras selecionadas previamente. Por exemplo, a primeira posição pode representar quantas vezes a palavra "*a*" aparece no texto, a segunda posição pode representar quantas vezes a palavra "*the*" aparece, e assim por diante. Se a palavra "Lucas" for lida, então será adicionado um à última posição do vetor, que é utilizado para contar qualquer palavra que não é uma das outras 511.

(imagem ilustrando BOW)

Reyes, Rosso e Buscaldi, 2012 exploram quatro grupos de características dentro de um contexto mais geral de detecção de humor e ironia. Esses grupos são: ambiguidade, através dos níveis estrutural, morfossintático e semântico; polaridade, através de palavras que possuam semântica positiva ou negativa; imprevisibilidade, através de desbalanceamentos contextuais entre o significado literal das palavras; e os cenários emocionais, através das emoções passadas por cada palavras.

Liebrecht *et al.*, 2013 utilizam uni-, bi- e trigramas como características do texto. Um n-grama é uma sequência de n palavras do texto. Portanto, nesse tipo de característica, o unigrama seria o *bag-of-words* clássico citado acima, com as contagens de cada palavra, já o bigrama se trata das contagens de duas palavras seguidas. Por exemplo, sempre que as palavras "*so nice*" aparecerem em sequência, adiciona-se um à coluna que se refere a essa sequência.

Reyes, Rosso e Veale, 2013 utilizam uma abordagem parecida com Reyes, Rosso e Buscaldi, 2012, com quatro grupos de características: assinaturas, imprevisibilidade, estilo e cenários emocionais. Cada um dos grupos foca em várias características. Dentro das assinaturas, por exemplo, há o foco em pontuações específicas, emojis, citações e palavras em maiúsculas; há o foco em marcas implícitas que geram oposição entre partes do texto; e há o foco na compressão temporal, que identifica oposição temporal entre elementos relacionados.

A grande diferença desse trabalho em relação aos demais está, entretanto, no grupo de características de estilo. Esse grupo tenta modelar o estilo de escrita do texto para, assim, fazer a diferenciação entre estilos sarcásticos e não sarcásticos. Para fazer isso, eles utilizam três tipos de sequências textuais: n-gramas no nível de caracteres (ao invés de palavras) (abreviado pelos autores por *c-grams*), *skip-grams* (*s-grams*) e *skip-grams* de polaridade (*ps-grams*).

Os *c-grams* capturam a frequência de sequências de informação morfológica como sufixos e prefixos (como *-ly*, *-ing*, *dis-*, *un-*). Os *s-grams* funcionam como n-gramas, mas

com alguns pulos entre palavras e são utilizados para obter relações entre palavras não adjacentes no texto. Por exemplo, na sentença "Hoje está chovendo, quero tanto ir à praia", um bigrama é representado pela sequência "Hoje está", enquanto um bigrama com pulo de uma palavra seria "Hoje chovendo". Por fim, os *ps-grams* criam uma sequência de rótulos de positividade baseado nos *s-grams*. Por exemplo, "Hoje chovendo" seria rotulado como "pos-neg" (positivo e negativo).

Barbieri *et al.*, 2014, por sua vez, criar um dos conjuntos mais completos de características, com setes grupos. Esses grupos são: frequência, que mede o quão comum é cada palavra utilizada no texto; escrito-falado (da tradução literal de *written-spoken*, utilizado pelos autores em seu artigo), que funciona como a frequência, mas a frequência que a palavra é usada em diálogos escritos em oposição a falados; intensidade, que mede o uso de adjetivos e advérbios para aumentar ou diminuir a intensidade semântica de outras palavras no texto; estrutura, que mede, por exemplo, o número de caracteres usados, o número de palavras, o tamanho médio das palavras, o número de verbos, de substantivos, o número total de pontuações, o número de risadas, o número de vírgulas, pontos finais, sinais de exclamação, o número de emojis (ou emoticons), o número de organizações, pessoas, títulos e datas citados pelos texto; sentimento, que mede valores de positividade entre as palavras e usa algumas métricas como soma dos valores positivos, soma dos negativos, média da diferença entre positivos e negativos, entre outros; sinônimos, que calcula métricas relacionadas aos sinônimos das palavras usadas no texto, como a quantidade de sinônimos que uma determinada palavra tem e que possuem frequência de utilização menor do que ela; e ambiguidade, que mede a quantidade de sentidos possíveis que uma palavra pode ter.

Joshi, Sharma *et al.*, 2015 propõem uma nova abordagem baseada em uma teoria linguística chamada de incongruência de contexto. Eles se baseiam no fato de que o tempo para um ser humano processar um texto sarcástico depende do grau de incongruência presente nele e criam características o que chamam de incongruências explícitas e implícitas. A primeira é caracterizada por palavras de sentimentos opostos e a segunda é caracterizada por frases de sentimento implícito, que não se pode detectar através de uma única palavra.

Mishra *et al.*, 2016 seguem por uma abordagem bastante diferente das demais, que utiliza características baseadas no movimento produzido pelos olhos de pessoas lendo textos sarcásticos. Algumas das características propostas por eles são a fixação da visão em pontos do texto e saltos mais rápidos entre duas porções do texto do que em relação às demais. Além disso, eles utilizam um grafo de informação estrutural ao utilizar as palavras como vértices do grafo e os saltos de visão entre as palavras como arestas.

2.3.3 Abordagens Baseadas em Métodos de Aprendizado

Métodos de aprendizado são aqueles que usam modelos baseados em aprendizado de máquina. Dentro desse campo do conhecimento, existem alguns tipos de aprendizado. Dentre eles, os principais são o **aprendizado supervisionado**, o **aprendizado semi-supervisionado**, o **aprendizado não supervisionado** e o **aprendizado por reforço**. Pesquisadores já fizeram uso dos três primeiros tipos de aprendizados listados, mas nesta monografia ater-se-á apenas aos trabalhos envolvendo aprendizado supervisionado, pois

são os mais similares às técnicas aqui utilizadas. Para uma listagem mais vasta dos trabalhos correlacionados, recomenda-se a leitura dos trabalhos de Joshi, Bhattacharyya *et al.*, 2017 e Yaghoobian *et al.*, 2021.

Dentro do campos do algoritmos de aprendizado supervisionado estão os algoritmos de classificação, que como dito anteriormente, trata-se de achar uma função f que mapeie bem os exemplos X em seus respectivos rótulos y. A matriz X contém diversas características extraídas do texto, como as exploradas em 2.3.2. E o vetor de rótulos y contém números associados às classes de classificação. No caso da classificação binária, são apenas duas classes (que, em detecção de sarcasmo, são *sarcasmo* e *não sarcasmo*), mas em contextos mais gerais, podem haver várias classes, como em análise de sentimento. Nesse caso, as classes podem ser o sentimento predominante no texto e sarcasmo é apenas mais uma entre outras classes. Isso depende de como cada autor quer modelar o problema e quais aplicações se espera ter de resultado.

Em abordagens clássicas utilizando aprendizado de máquina, muitos trabalhos se baseiam em modelos bem conhecidos e estudados como *decision trees* (DT), *random forecasts* (RF), *support vector machines* (SVM), *naive Bayes* (NB) e *Neural Networks* (NN) (Yaghoobian *et al.*, 2021). Esses modelos são utilizados em várias áreas nas quais se aplica o aprendizado de máquina e são aplicados por muitos autores sem colocar seu foco principal em modificar o modelo ou criar modelos específicos para a detecção de sarcasmo. Em alguns outros trabalhos, entretanto, os autores procuram alternativas menos conhecidas ou modelos híbridos para conseguir melhores resultados.

Riloff *et al.*, 2013 utilizam uma abordagem híbrida que combina as previsões de uma SVM com um método baseado em contraste. Dessa forma, os autores combinam uma abordagem baseada em aprendizado com uma baseada em regras para criar um novo modelo. Nesse modelo, sempre que qualquer uma das duas abordagens classificar o texto como sarcástico, então o texto é classificado como sarcástico pelo modelo final.

Liebrecht *et al.*, 2013 fazem uso de um algoritmo menos conhecido chamado de *balanced winnow*, que é similar ao algoritmo *perceptron*. Esse algoritmo permite a investigação de quais características foram mais importantes para fazer a classificação. Dessa forma, é possível procurar quais os padrões mais importantes para o classificador e tentar entender por que esse é o caso.

Similarmente, árvores de decisão (do inglês *decision trees*) foram experimentadas por Reyes, Rosso e Veale, 2013 para poder investigar a ordem de decisões tomadas pela árvore para ver também quais características fazem a maior diferença e explicam mais o resultado. Além disso, eles usam também o modelo *naive Bayes*, pois seu artigo utiliza diversas características booleanas (com valores zero ou um).

Joshi, Tripathi *et al.*, 2016 partem para um estudo e abordagem diferentes. Ao invés de utilizar bases de dados de redes sociais, como o frequentemente utilizado Twitter, eles exploram a detecção de sarcasmo em diálogos da séries de TV "*Friends*". E argumentam que o problema de detecção de sarcasmo de diálogos é melhor modelado como uma tarefa de rotulação de sequências ao invés de uma tarefa de classificação de cada rótulo isoladamente.

Nessa modelagem, a ordem em que as falas de um diálogo aparecem é levada em

conta. Ao invés de tomar uma fala isoladamente e classificá-la, o modelo deve classificá-la utilizando todas as falas que vieram antes dela. Eles, então, utilizam dois modelos de aprendizado de máquina, o SEARN e o SVM-HMM e mostram que esses modelos de sequência performam melhor que os classificadores tradicionais.

P. Liu *et al.*, 2014 apresentam uma nova estratégia chamada por eles de *multi-strategy ensemble learning approach* (MSELA). Em seu trabalho, eles exploram a detecção de sarcasmo em um conjunto de dados bastante desbalanceado (ou seja, com uma proporção pequena de textos sarcásticos em relação aos não sarcásticos). Para lidar com esse problema, eles propõem essa estratégia.

A MSELA é constituída de três partes diferentes. A primeira é chamada de *sample-ensemble strategy* (em tradução livre, "estratégia de conjunto de amostragens") e consiste em fazer amostragens nos dados originais de forma a criar N conjuntos da dados balanceados. Cada um desses sub-conjuntos de dados é então usado para treinar pequenos classificadores, que consistem da segunda parte: a *classifier-ensemble strategy* (em tradução livre, "estratégia de conjunto de classificadores"). Nessa etapa, cada os classificadores de cada sub-conjunto de dados não necessariamente são os mesmos. Os autores empregam três tipos de classificadores (sendo que, em teoria, pode-se utilizar mais) e, para cada sub-conjunto, escolhem um classificador aleatório para ser treinado. Por fim, a terceira etapa serve para juntar cada uma das classificações em uma única e é chamada de *weighted voting strategy* (em tradução livre, "estratégia de voto ponderado"). Nessa última estratégia, as decisões de cada classificador são ponderadas por uma medida de entropia da informação e, dessa forma, unidas para resultar em uma última decisão de se o texto é ou não sarcástico.

Além do modelos de aprendizado tidos como mais "clássicos", modelos de aprendizado profundo (ou seja, aqueles que utilizam grandes redes neurais) também foram utilizados. Esse tipo de modelo costuma resultar em melhores métricas, mas também exige uma quantidade muito grande de dados. Isso por muitas vezes é um problema, pois a maioria dos conjuntos de dados é rotulado por seres humanos que decidem se um texto é ou não sarcástico utilizando seus próprios conhecimentos.

A. Ghosh e Veale, 2016 utilizam três tipos de redes neurais em conjunto e comparam seu desempenho em um conjunto de dados extraído do Twitter. Eles utilizam *long short-term memories* (LSTMs), *convolutional neural networks* (CNNs) e *deep neural networks* (DNNs), combinando esses modelos entre si para conseguir melhores resultados.

Van Hee *et al.*, 2018 mostram em seu artigo os resultados da terceira tarefa da SemEval-2018, um *workshop* internacional de PLN com o objetivo de avançar as fronteiras do estado-da-arte em determinadas tarefas. Nesse artigo, eles mostram que as melhores soluções (em termos da métrica F_1) propostas pelos times utilizam modelos baseados em aprendizado profundo. Eles fazem principalmente o uso de redes do tipo LSTMs, CNNs e DNNs. Além disso, é presente o uso de *word embeddings* como o GloVe, que retratam cada uma das palavras presentes em um texto como vetores reais nos quais palavras de sentidos parecidos possuem vetores parecidos.

A tarefa proposta pela SemEval-2018 era dividida em duas sub-tarefas. A primeira era uma detecção de sarcasmo comum, onde bastava o modelo distinguir entre *tweets*

sarcásticos e não sarcásticos. A segunda tarefa, entretanto, era mais complexa, pois exigia que o modelo também classificasse o tipo de sarcasmo que ocorria no texto. Os resultados mostraram que mesmo modelos de aprendizado profundo bastante sofisticados ainda estavam bastante longe de resultados promissores. Enquanto na primeira sub-tarefa o melhor time obteve uma métrica F_1 de 0.705, o melhor time na segunda sub-tarefa obteve apenas 0.507 pontos dessa mesma métrica.

2.3.4 Abordagens Baseadas em Contexto

Até então, foram exibidos trabalhos que se restringem apenas ao texto de interesse. Entretanto, muitos outros trabalhos partem de outra abordagem, que é utilizar informações além do texto para se determinar se um texto é ou não sarcástico, chamadas de contexto.

Wallace *et al.*, 2014 mostram em seu artigo que humanos obtém melhores resultados para detectar se um texto é ou não sarcástico quando eles recebem contexto além do texto original. Em seu trabalho, eles exploram a rede social Reddit e pedem para seres humanos decidirem se uma postagem nessa rede é ou não sarcástica e qual o seu grau de confiança nessa decisão. O humano pode pedir por contexto quando estiver em dúvida e os autores, então, apresentam qual era o tópico em que a postagem estava inserida e uma lista de outras postagens do mesmo usuário.

Eles então mostram que serem humanos acertam consistentemente mais quando obtém esse contexto extra, além de ficarem mais confiantes em sua decisão. Não só isso, mas os autores também mostram que um modelo *bag-of-words* costuma errar nesses casos em que humanos precisam de contexto extra. Portanto, os autores argumentam que máquinas provavelmente também precisam de contexto extra para conseguir melhores resultados. Nos anos seguintes, muitos outros autores criaram modelos que utilizam o contexto de uma postagem e provam empiricamente que o contexto pode ajudar.

Há muitas formas de se adicionar informações extras referentes ao contexto. Hazarika *et al.*, 2018 criam um método para criar representar *embeddings* de usuários, que capturam o estilo de descrita e personalidade para criar uma representação que pode ser utilizada em adição ao texto escrito por aquele usuário.

Kolchinski e Potts, 2018 exploram outra forma de se criar *embeddings* para os usuário de uma rede social. Para cada autor nos dados de treinamento, ele criam um vetor de 15 valores randômicos. Então esses vetores são passados como entrada para um modelo de rede neural recorrente (RNN) bidirecional e são atualizados durante o treinamento, com o objetivo de aprender características importantes do usuário. Então se aquele usuário aparece novamente na hora de fazer uma predição fora dos dados de treinamento, seu vetor de *embeddings* é utilizado. Contudo, caso o autor não tenha aparecido no treinamento, um vetor randômico é utilizado.

Wang *et al.*, 2015 utilizam o contexto da conversa e do tópico, utilizando os *tweets* que vieram antes do *tweet* de interesse, outros *tweets* do mesmo autor e outros *tweets* que tenham as mesmas *hashtags* do *tweet* de interesse (ou seja, que pertençam ao mesmo tópico).

O melhor resultado obtido por eles é quando utilizam o histórico de *tweets* do autor. Eles experimentam com o último *tweet* do autor, os três últimos e os cinco últimos, obtendo melhores resultados conforme aumentavam esse número. Além disso, utilizar o contexto da conversa (ou seja, o histórico de respostas de que levou até o *tweet* de interesse) possui também um ganho nas métricas, mas variar entre 1, 3 ou 5 *tweets* não muda muito. Isso mostra que se o modelo tivesse acesso apenas ao *tweet* de interesse (que queremos determinar sarcasmo) e o *tweet* que ele respondeu (o anterior a ele na conversa) já teria a maior parte do ganho que se obteria ao olhar para a conversa toda. Intuitivamente, isso faz sentido, pois quando se dá uma resposta sarcástica, essa é uma resposta àquilo que foi falado imediatamente antes na vasta maioria dos casos. Por fim, eles mostram que a métrica F_1 cai ao adicionar contexto do tópico e que, portanto, adicionar outros *tweets* com a mesma *hashtag* do de interesse apenas atrapalha.

D. Ghosh *et al.*, 2018 também utilizam contexto da conversa em conjunto com LSTMs e mostram que ao adicionar a postagem que foi respondida pela postagem de interesse melhora os resultados. Eles utilizam vários conjuntos de dados e várias formas diferentes de LSTMs, assim, eles testam diversas maneiras de se acoplar o comentário anterior ao de interesse e mostram que adicionar o contexto referente à postagem anterior sempre melhora os resultados.

Kumar Jena *et al.*, 2020 criam um novo modelo chamado por eles de C-Net que foca também em utilizar o contexto gerado pelos comentários anteriores de um comentário de interesse. Dado que eles tenham $n - 1$ comentários anteriores, eles criam n modelos BERT, um para cada comentário. Assim, em uma cadeia de n comentários (onde o n-ésimo é o comentário de interesse, o $n - 1$-ésimo é o comentário anterior a ele e assim por diante), o i-ésimo BERT é treinado utilizando apenas o i-ésimo comentário e deve retornar uma porcentagem y_i de que o comentário i leve a uma resposta sarcástica. Depois, eles agregam esses resultados dos n modelos utilizando uma camada de fusão (do inglês, *fusion layer*) que utiliza algum modelo que recebe n valores e retorna uma probabilidade final y_{n+1} que será utilizada para dizer se o último comentário foi ou não sarcástico. Para fazer essa concatenação, eles utilizam um modelo de séries temporais chamado de *simple exponential smoothing* (SES).

Por fim, há outras duas formas de se adicionar contexto à tarefa. A primeira delas é utilizando *word embeddings*, pois elas permitem aos modelos que as utilizam fazer comparações entre palavras e compreender a semântica passada por elas. Alguns exemplos são `Word2vec` (Mikolov *et al.*, 2013), `GloVe` (Pennington *et al.*, 2014), `fastText` (Bojanowski *et al.*, 2016), `ELMo` (Peters *et al.*, 2018), `BERT` (Devlin *et al.*, 2018). A segunda é pelo uso de uma técnica chamada de transferência de aprendizado (ou aprendizado por transferência) (do inglês, *transfer learning*), que consiste de um modelo treinado em tarefas genéricas utilizando um conjunto de dados realmente grande. Esse modelo passa a ser chamado de **pré-treinado** e seus parâmetros são salvos para serem refinados (do inglês, *fine-tuning*) depois em uma tarefa específica. Dentro do contexto de PLN, os modelos mais utilizados são os conhecidos como *transformers* (explicados em 2.4). Alguns exemplos são `GPT-3` (Brown *et al.*, 2020), `XLNet` (Yang *et al.*, 2019), `BERT` (Devlin *et al.*, 2018), `RoBERTa` (Y. Liu *et al.*, 2019) e `DeBERTa` (He, X. Liu *et al.*, 2020), que é o foco deste trabalho.

Entre os trabalhos que utilizam *transformers*, um dos mais bem sucedidos é o de Potamias *et al.*, 2020, que criam um modelo que utiliza um RoBERTa em conjunto com uma LSTM bidirecional (BiLSTM). Os autores argumentam que o RoBERTa é capaz de mapear as palavras para um espaço vetorial rico em semântica e que outros modelos de redes neurais podem usar esses valores para capturar dependências tempo-espaciais entre as palavras. Eles, portanto, removem a última camada do modelo pré-treinado RoBERTa e passam os valores da sua última camada escondida (do inglês, *hidden layer*) para uma BiLSTM que, por sua vez, tem seus resultados concatenados por uma camada completamente conectada e passados para uma camada *softmax*.

2.4 Modelos de Redes Neurais *Transformers*

A proposta desse trabalho é aplicar um tipo de *transformer* chamado de DeBERTa (He, X. Liu *et al.*, 2020) e comparar seu resultado com outros *transformers*, que hoje constituem o estado-da-arte em muitas tarefas de PLN (entre elas, a detecção de sarcasmo) (Joshi, Bhattacharyya *et al.*, 2017). Essa seção tem por objetivo apresentar em maiores detalhes o funcionamento desse tipo de arquitetura de redes neurais (NN) e apresentar os *transformers* mais conhecidos e utilizados.

2.4.1 RNNs

Para se chegar em como os *transformers* foram criados, é interessante dar um panorama histórico do estado-da-arte no processamento de linguagem natural e, em especial, na tarefa de tradução automática. Devido à característica sequencial de qualquer texto, redes neurais recorrentes (RNNs) são a principal escolha para esse tipo de tarefa.

Uma RNN é uma rede neural que consiste de um estado escondido (do inglês, *hidden state*) h e uma saída opcional y que recebe como entrada uma sequência de tamanho variável $x = (x_1, x_2, \ldots, x_{T_x})$. A cada passo temporal t, o estado escondido h_t da RNN é atualizado pela função f:

$$h_t = f\left(h_{t-1}, x_t\right), \tag{2.3}$$

onde f é uma função de ativação não linear que pode ser tão simples quando uma sigmoide ou tão complexa quanto uma unidade *long short-term memory* (LSTM), introduzida por Hochreiter e Schmidhuber, 1997 em 1997.

A RNN, então, consegue aprender uma distribuição de probabilidade com base na sequência de entrada ao ser treinada para predizer o próximo elemento da sequência. Portanto, ao receber uma sequência $x = \left(x_1, x_2, \ldots, x' t-1\right)$, ela consegue retornar a probabilidade do próximo elemento da sequência ser j ao utilizar uma função de ativação do tipo *softmax* em sua saída:

$$p\left(x_{t,j} = 1 \mid x_1, x_2, \ldots, x_{t-1}\right) = \frac{\exp\left(w_j h_t\right)}{\sum_{j'=1}^{K} \exp\left(w_{j'} h_t\right)}, \tag{2.4}$$

onde w_j é a j-ésima linha de uma matriz de pesos W.

(imagem RNN)

Utilizando esse tipo de rede neural, Cho *et al.*, 2014 propõem uma nova arquitetura chamada de *RNN Encoder-Decoder*, que utiliza uma RNN para codificar a sequência de entrada x em um único vetor c de dimensão pré-definida chamado de **contexto** (não confundir com contexto já mencionado em 2.3.4), que, por sua vez, é passado como entrada para uma segunda etapa de decodificação, que produz uma nova sequência y. No caso de tradução de texto, a sequência x é o texto a ser traduzido e a sequência y é o texto após tradução.

(imagem RNN E-D)

Diferentemente de 2.3, o estado escondido do decodificador, s_t, não é uma função da entrada, mas sim do contexto gerado pelo codificador e também pelo último elemento da saída produzido:

$$s_t = f\left(s_{t-1}, y_{t-1}, c\right), \tag{2.5}$$

e o próximo elemento da saída é gerado utilizando a função de distribuição dada por:

$$P(y_t \mid y_1, y_2, \ldots, y_{t-1}, c) = g\left(y_{t-1}, s_t, c\right), \tag{2.6}$$

onde f e g são funções de ativações e g deve ser uma função que produz probabilidades válidas (como, por exemplo a *softmax*). Vale ainda ressaltar que para escolher y_t, o próximo elemento da sequência de saída do modelo, pode-se utilizar várias estratégias. As mais conhecidas são escolher y_t como o elemento de maior probabilidade da função de distribuição ou então escolher um elemento aleatoriamente ponderando pelas probabilidades de cada elemento (isto é, elementos com valores de probabilidade mais altos dados pela distribuição têm maiores chances de serem escolhidos como o próximo elemento da sequência de saída).

2.4.2 Mecanismos de Atenção

Bahdanau *et al.*, 2014 estendem a arquitetura de Cho *et al.*, 2014 criando um mecanismo que mais tarde ficaria conhecido como **mecanismo de atenção**.

Primeiramente, eles o fato de que o vetor c é fixo em todas as iterações de cálculo do novo estado escondido. Cada s_i é uma função de um vetor de contexto c_i:

$$s_i = f\left(s_{i-1}, y_{i-1}, c_i\right). \tag{2.7}$$

O vetor de contexto, por sua vez, é criado como sendo uma combinação linear dos estados gerados pelo codificador, $\{ h_1, h_2, \ldots, h_{T_x} \}$, ponderado por valores α_{ij}:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{2.8}$$

Os valores α_{ij} são calculados de maneira que formem uma distribuição de probabilidades. Ou seja, cada valor α_{ij} está entre zero e um e a sua soma em j vale um. Isso é garantido ao utilizar a função *softmax*:

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)}, \tag{2.9}$$

onde

$$e_{ij} = a\left(s_{i-1}, h_j\right). \tag{2.10}$$

A função a por sua vez é uma função que retorna uma pontuação para quanta atenção o modelo deve prestar ao redor da posição j da entrada para predizer a posição i da saída. No artigo original de Bahdanau *et al.*, 2014 é utilizada uma rede completamente conectada que é treinada juntamente com o restante do modelo.

Para resumir antes de prosseguir, o codificador itera sobre os valores da sequência de entrada $x = (x_1, x_2, \ldots, x_{T_x})$ e gera uma sequência $h = (h_1, h_2, \ldots, h_{X_T})$ que representa aquela entrada, mas codificada através da equação 2.3 (no artigo original, os autores na verdade utilizam uma BiRNN, mas por propósitos didáticos, isso foi abstraído - COLOCAR ISSO COMO FOOTNOTE).

Na arquitetura comum de um *encoder-decoder*, os valores de h são utilizados diretamente para gerar a saída. Entretanto, Bahdanau *et al.*, 2014 propõem que exista uma camada intermediária entre o codificador e decodificador que faz com que o decodificador foque sua atenção (intuitivamente falando) em elementos específicos da sequência h. Para tanto, o i-ésimo contexto, c_i, que gera a i-ésima saída y_i, é criado através de uma soma que percorre cada elemento h_j e o multiplica por uma probabilidade α_{ij} de h_j ser relevante na predição de y_i.

Após essa nova proposta de Bahdanau *et al.*, 2014, muito são os trabalhos que utilizam, adaptam e incrementam o mecanismo de atenção. Galassi *et al.*, 2021 fazem uma revisão de diversos mecanismos de atenção utilizados na literatura e propõem um modelo genérico que engloba grande parte dos trabalhos que fazem uso desse mecanismo.

2.4.3 *Transformers*

Mecanismos de atenção são utilizados em conjunto a uma RNN, de maneira similar a Bahdanau *et al.*, 2014, por muitos outros trabalhos. RNNs, entretanto, possuem um problema intrínseco em sua arquitetura, que é o fato de serem recorrentes. Isso significa que uma RNN precisa do valor do estado escondido anterior h_{t-1} para calcular o novo estado h_t (seja no codificador ou decodificador). Dessa forma, não se pode paralelizar sua computação facilmente. Vaswani *et al.*, 2017 criam, então, um novo modelo chamado de *transformer* [1].

Essa nova arquitetura se baseia apenas em uma versão nova do mecanismo de atenção para criar um modelo do tipo *encoder-decoder*, que pode ser aplicado sobre uma sequência, de forma análoga a uma RNN. Esse novo modelo de atenção é chamado de *self-attention* (em tradução literal, "*atenção a si próprio*").

Nele, todos os elementos da sequência de entrada podem ser processados ao mesmo tempo pelo codificador e os de saída pelo decodificador. Entretanto, suponha que se quer prever o elemento y_i da sequência de saída. Nesse caso, o *transformer* recebera a sequência x e a sequência correta de saída y, mas os elementos y_j com $j \geq i$ são mascarados para simular uma RNN, que não tem acesso a esses valores (e também limitar a trivialidade que seria predizer o elemento y_i com esse elemento dado como entrada para o modelo).

Para dar um exemplo, suponha que a tarefa é de tradução do português para o inglês e que se deseja traduzir a frase "Atenção é tudo que você precisa" para a frase "*Attention is all you need*". Portanto, tem-se $x = (\texttt{Atenção}, \texttt{é}, \texttt{tudo}, \texttt{que}, \texttt{você}, \texttt{precisa})$ e $y = (\textit{Attention}, \textit{is}, \textit{all}, \textit{you}, \textit{need})$. Em caso de se desejar prever o terceiro elemento da sequência, y_3, o valor de entrada é a própria sequência x, mas a entrada da saída gerada até agora é $y = (\textit{Attention}, \textit{is}, \texttt{<MASK>}, \texttt{<MASK>}, \texttt{<MASK>})$, onde `<MASK>` é um token especial que acaba recebendo uma atenção de menos infinito. Dessa forma, portanto, o modelo consegue o mesmo resultado dos modelos anteriores, mas com uma arquitetura que pode ser paralelizada para alcançar uma velocidade de processamento maior.

(img transformer)

O mecanismo de atenção do transformer é um pouco diferente e é chamado de *scaled dot-product attention* (do inglês, atenção de produto escalar escalado). Assim como na equação 2.8, a atenção é calculada fazendo uma média ponderada entre os valores atribuídos aos elementos da sequência. Vaswani *et al.*, 2017, entretanto, modificam como os pesos de ponderação são calculados. Para isso, eles introduzem a equação

$$\texttt{Attention}(Q, K, V) = \texttt{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{2.11}$$

Nela, os valores Q, K e V são matrizes que representam respectivamente *consultas*, *chaves* e *valores* (do inglês, *queries*, *keys* e *values*). Essa função de atenção é aplicada um número h de vezes na mesma entrada, mas resultando em consultas, chaves e valores

[1] o termo *transformer* é uma palavra em inglês que significa "transformador", mas nesta monografia opta-se por utilizar o próprio termo em inglês *transformer*

diferentes. Isso pois um tipo de atenção pode relacionar elementos que sejam das mesmas classes sintáticas e replicar o comportamento de alguns dos métodos de regras vistos na sessão 2.3.1, enquanto outras atenções podem olhar para a semântica de termos relacionados.

Essas h "cabeças" de atenção possuem seus resultados concatenados em uma saída única que pode ser processada pela próxima camada da rede neural. Essa camada cheia de cabeças recebe, portanto, o nome de atenção multi-cabeça (do inglês, *multi-head attention*).

(img atenção multi-cabeça)

No codificador, Q, K e V são todas providas pelo próprio codificador (ou pela sequência de entrada, no cabeço da primeira camada). Já no decodificador, os valores de Q e K são providos pelo codificador e os valores V são criados pelo próprio decodificador com base na sequência esperada de saída. Dessa forma, a rede presta atenção na sequência de saída, mas levando como base a sequência de entrada.

Por fim, é importante mencionar que, apesar de resolver o problema do processamento sequência anteriormente mencioado, os *transformers* criam um novo problema que é o de não representar adequadamente a característica sequêncial de suas entradas. Como não se utiliza RNNs, pode-se pensar em um *transformer* como se fosse uma rede neural completamente conectada (DNN). Esse tipo de rede não é bom em saber que existe relação entre o elemento x_1 e x_2 e que, essa relação é uma métrica com o dobro de valor para x_1 e x_3. Para dar uma solução parcial a esse problema, os autores propõem uma codificação posicional que deve ser adicionada à representação original das palavras. Portanto, o valor da entrada que o modelo realmente recebe é $x_i + PE(i)$ para a entrada i.

O modelo completo é muito mais complexo do que o apresentado aqui e recomenda-se ao leitor a leitura completa do artigo por Vaswani *et al.*, 2017 caso ele tenha mais interesse no assunto.

2.4.4 BERT

Para finalizar o capítulo, falar-se-á de um dos modelos de redes *transformers* que é vastamente aplicado em problemas de NLP e que é uma das bases para o modelo explorado nesse trabalho. Este *transformer* é o `BERT`, que é uma sigla para *Bidirectional Encoder Representations from Transformers* (em tradução livre, "representações de codificação bidirecional de *transformers*") (Devlin *et al.*, 2018).

A principal inovação desse modelo é que ele torna o treinamento bidirecional através de um objetivo de treinamento denominado *masked language model* (MLM) (do inglês, modelo de linguagem mascarada). Nesse tipo de treinamento, alguns dos *tokens* da sequência de entrada são aleatoriamente mascarados (isto é, trocados por um *token* específico `<MASK>`) e a tarefa alvo do treinamento é justamente identificar quais eram as palavras originais, antes de serem mascaradas.

Isso faz com que o modelo possa utilizar o contexto bidirecionalmente, ou seja, observar as palavras ao redor da palavra alvo para predizê-la. Por exemplo, na frase "A roda do meu `<MASK>` quebrou novamente", as palavras "roda" e "quebrou" são importantes para supor que `<MASK>` corresponde à palavra "carro" ou algum automóvel similar. De forma análoga,

o modelo BERT pode utilizar essas palavras ao redor da palavra mascarada para estimar qual palavra ela era originalmente.

As aplicações desse modelo são vastas, mas para o escopo deste trabalho, basta saber que ele pode ser utilizado também na detecção automática de sarcasmo (Yaghoobian *et al.*, 2021).

Capítulo 3

Materiais e Métodos

Este capítulo abordará os principais recursos utilizados na experimentação com o modelo DeBERTa na tarefa de detecção automática de sarcasmo. Inicialmente, falar-se-á um pouco sobre o modelo em si e como ele aborda alguns dos problemas anteriormente encontrados na literatura. Depois, abordar-se-á o conjunto de dados utilizado nos experimentos. Por fim, discutir-se-á o código em si e as ferramentas pré-existentes que foram utilizadas neste trabalho.

3.1 DeBERTa

O modelo DeBERTa foi criado por um grupo de pesquisadores da Microsoft e publicado em 2021 (He, X. Liu *et al.*, 2020). Seu nome é uma sigla que significa "*Decoding-Enhanced BERT with Disentangled Attention*" (em tradução livre, "BERT com decodificação melhorada por atenção decomposta"). Como o nome já sugere, o DeBERTa foi criado com o modelo BERT (Devlin *et al.*, 2018) como uma base, mas também tem principal inspiração o modelo RoBERTa (Y. Liu *et al.*, 2019), que é outro modelo que propõe melhorias para o BERT.

Suas principais contribuições para a área são duas: *disentangled attention* (em tradução livre, "atenção decomposta") e *enhanced mask decoder* (em tradução livre, "decodificador de máscara melhorado") [1].

A primeira e mais importante contribuição é a *disentangled attention*. Como visto em 2.4.3, os modelos *transformers* possuem o problema de não conseguirem mapear muito bem a distância entre os elementos da sequência de entrada. Assim como no artigo original de Vaswani *et al.*, 2017, o modelo BERT também utiliza uma soma da codificação dos elementos da sequência com uma codificação de posições da sequência. Ao somar entretanto, perde-se um pouco do conteúdo original de cada uma das codificações. No mecanismo de *disentangled attention*, o *conteúdo* (ou seja, a codificação dos elementos da sequências - em geral, palavras) e a *posição* são representados utilizando dois vetores diferentes. E as

[1] Assim como alguns outros termos utilizados nessa monografia, optou-se por não traduzir *disentangled attention* e *enhanced mask decoder* pelo fato de serem termos muito novos na literatura e não terem uma tradução canônica para o português.

atenções entre as palavras são computadas utilizando também matrizes distintas e baseadas no conteúdo das palavras e na sua distância relativa.

Utilizando a própria notação presente no artigo original do modelo (He, X. Liu *et al.*, 2020), um elemento na posição i de uma sequência é representado por dois vetores $\{H_i\}$ e $\left\{P_{i|j}\right\}$, que representam respectivamente o conteúdo do elemento e a sua posição relativa com um outro elemento na posição j. Então, a atenção cruzada que os elementos i e j prestam entre si (essa relação é simétrica) é calculada pelo produto

$$\begin{aligned} A_{i,j} &= \left\{H_i,\ P_{i|j}\right\} \times \left\{H_j,\ P_{j,i}\right\}^T \\ &= H_i H_j^T + H_i P_{j|i}^T + P_{i|j} H_j^T + P_{i|j} P_{j|i}^T. \end{aligned} \tag{3.1}$$

Portanto, no caso em que as informações relativas ao conteúdo e posição dos elementos, a atenção total entre i e j pode ser calculada como quatro outras atenções: conteúdo-para-conteúdo, conteúdo-para-posição, posição-para-conteúdo e posição-para-posição. Em seu artigo, os autores argumentam que apenas as três primeiras atenções são realmente úteis e, portanto, descartam o caso de atenção entre posições relativas.

Vale a notar que os autores fazem uma leve mudança de nomenclatura em seu artigo. Enquanto originalmente, Vaswani *et al.*, 2017 chamam de atenção o resultado da equação 2.11. He, X. Liu *et al.*, 2020 chamam de atenção apenas o resultado da multiplicação das consultas, Q, pelas chaves K. Para fazer uma comparação, no modelo de *self-attention* original, tem-se:

$$Q = H W_q,\ K = H W_k,\ V = H W_v,\ A = \frac{Q K^T}{\sqrt{d}}$$

$$H_o = \texttt{softmax}(A) V,$$

onde H representa os vetores de conteúdo, H_o a saída da *self-attention*, W_q, W_k, W_v são matrizes de pesos aprendidas pela rede neural e A é a matriz de atenção.

No modelo de *disentangled attention*, entretanto, a atenção é calculada como

$$\begin{aligned} &Q_c = H W_{q,c},\ K_c = H W_{k,c},\ V_c = H W_{v,c},\ Q_r = P W_{q,r},\ K_r = P W_{k,r} \\ &\tilde{A}_{i,j} = \underbrace{Q_i^c K_j^{cT}}_{\text{conteúdo-para-conteúdo}} + \underbrace{Q_i^c K_{\delta(i,j)}^{rT}}_{\text{conteúdo-para-posição}} + \underbrace{K_j^r Q_{\delta(j,i)}^{cT}}_{\text{posição-para-conteúdo}} \\ &H_o = \texttt{softmax}\left(\frac{\tilde{A}}{\sqrt{3d}}\right) V_c, \end{aligned} \tag{3.2}$$

onde $\tilde{A}$ é a matriz de atenção e $\tilde{A}_{i,j}$ representa a atenção do elemento i para o elemento j, Q_i^c é a i-ésima linha de Q_c, K_j^c é a j-ésima linha de K_c e $\delta(i,j)$ é uma função que dá a distância relativa entre i e j (para mais detalhes, ler o artigo original).

Com essa nova forma de calcular a atenção, os autores conseguem dividir as rotas utilizadas pela informação de forma que o modelo consiga ter aceso a informação das posições relativas e conteúdos do texto de forma "limpa", sem ser por meio de uma soma como feito em trabalhos anteriores.

Por fim, a segunda contribuição desse modelo é a *enhanced mask decoder*, que é quase uma consequência direta da primeira contribuição mencionada anteriormente. Ao desacoplar conteúdo e posição, os autores utilizam as posições relativas entre as palavras, mas esse modelo acaba perdendo as informações das posições absolutas do texto. Por exemplo, na frase "A roda do meu carro quebrou novamente", o modelo saberia apenas que "roda" e "carro" possuem uma distância de 3 posições entre si, mas não saberia que "roda" está na segunda posição do texto. Para endereçar esse problema, o modelo precisa, em algum ponto de sua arquitetura, receber a informação da posição absoluta das palavras no texto de entrada.

O modelo BERT recebia essa informação logo no começo, através de uma soma dos vetores, como já mencionado. Mas o modelo DeBERTa adiciona essa informação apenas no último passo da arquitetura, logo antes de uma camada `softmax` predizer qual é o elemento mascarado que se deseja predizer. Portanto, o DeBERTa utiliza apenas as posições relativas entre as palavras durante toda sua arquitetura e recebe a informação das posições absolutas apenas como um complemento para decodificar as palavras mascaradas. Os autores fazem um estudo empírico e mostram que essa abordagem é mais eficiente do que a abordagem utilizada pelo BERT e outros modelos anteriores a esse estudo.

3.1.1 DeBERTa V3

Este trabalho utiliza a nova versão do DeBERTa, o DeBERTaV3 He, Gao *et al.*, 2021, que é uma melhoria da versão original utilizando ideias apresentadas por Clark *et al.*, 2020 no modelo ELECTRA.

O ELECTRA é um modelo que utiliza uma proposta de trocar o *masked language model* (MLM) utilizado pelo BERT (2.4.4) e demais modelos que o têm o referência (como RoBERTa e DeBERTa) por um novo modelo chamado de *replaced token detection* (RTD) (do inglês, detecção de símbolo substituído). Nesse tipo de modelagem, ao invés de substituir alguns *tokens* aleatoriamente por um símbolo especial `<MASK>`, troca-se símbolos válidos por outros símbolos válidos, mas que são gerados por um gerador. Então, um discriminador é treinado na tarefa de reconhecer quais símbolos foram trocados pelo gerador e não são os símbolos originais da sequência. Ao utilizar esse novo tipo de modelagem, os autores demonstram ganhos significativos na efetividade do modelo.

Além disso, nesse novo modelo, os autores melhoram o modelo ELECTRA ao propor uma nova forma de treinamento chamada de *gradient-disentangled embedding sharing* (em tradução livre, *compartilhamento de representação com gradiente desagradado*). Os autores argumentam que o compartilhamento das *embeddings* entre o gerador e discriminador é bastante prejudicial para a tarefa que ambos pretendem realizar, pois o gerador deseja representar palavras com semânticas similares de forma próxima, enquanto o discriminador deseja fazer o oposto. Dessa forma, eles propõem uma nova forma de otimização da rede que desacopla o gradiente utilizado no discriminador do gradiente usado pelo gerador.

3.2 Conjunto de dados SARC

Para realizar a detecção de sarcasmo, é necessário um conjunto de dados rotulados. Muitos trabalhos na literatura utilizam o Twitter (ver 2), mas há vários trabalhos que, mais recentemente, têm utilizado o Reddit.

Esta é uma rede social baseada em comunidades. Cada comunidade é chamada de *subreddit*, e, muitas vezes, referencia-se a elas por tópicos, pois o nome do *subreddit* define o tipo de publicações que os usuários fazem dentro dele. Por exemplo, em `gaming` as pessoas falam sobre jogos, em `datascience` as pessoas falam sobre ciência de dados (os nomes geralmente são bastante autoexplicativos). Além disso, cada postagem feita em um *subreddit* pode receber comentários e cada comentário, por sua vez, pode receber sub-comentários (formando uma estrutura recursiva).

Khodak *et al.*, 2017 em 2017 introduzem um *corpus* de textos retirados do Reddit que recebeu o nome de SARC, uma sigla para *Self-Annotated Reddit Corpus* (em tradução livre, "corpus do Reddit auto-anotado"). Esse conjunto de dados revolucionou a detecção de sarcasmo, pois foi o primeiro a passar de um milhão de comentários sarcásticos (e mais de quinhentos milhões de comentários no total). Esse marco se deve ao fato de este conjunto de dados ser o primeiro a utilizar a estrutura de comentários do Reddit e rotular o sarcasmo automaticamente, com base em anotações dos próprios autores dos comentários.

Para rotular um comentário como sarcásticos, os autores utilizam o fato de que muitos usuários do Reddit utilizam a marcação `/s` em um comentário para denotar que estão sendo sarcásticos (de forma similar à *hashtag* "*#sarcasm*" no Twitter) (Reddit, 2022). Dessa forma, todo comentário encontrado pelos autores com essa marcação é anotado como sendo sarcástico, enquanto comentários sem essa anotação são marcados como não sarcásticos.

Os autores, entretanto, reveem seu trabalho e apontam que há uma certa quantidade de erros cometidos por essa abordagem. Por utilizarem uma metodologia automática de rotulação, o próprio conjunto de dados acaba possuindo erros do tipo I e II. Os falsos positivos acontecem quando o algoritmo proposto encontra um comentário com "/s", mas não é sarcástico. Esse tipo de erro pode acontecer, por exemplo, quando alguém não conhece essa anotação, quando alguém está referenciando a anotação (como para avisar que um outro usuário esqueceu de colocá-la ou explicando/perguntando seu significado), quando se utiliza a *tag* de *HTML* `<s>...</s>`. Os falsos negativos acontecem, por sua vez, nos comentários que são sarcásticos, mas não utilizam a marcação. Isso acontece principalmente por dois motivos: o usuário não conhecer a convenção de se utilizar "/s" ou por achar que seu sarcasmo é óbvio o suficiente para que ele não precise utilizar.

Como forma de minimizar esses erros, os autores propõem algumas filtragens. Algumas delas são bastante padrões como remover URLs e limitar os caracteres aos da tabela ASCII. Outro filtro é remover os comentários que aparecem após um comentário sarcástico na árvore de comentários (pois os autores afirmam que esses comentários são extremamente ruidosos). Além disso, os autores apenas coletam dados de usuários que estão cientes do uso de "/s", verificando se eles já utilizaram a notação anteriormente. Eles também lidam com os falsos positivos mantendo apenas os comentários que contém a notação "/s" no final do comentário.

Esses filtros são, então, testados manualmente utilizando uma amostragem de 1000 comentários (500 sarcásticos e 500 não sarcásticos). Os autores demonstram seu procedimento resultou em uma taxa de 1.0% de falsos positivos e 2.0% de falsos negativos. Por fim, vale dizer que os autores, por motivos óbvios, removem as anotações "/s" dos comentários.

3.3 Código e ferramentas

Essa seção comenta um pouco sobre as ferramentas utilizadas e o código fonte[2].

As duas principais ferramentas utilizadas foram evidentemente o modelo DeBERTa, disponibilizado em HuggingFace e o conjunto de dados SARC disponibilizado em SARC.

Para fazer o *download* e preprocessamento do conjunto de dados, os arquivos `dataset_downloader.py`, `dataset_unhasher.py` e `download_and_preprocess_data.py` são utilizados, todos escritos na linguagem de programação `Python`, assim como o projeto em sua totalidade. O arquivo `download_and_prerocess_data.py` é um executável que utiliza as variáveis globais definidas em `global_vars.json` para definir o caminho onde ficarão os arquivos do conjunto de dados (por padrão, ficam na pasta `SARC_dataset`). Há também o arquivo `preloaded_dataset_downloader.py`, que baixa o conjunto de dados já preprocessados e prontos para serem utilizados pelo projeto.

Para fazer o treinamento de um modelo, utiliza-se o arquivo `run_train.py`, que recebe o nome do modelo e outros parâmetros de treinamento. Ao final do treinamento, o modelo é salvo utilizando o nome passado como entrada para o executável.

Para testar um modelo, utiliza-se o arquivo `run_test.py`, que recebe o nome de um modelo e realiza seu teste no conjunto de dados de teste. Ao final do testes as métricas são salvas, assim como as predições do modelo.

Por fim, o arquivo `baseline_model.py` é utilizado para fazer a comparação dos modelos de arquitetura *transformers* com um modelo mais clássico de *machine learning* que utiliza Bag-of-Words. Dessa forma, pode-se comparar a eficácia desses modelos.

[2] disponível em https://github.com/Giatroo/TCC/tree/main/Project

Capítulo 4

Avaliação Experimental e Discussão dos Resultados

Capítulo 5

Conclusão

Referências

[Bahdanau *et al.* 2014] Dzmitry Bahdanau, Kyunghyun Cho e Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. doi: 10.48550/ARXIV.1409.0473. url: https://arxiv.org/abs/1409.0473 (citado nas pgs. 17–19).

[Barbieri *et al.* 2014] Francesco Barbieri, Horacio Saggion e Francesco Ronzano. "Modelling sarcasm in Twitter, a novel approach". Em: *Proceedings of the 5th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. Baltimore, Maryland: Association for Computational Linguistics, jun. de 2014, pgs. 50–58. doi: 10.3115/v1/W14-2609. url: https://aclanthology.org/W14-2609 (citado na pg. 11).

[Bharti *et al.* 2015] Santosh Kumar Bharti, Korra Sathya Babu e Sanjay Kumar Jena. "Parsing-based sarcasm sentiment recognition in twitter data". Em: *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 2015, pgs. 1373–1380. doi: 10.1145/2808797.2808910 (citado na pg. 9).

[Bojanowski *et al.* 2016] Piotr Bojanowski, Edouard Grave, Armand Joulin e Tomas Mikolov. *Enriching Word Vectors with Subword Information*. 2016. doi: 10.48550/ARXIV.1607.04606. url: https://arxiv.org/abs/1607.04606 (citado na pg. 15).

[Brown *et al.* 2020] Tom B. Brown *et al. Language Models are Few-Shot Learners*. 2020. doi: 10.48550/ARXIV.2005.14165. url: https://arxiv.org/abs/2005.14165 (citado na pg. 15).

[Cho *et al.* 2014] Kyunghyun Cho *et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. doi: 10.48550/ARXIV.1406.1078. url: https://arxiv.org/abs/1406.1078 (citado na pg. 17).

[Clark *et al.* 2020] Kevin Clark, Minh-Thang Luong, Quoc V. Le e Christopher D. Manning. "ELECTRA: pre-training text encoders as discriminators rather than generators". Em: *CoRR* abs/2003.10555 (2020). arXiv: 2003.10555. url: https://arxiv.org/abs/2003.10555 (citado na pg. 25).

[Devlin *et al.* 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee e Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. doi: 10.48550/ARXIV.1810.04805. url: https://arxiv.org/abs/1810.04805 (citado nas pgs. 15, 20, 23).

[Dicio 2022a] Dicio. *Ironia - Dicio, Dicionário Online de Português*. url: https://www.dicio.com.br/ironia/ (acesso em 19/09/2022) (citado na pg. 3).

[Dicio 2022b] Dicio. *Sarcasmo - Dicio, Dicionário Online de Português*. url: https://www.dicio.com.br/sarcasmo/ (acesso em 19/09/2022) (citado na pg. 3).

[Galassi *et al.* 2021] Andrea Galassi, Marco Lippi e Paolo Torroni. "Attention in natural language processing". Em: *IEEE Transactions on Neural Networks and Learning Systems* 32.10 (out. de 2021), pgs. 4291–4308. doi: 10.1109/tnnls.2020.3019893. url: https://doi.org/10.1109%2Ftnnls.2020.3019893 (citado na pg. 18).

[A. Ghosh e Veale 2016] Aniruddha Ghosh e Tony Veale. "Fracking sarcasm using neural network". Em: *Proceedings of the 7th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. San Diego, California: Association for Computational Linguistics, jun. de 2016, pgs. 161–169. doi: 10.18653/v1/W16-0425. url: https://aclanthology.org/W16-0425 (citado na pg. 13).

[D. Ghosh *et al.* 2018] Debanjan Ghosh, Alexander R. Fabbri e Smaranda Muresan. "Sarcasm Analysis Using Conversation Context". Em: vol. 44. 4. Dez. de 2018, pgs. 755–792. doi: 10.1162/coli_a_00336. eprint: https://direct.mit.edu/coli/article-pdf/44/4/755/1809923/coli_a_00336.pdf. url: https://doi.org/10.1162/coli%5C_a%5C_00336 (citado na pg. 15).

[Hazarika *et al.* 2018] Devamanyu Hazarika *et al.* "CASCADE: contextual sarcasm detection in online discussion forums". Em: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, ago. de 2018, pgs. 1837–1848. url: https://aclanthology.org/C18-1156 (citado na pg. 14).

[He, Gao *et al.* 2021] Pengcheng He, Jianfeng Gao e Weizhu Chen. "Debertav3: improving deberta using electra-style pre-training with gradient-disentangled embedding sharing". Em: *CoRR* abs/2111.09543 (2021). arXiv: 2111.09543. url: https://arxiv.org/abs/2111.09543 (citado na pg. 25).

[He, X. Liu *et al.* 2020] Pengcheng He, Xiaodong Liu, Jianfeng Gao e Weizhu Chen. *DeBERTa: Decoding-enhanced BERT with Disentangled Attention*. 2020. doi: 10.48550/ARXIV.2006.03654. url: https://arxiv.org/abs/2006.03654 (citado nas pgs. 15, 16, 23, 24).

[Hochreiter e Schmidhuber 1997] Sepp Hochreiter e Jürgen Schmidhuber. "Long short-term memory". Em: *Neural computation* 9 (dez. de 1997), pgs. 1735–80. doi: 10.1162/neco.1997.9.8.1735 (citado na pg. 16).

[Joshi, Bhattacharyya *et al.* 2017] Aditya Joshi, Pushpak Bhattacharyya e Mark J. Carman. "Automatic sarcasm detection: a survey". Em: *ACM Comput. Surv.* 50.5 (set. de 2017). issn: 0360-0300. doi: 10.1145/3124420. url: https://doi.org/10.1145/3124420 (citado nas pgs. 12, 16).

[Joshi, Sharma *et al.* 2015] Aditya Joshi, Vinita Sharma e Pushpak Bhattacharyya. "Harnessing context incongruity for sarcasm detection". Em: jul. de 2015. doi: 10.3115/v1/P15-2124 (citado na pg. 11).

[Joshi, Tripathi *et al.* 2016] Aditya Joshi, Vaibhav Tripathi, Pushpak Bhattacharyya e Mark J. Carman. "Harnessing sequence labeling for sarcasm detection in dialogue from TV series 'Friends'". Em: *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*. Berlin, Germany: Association for Computational Linguistics, ago. de 2016, pgs. 146–155. doi: 10.18653/v1/K16-1015. url: https://aclanthology.org/K16-1015 (citado na pg. 12).

[Khodak *et al.* 2017] Mikhail Khodak, Nikunj Saunshi e Kiran Vodrahalli. "A large self-annotated corpus for sarcasm". Em: *CoRR* abs/1704.05579 (2017). arXiv: 1704.05579. url: http://arxiv.org/abs/1704.05579 (citado na pg. 26).

[Kolchinski e Potts 2018] Y. Alex Kolchinski e Christopher Potts. "Representing social media users for sarcasm detection". Em: *CoRR* abs/1808.08470 (2018). arXiv: 1808.08470. url: http://arxiv.org/abs/1808.08470 (citado na pg. 14).

[Kumar Jena *et al.* 2020] Amit Kumar Jena, Aman Sinha e Rohit Agarwal. "C-net: contextual network for sarcasm detection". Em: *Proceedings of the Second Workshop on Figurative Language Processing*. Online: Association for Computational Linguistics, jul. de 2020, pgs. 61–66. doi: 10.18653/v1/2020.figlang-1.8. url: https://aclanthology.org/2020.figlang-1.8 (citado na pg. 15).

[Liebrecht *et al.* 2013] Christine Liebrecht, Florian Kunneman e Antal van den Bosch. "The perfect solution for detecting sarcasm in tweets #not". Em: *WASSA@NAACL-HLT*. 2013 (citado nas pgs. 10, 12).

[P. Liu *et al.* 2014] Peng Liu *et al.* "Sarcasm detection in social media based on imbalanced classification". Em: jun. de 2014, pgs. 459–471. isbn: 978-3-319-08009-3. doi: 10.1007/978-3-319-08010-9_49 (citado na pg. 13).

[Y. Liu *et al.* 2019] Yinhan Liu *et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. doi: 10.48550/ARXIV.1907.11692. url: https://arxiv.org/abs/1907.11692 (citado nas pgs. 15, 23).

[Maynard e Greenwood 2014] Diana Maynard e Mark Greenwood. "Who cares about sarcastic tweets? investigating the impact of sarcasm on sentiment analysis." Em: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. Reykjavik, Iceland: European Language Resources Association (ELRA), mai. de 2014, pgs. 4238–4243. url: http://www.lrec-conf.org/proceedings/lrec2014/pdf/67_Paper.pdf (citado na pg. 8).

[Mikolov *et al.* 2013] Tomas Mikolov, Kai Chen, Greg Corrado e Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. doi: 10.48550/ARXIV.1301.3781. url: https://arxiv.org/abs/1301.3781 (citado na pg. 15).

[Mishra *et al.* 2016] Abhijit Mishra, Diptesh Kanojia, Seema Nagar, Kuntal Dey e Pushpak Bhattacharyya. "Harnessing cognitive features for sarcasm detection". Em: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, ago. de 2016, pgs. 1095–1104. doi: 10.18653/v1/P16-1104. url: https://aclanthology.org/P16-1104 (citado na pg. 11).

[Pennington *et al.* 2014] Jeffrey Pennington, Richard Socher e Christopher Manning. "GloVe: global vectors for word representation". Em: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, out. de 2014, pgs. 1532–1543. doi: 10.3115/v1/D14-1162. url: https://aclanthology.org/D14-1162 (citado na pg. 15).

[Peters *et al.* 2018] Matthew E. Peters *et al. Deep contextualized word representations*. 2018. doi: 10.48550/ARXIV.1802.05365. url: https://arxiv.org/abs/1802.05365 (citado na pg. 15).

[Potamias *et al.* 2020] Rolandos Alexandros Potamias, Georgios Siolas e Andreas - . Georgios Stafylopatis. "A transformer-based approach to irony and sarcasm detection". Em: *Neural Computing and Applications* 32.23 (dez. de 2020), pgs. 17309–17320. issn: 1433-3058. doi: 10.1007/s00521-020-05102-3. url: https://doi.org/10.1007/s00521-020-05102-3 (citado na pg. 16).

[Reddit 2022] Reddit. *What does /s mean?* url: https://www.reddit.com/r/OutOfTheLoop/comments/1zo2l4/comment/cfvupgz/?utm_source=share&utm_medium=web2x&context=3 (acesso em 08/11/2022) (citado na pg. 26).

[Reyes, Rosso e Buscaldi 2012] Antonio Reyes, Paolo Rosso e Davide Buscaldi. "From humor recognition to irony detection: the figurative language of social media". Em: *Data & Knowledge Engineering* 74 (2012). Applications of Natural Language to Information Systems, pgs. 1–12. issn: 0169-023X. doi: https://doi.org/10.1016/j.datak.2012.02.005. url: https://www.sciencedirect.com/science/article/pii/S0169023X12000237 (citado na pg. 10).

[Reyes, Rosso e Veale 2013] Antonio Reyes, Paolo Rosso e Tony Veale. "A multidimensional approach for detecting irony in twitter". Em: *Language Resources and Evaluation* 47.1 (mar. de 2013), pgs. 239–268. issn: 1574-0218. doi: 10.1007/s10579-012-9196-x. url: https://doi.org/10.1007/s10579-012-9196-x (citado nas pgs. 10, 12).

[Riloff *et al.* 2013] Ellen Riloff *et al.* "Sarcasm as contrast between a positive sentiment and negative situation". Em: *EMNLP*. 2013 (citado na pg. 12).

[Van Hee *et al.* 2018] Cynthia Van Hee, Els Lefever e Véronique Hoste. "SemEval-2018 task 3: irony detection in English tweets". Em: *Proceedings of the 12th International Workshop on Semantic Evaluation*. New Orleans, Louisiana: Association for Computational Linguistics, jun. de 2018, pgs. 39–50. doi: 10.18653/v1/S18-1005. url: https://aclanthology.org/S18-1005 (citado na pg. 13).

[Vaswani *et al.* 2017] Ashish Vaswani *et al. Attention Is All You Need*. 2017. doi: 10.48550/ARXIV.1706.03762. url: https://arxiv.org/abs/1706.03762 (citado nas pgs. 19, 20, 23, 24).

[Veale e Hao 2010] Tony Veale e Yanfen Hao. "Detecting ironic intent in creative comparisons". Em: *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*. NLD: IOS Press, 2010, pgs. 765–770. isbn: 9781607506058 (citado na pg. 8).

[Wallace *et al.* 2014] Byron C. Wallace, Do Kook Choe, Laura Kertz e Eugene Charniak. "Humans require context to infer ironic intent (so computers probably do, too)". Em: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, jun. de 2014, pgs. 512–516. doi: 10.3115/v1/P14-2084. url: https://aclanthology.org/P14-2084 (citado nas pgs. 3, 14).

[Wang *et al.* 2015] Zelin Wang, Zhijian Wu, Ruimin Wang e Yafeng Ren. "Twitter sarcasm detection exploiting a context-based model". Em: *Web Information Systems Engineering – WISE 2015*. Ed. por Jianyong Wang *et al.* Cham: Springer International Publishing, 2015, pgs. 77–91. isbn: 978-3-319-26190-4 (citado na pg. 14).

[Yaghoobian *et al.* 2021] Hamed Yaghoobian, Hamid R. Arabnia e Khaled Rasheed. "Sarcasm detection: a comparative study". Em: (2021). doi: 10.48550/ARXIV.2107.02276. url: https://arxiv.org/abs/2107.02276 (citado nas pgs. 12, 21).

[Yang *et al.* 2019] Zhilin Yang *et al. XLNet: Generalized Autoregressive Pretraining for Language Understanding*. 2019. doi: 10.48550/ARXIV.1906.08237. url: https://arxiv.org/abs/1906.08237 (citado na pg. 15).