Universidade Federal de Pernambuco
Centro de Informática

Graduação em Ciência da Computação

Deep Learning na Detecção de Posicionamento em Notícias *Online*

João Pedro de Carvalho Magalhães

Trabalho de Graduação

Recife
21 de Junho de 2017

Universidade Federal de Pernambuco
Centro de Informática

João Pedro de Carvalho Magalhães

Deep Learning na Detecção de Posicionamento em Notícias *Online*

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Prof. Tsang Ing Ren*

Recife
21 de Junho de 2017

Agradecimentos

Muito obrigado

Ao professor Tsang Ing Ren, por me orientar neste trabalho e também durante a graduação, ajudando a corrigir falhas, ensinando o que é ciência, a quando ser pragmático e quando buscar ir além.

Aos demais professores do Centro de Informática, em especial Pedro Manhães, pelos desafios em suas aulas, que muito me motivaram para o prosseguimento do curso; Liliane Salgado, pela chefia da Maratona de Programação, projeto tão importante do qual fiz parte por um ano; Ruy de Queiroz, por me dar a oportunidade de ser monitor na disciplina de Lógica para Computação; e Luciano Barbosa, por ensinar brilhantemente em uma área de grande interesse para mim e estar sempre aberto para conversas.

À Coordenadoria de Aperfeiçoamento de Pessoal de Nível Superior, pela aprovação e custeamento da minha bolsa nos programas Jovens Talentos para Ciência e Ciências Sem Fronteiras.

À *Georgia Institute of Technology*, *The Ohio State University*, e todos seus funcionários, por aceitarem me receber durante o intercâmbio e me presentearem com estadias de grande crescimento acadêmico, profissional e pessoal. Em especial ao professor Alexander Lerch, que amigavelmente orientou meu trabalho em seu laboratório de pesquisa.

Aos que tornaram minha adaptação mais fácil e meu dia-a-dia melhores nos Estados Unidos, como André, Anna, Bruno, Carol, Mari, Randall, Rithesh, Yang e tantos outros.

Aos amigos feitos durante o curso, em especial Bertha, Duhan, Eduardo, Guilherme, Larissa, Leonardo, Lucas Lima, Lucas Netto, Maria Gabriela, Marina, Mateus, Rafael Acevedo, Rafael Francisco, Raíssa e Vinícius, por me manterem ciente do meu potencial e com os quais sempre pude contar, dentro e fora da universidade.

Aos meus amigos de vida, Aline, Allana, Luara, Pedro e Petty, por estarem presentes em momentos de diversão ou de necessidade, sempre que possível.

À minha família, especialmente à minha mãe Gerlane, ao meu pai João Mauricio, e ao meu irmão João Guilherme. Por tornarem meus dias mais alegres com tanto carinho, meu futuro maior com muito incentivo, e uma pessoa melhor pelos exemplos que são.

No matter what anybody tells you, words and ideas can change the world.
—JOHN KEATING (do filme "Sociedade dos Poetas Mortos")

Resumo

O compartilhamento de notícias falsas vem aumentando demasiadamente na *web*. Isso é alarmante, pois histórias ou alegações falsas causam um forte impacto negativo na sociedade se entendidas como verdadeiras. Para rapidamente evitar a circulação de tais notícias, é necessário criar um sistema automatizado capaz de auxiliar humanos a detectar alegações falsas. Este trabalho propõe abordagens *deep learning* para detecção de posicionamento em notícias *online*. É discutido como o posicionamento pode ajudar a detecção de notícias falsas e como as abordagens criadas funcionam. É mostrado que uma delas supera as performances do *baseline*.

Palavras-chave: notícias online, notícias falsas, detecção de posicionamento, deep learning, processamento de linguagem natural, aprendizagem de máquina

Abstract

Fake news sharing has overly increased in the web. This is dangerous because false stories or statements cause a strong negative impact to society when believed to be true. In order to quickly avoid circulation of such news, an automated system capable of helping humans detect false claims is needed. This work proposes deep learning approaches for stance detection in online news. It is discussed how stances can help false news detection, and how such approaches operates. It is showed that one of these approaches outperforms the baseline.

Keywords: online news, fake news, stance detection, deep learning, natural language processing, machine learning

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

A *World Wide Web* é uma enorme fonte de dados. A cada segundo, milhares de novos arquivos e informações são colocadas *online* por pessoas do mundo inteiro. E essa produção vem crescendo ainda mais com a popularização de *blogs* e redes sociais, pois agora cada indivíduo possui um espaço próprio e interconectado para elaborar e compartilhar seu conteúdo ou o de outras pessoas e veículos de comunicação.

As pessoas podem fazer isso de qualquer lugar se em posse de um *smartphone* e acesso à internet móvel. Isso torna qualquer um nessas condições um divulgador instantâneo e global de informações, sejam elas factuais ou opinativas. Esse poder, apesar de impressionante e animador, também pode ser perigoso quando os usuários não conseguem checar a veracidade das informações [1], tarefa complicada até para alguns experts [2].

Tal incapacidade, em conjunto com a acelerada propagação de notícias ou informações falsas [3], pode influenciar a opinião pública e até o mercado financeiro, afetando negativamente a sociedade [4, 5, 6]. Nesta conjuntura, grandes empresas de tecnologia estão se juntando para estudar o problema [7], e a detecção de posicionamento surgiu como um passo importante para possibilitar o avanço na detecção automática de notícias falsas [8].

A detecção de posicionamento é a tarefa de identificar o tipo de relação entre dois discursos. Um discurso pode ser um tópico, um argumento ou uma afirmação, e essa relação pode ser de concordância, discordância, ou neutralidade – quando um discurso não tem uma posição polarizada acerca de outro discurso. Para ajudar na detecção de notícias de conteúdo falso, a detecção de posicionamento seria uma de várias partes de um futuro sistema automatizado por Inteligência Artificial e voltado a auxiliar os profissionais que checam a veracidade de fatos em notícias.

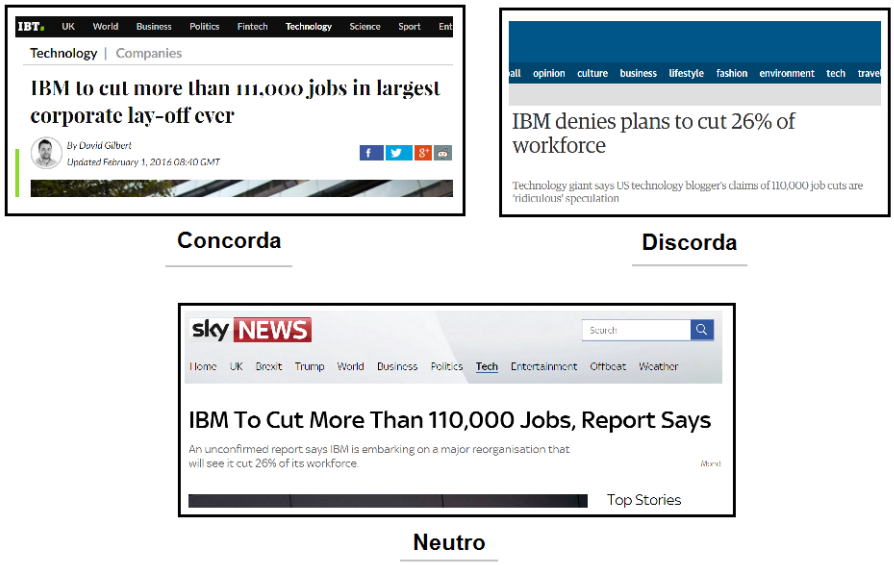
Nesse contexto, o posicionamento seria definido entre alegações e manchetes ou corpos de notícias, de forma que se poderia comparar de maneira concisa como fontes diferentes estão noticiando um mesmo tópico, assim como pode ser visto na Figura 1.1. A tarefa também pode ser estendida para detectar se os itens comparados estão tematicamente relacionados ou não [8], automatizando e enriquecendo ainda mais o processo.

A sua automação pode ser alcançada através de abordagens computacionais. Uma que vem tendo sucesso em tarefas de detecção, passando a ser bastante usada em Processamento de Linguagem Natural, é chamada de *deep learning*. Esse conjunto de técnicas, que exige um grande poder de processamento e vasta quantidade de dados, pode eficientemente aumentar o aprendizado sobre uma tarefa [9].

Uma acurada detecção automática de posicionamento em notícias pode fomentar os trabalhos que explorem especificamente a correlação entre posicionamento e conteúdo falso, como a relação entre: posionamento em uma determinada notícia e a presença de afirmações fal-

sas nela; credibilidade de uma fonte e distribuição de posicionamento nas diversas notícias dessa fonte; ou ainda, posicionamentos em notícias de mesmo tópico mas distintas fontes e a presença de alegações falsas em cada. Também é possível que se estude uma maneira de quantificar a relação de posicionamento predita para que se possa determinar o quão enviesadas ou sensacionalistas são as notícias divulgadas por uma fonte.

Figura 1.1 Exemplos de posicionamento entre a alegação "*IBM to cut more than 111,000 jobs*"e manchetes de notícias extraídas do IBTimes [10], The Guardian [11] e Sky News [12] (da esquerda para a direita, de cima para baixo).



Fonte: Adaptada pelo autor

1.1 Objetivos

Este trabalho foca-se na construção de um classificador com abordagem *deep learning* para a determinação de posicionamento entre manchete e corpo de notícias, considerando a extensão proposta em que os temas podem não estar relacionados [8]. Adicionando outras *features* complementares e utilizando um conjunto de dados pronto [13], fazem-se experimentos para avaliar se sua performance é melhor que o método *baseline* [14] e boa o suficiente para que seja usado em um sistema real de auxílio à detecção de notícias falsas.

1.2 Estrutura do Trabalho

O resto deste trabalho foi estruturado da seguinte maneira: o capítulo 2 introduz áreas, conceitos e trabalhos anteriores relacionadas ao problema e à solução proposta. No capítulo 3, detalha-se o funcionamento geral do classificador *deep learning* proposto, desde o pré-processamento dos textos ao resultado final, além da abordagem *baseline* que será utilizada para avaliação e de *features* adicionais. No capítulo 4, é apresentada a coleção de dados, a metodologia criada para os experimentos, e uma análise dos resultados obtidos. Por último, é feita a conclusão do trabalho no capítulo 5, finalizando com ideias para trabalhos futuros.

CAPÍTULO 2

Fundamentos

Este capítulo descreve as áreas e os conceitos relacionados ao problema e às abordagens relatadas. Trabalhos anteriores correlacionados também são apresentados.

2.1 Aprendizagem de Máquina

Um dos campos fundamentais para a automação de tarefas e habilidades humanas, a Aprendizagem de Máquina envolve o estudo de algoritmos capazes de aprender informações sobre conjuntos de dados ou experiências passadas para fazer previsões corretas sobre um problema ou ação. Existem várias políticas de aprendizado, sendo as mais exploradas a supervisionada, não-supervisionada, e por reforço. Para cada política, existem diversos algoritmos à disposição.

A boa performance de predição depende de alguns fatores, como a escolha do algoritmo. Caso o algoritmo escolhido extraia informações inadequadas ou muito simples para o problema em questão, é possível que ocorra *underfitting*, resultando em uma baixa performance para quaisquer dados. Se o algoritmo extrair informações muito complexas, ele pode sofrer *overfitting*, modelando-se demais sobre os dados usados no aprendizado e gerando resultados ruins sobre outros conjuntos de dados.

Um dos objetivos mais importantes dessa área, portanto, é a geração de soluções generalizáveis, que não apenas façam boas predições para um conjunto de dados específico, mas para quaisquer disponíveis. Por esse motivo, a Aprendizagem de Máquina também envolve o estudo de técnicas que regularizem a performance dessas soluções [15].

2.1.1 Aprendizagem Supervisionada

Dado um problema, deseja-se inferir uma função que mapeie entradas $\mathtt{x} \in \mathtt{X}$ para saídas $\mathtt{y} \in \mathtt{Y}$. Diz-se que $\mathtt{X}$ é o espaço de informações (*features*) das quais o algoritmo aprenderá padrões, e $\mathtt{Y}$ é o espaço de valores possíveis de serem preditos. Quando $\mathtt{Y}$ contém valores contínuos como números reais, o problema é chamado de regressão; e quando contém valores discretos, o problema é chamado de classificação, em que cada $\mathtt{y}$ é uma classe possível [15].

Algoritmos dessa classe necessitam de conjuntos de dados que contenham informações da entrada $\mathtt{x}$ e do verdadeiro resultado $\mathtt{y}$ para todas as instâncias usadas no aprendizado. Eles são ditos supervisionados por funcionarem através de correção de erros, como um professor que ensina seus alunos. Então, a cada exemplo novo aprendido, compara-se o valor predito pelo algoritmo à sua verdadeira previsão $\mathtt{y}$ para testar o nível de erro e se é possível minimizá-lo.

2.1.2 *Gradient Boosted Trees*

Boosting é um meta-algoritmo usado no contexto de aprendizagem supervisionada para transformar um conjunto de classificadores fracos em um único classificador forte através da minimização iterativa de uma função de custo [16]. A ideia é que a cada iteração (ou estágio de *boosting*) se construa um classificador fraco – ou mais de um para problemas multiclasse – que aprenda padrões das *features* de entrada para classificar pares `(x, y)` dos dados de treinamento e calcular o erro sobre eles com a função de custo. Antes do fim de um estágio, pares incorretamente classificados recebem pesos maiores que os corretamente classificados para que tenham maior atenção no próximo estágio. Esse processo é feito por até `E` estágios.

Existem diversas versões de *boosting*. Elas variam a depender de como esses pesos são alterados, como o resultado de cada estágio é combinado, pela escolha do classificador fraco, ou pela escolha da função de custo. Uma dessas é o *gradient boosting*, que aumenta a abundância de funções de custos possíveis de serem escolhidas ao generalizar o *boosting* a partir de uma abstração que o relaciona com o *gradient descent* [17]. Essa variação é muito empregada com árvores de regressão como o classificador fraco, resultando no classificador forte nomeado de *Gradient Boosted Trees*.

Àrvores de regressão funcionam similarmente à árvores de decisão. Cada nó divide amostras dos dados em subamostras e distribui entre nós filhos sucessivamente. Essa divisão é feita com base em alguma *feature*, que é qualificada por algum critério como a geradora da melhor divisão para tal amostra nesse dado nó. Existem outros critérios associados ao nível ou à árvore completa, mas eles não costumam ser usados no contexto de *Gradient Boosted Trees* por aumentarem a complexidade de geração das árvores.

Esse tipo de árvore é interessante porque cria uma função que mapeia instâncias para folhas da árvore, associadas a escores de predição. Então, para o *boosting*, as funções criadas até o `i`-ésimo estágio de aprendizado são combinadas para definir a classe predita de instâncias testadas nesse estágio `i`. É possível aplicar um peso a essas funções, funcionando como uma taxa de aprendizagem.

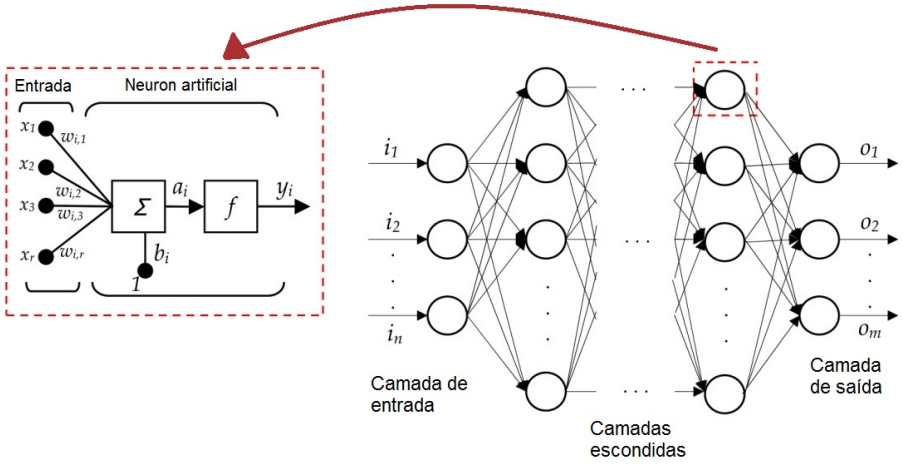
2.1.3 Redes Neurais Convolucionais

Redes neurais artificiais são algoritmos clássicos de Aprendizagem de Máquina inspirados no funcionamento biológico do cérebro humano. Elas são comparáveis a grafos dispostos em camadas, onde os nós são neurons e as arestas são sinapses. Cada camada é composta por múltiplos neurons que se ligam através dessas sinapses a neurons de outras camadas.

Camadas intermediárias são chamadas de camadas escondidas e toda rede tem uma camada de entrada e uma de saída. A camada de entrada é formada por *features* e a de saída é por onde saem os resultados. Um neuron processa os sinais que chegam pelas sinapses combinando-os com pesos vinculados à essas sinapses. Essa combinação pode contabilizar um valor de viés.

A informação processada passa por uma função de ativação, que determina se o neuron está ativado. Isto é, se a informação que ele processou vai ser passada adiante para outros neurons. A Figura 2.1 demonstra visualmente uma rede dessas. Os pesos das sinapses são atualizados pelo aprendizado, como no algoritmo de *backpropagation* por exemplo, em que uma função de custo calcula o erro entre a saída da rede e a saída verdadeira e propaga esse erro de volta.

Figura 2.1 Exemplo de rede neural artificial.

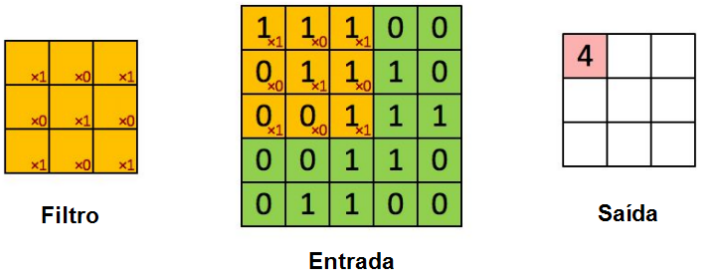


Fonte: https://www.analyticsvidhya.com/blog/2016/03/introduction-deep-learning-fundamentals-neural-networks/

A aplicação de convoluções e *pooling* estão entre as técnicas de *deep learning* com ótimos resultados em Visão Computacional e Processamento de Linguagem Natural, e que possibilitaram a aparição de novas redes. Quando uma rede possui ao menos uma camada de convolução, ela é chamada rede neural convolucional.

Uma convolução é a aplicação de um filtro (ou *kernel*) a várias regiões de uma matriz de entrada, produzindo um mapa de *features*. Como observado na Figura 2.2, o filtro é uma matriz de 3x3 com 1 ou 0 e cada posição é multiplicada por posições na matriz de entrada, depois somadas, resultando no valor 4.

Figura 2.2 Exemplo de convolução. O filtro é aplicado a uma submatriz da entrada produzindo uma *feature* na saída.



Fonte: http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/

Um *kernel* pode ter diferentes tamanhos, a depender de como se quer gerar o mapa de *features*, sendo habitual o emprego de múltiplos *kernels* para que se produza mais de um mapa. Uma propriedade importante de uma convolução é seu *stride*. Ele determina os intervalos entre regiões de aplicação de um filtro. Um valor de 1 de *stride* aplicar filtros em regiões vizinhas, com sobreposição de subregiões, por exemplo.

O *pooling* é uma operação de subamostragem em que se extrai um único valor para representar uma região de valores. Essa operação pode ser de máximo (*maximum pooling*), mínimo (*minimum pooling*) ou média (*average pooling*), quando respectivamente se extrai o maior, menor ou a média dos valores dessa região. Assim como na convolução, é possível escolher o tamanho da região da matriz de entrada da qual se quer fazer *pooling* e seu *stride*. É comum o emprego de *pooling* após camadas de convolução [18].

2.2 Processamento de Linguagem Natural

Área da Ciência da Computação que está na interseção entre a Inteligência Artificial e a Linguística, o Processamento de Linguagem Natural tem como objetivo básico estudar métodos que façam com que o computador saiba interpretar e reproduzir a linguagem natural humana, seja ela por voz ou texto [19].

As tarefas mais complexas estudadas – que incluem a tradução entre idiomas, extração de relações entre palavras, reconhecimento de entidades de nome próprio e a sumarização, por exemplo – dependem da automatização de tarefas mais básicas ligadas à análise sintaxe, semântica e ao reconhecimento de palavras.

Neste sentido, métodos como a tokenização e a normalização visam facilitar a identificação e a uniformização das palavras em textos, enquanto *word embeddings* discernem significados.

2.2.1 Tokenização

A tokenização é o processo de divisão de um texto em partes, chamadas de *tokens*. Essa divisão é baseada em uma ou mais regras, que são escolhidas a partir do objetivo final da transformação e do idioma em que o texto está escrito, de modo que um *token* pode ser representado por uma única palavra, palavra seguida de sinal de pontuação, sequência de palavras, entre outras formas, como observado na Figura 2.3.

Geralmente, o objetivo da tokenização é que cada *token* represente apenas uma palavra. Por isso, uma regra muito comum por conseguir cobrir decentemente vários casos em diversos idiomas consiste em fazer a divisão na posição de ocorrência de caracteres não-alfanuméricos, como espaços e sinais de pontuação, podendo ou não remover estes [20]. Entretanto, frequentemente regras mais complexas são necessárias para atingir uma precisão maior na segmentação e lidar com casos de palavras compostas por hífen ou apóstrofo. Isso pode ser alcançado com o uso de expressões regulares, por exemplo.

Nos idiomas que possuem uma supercomposição de palavras ou cujos símbolos representam sílabas e não existem espaços entre palavras, respectivamente casos do alemão e do chinês, são necessárias regras que encontrem o ponto de divisão das palavras de maneira inteligente, utilizando dicionários e *backtracking*.

Figura 2.3 Exemplos de tokenização da sentença "Ice-cream? It's good!". À direita das setas, cada *token* está destacado em cinza.

Método		*Tokens*
Espaços	→	Ice-cream? It's good!
Espaços e sinais de pontuação	→	Ice - cream ? It ' s good !
Espaços e sinais de pontuação sem letras nos dois caracteres vizinhos	→	Ice-cream ? It's good !

Fonte: Elaborada pelo autor.

2.2.2 *Case Folding*

Case folding é a forma mais simples de se realizar normalização em um texto, e trata-se da conversão de letras maiúsculas em minúsculas de acordo com alguma estratégia. A mais adotada é fazer a conversão para todos os caracteres maiúsculo encontrados, não discernindo palavras capitalizadas por serem substantivos próprios das demais.

Duas outras estratégias podem ser adotadas em seu lugar para tentar contornar isso, inclusive em conjunto. Uma é a conversão para minúsculo de sentenças que estão completamente capitalizadas, e a outra é a conversão apenas da primeira palavra que inicia um parágrafo ou procede um ponto final [20], como mostra a Figura 2.4.

Figura 2.4 Exemplos de *case-folding* da sentença "The headquarters of the UN is in New York".

Método		Sentença resultante
Todos os caracteres	→	the headquarters of the un is in new york
Primeira palavra da sentença	→	the headquarters of the UN is in New York

Fonte: Elaborada pelo autor.

2.2.3 Lematização

A lematização é uma técnica de normalização que visa transformar uma palavra em seu lema. Também nomeado de forma canônica, o lema é o formato que uma palavra toma quando não está flexionada de nenhuma maneira, seja por gênero, grau, número, pessoa, tempo ou modo.

Como exposto na Figura 2.5, esse formato básico é o mesmo que se encontra em dicionários reais [20], em que verbos estão no infinitivo impessoal e substantivos no singular. Então, por estar diretamente ligada à flexão das palavras e às convenções de um dicionário, a lematização

é dependente do idioma e costuma ser implementada com representações computacionais de dicionários e thesauri.

Figura 2.5 Exemplos de palavras inglesas lematizadas.

Palavras originais		Lema
am, is, are	→	be
car, cars, car's, cars'	→	car
good, better, best	→	good

Fonte: Elaborada pelo autor.

2.2.4 *Word Embeddings*

Para que textos sejam classificados por algoritmos de Aprendizagem de Máquina, eles precisam ser convertidos em números reais. Em Processamento de Linguagem Natural clássico, um texto é transformado em *Bag-of-Words*, um vetor de números inteiros que representam as contagens das palavras vistas ou não nesse texto, contanto que existam no vocabulário do conjunto de todos os textos analisados [21].

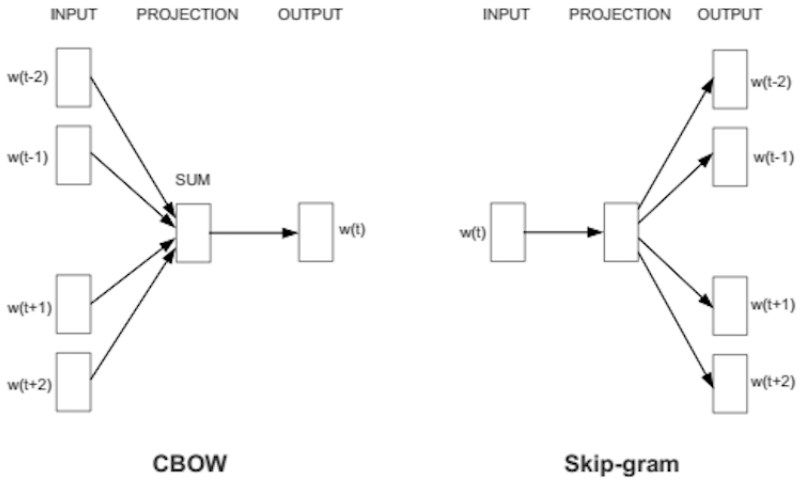
Todavia, essa estrutura é muito esparsa, pois cada palavra não vista tem sua contagem mapeada para zero, e o número de palavras contidas no vocabulário e que não foram vistas em um dado texto é enorme. Precisava-se encontrar uma outra representação de texto com menos perda de informações e que levasse a resultados mais acurados. Eis que em 2003 [22], ela surgiu e recentemente vem ganhando cada vez mais atenção.

Esta representação é obtida através de *word embeddings*, que tecnicamente são funções parametrizadas desenvolvidas para mapear uma palavra em um denso vetor espacial de n dimensões. Consequentemente, um texto deixa de ser um único vetor numérico e se transforma em uma sequência de vetores.

Os dois modelos mais comuns para criação desses vetores são pesos resultantes de redes neurais treinadas sobre o contexto em que as palavras estão inseridas, e podem ser vistos na Figura 2.6. Os pesos podem ser inicializados aleatoriamente ou recuperados de redes pré-treinadas, sendo ajustados pelo algoritmo de *backpropagation*. O *Continuous Bag of Words* (CBoW) treina uma rede neural para gerar a probabilidade de uma dada palavra ocorrer com base nas ocorrências de palavras vizinhas a ela nos textos do corpus. O modelo *Skip-Gram* faz o inverso, gerando as probabilidades de palavras serem vizinhas de uma palavra central [23].

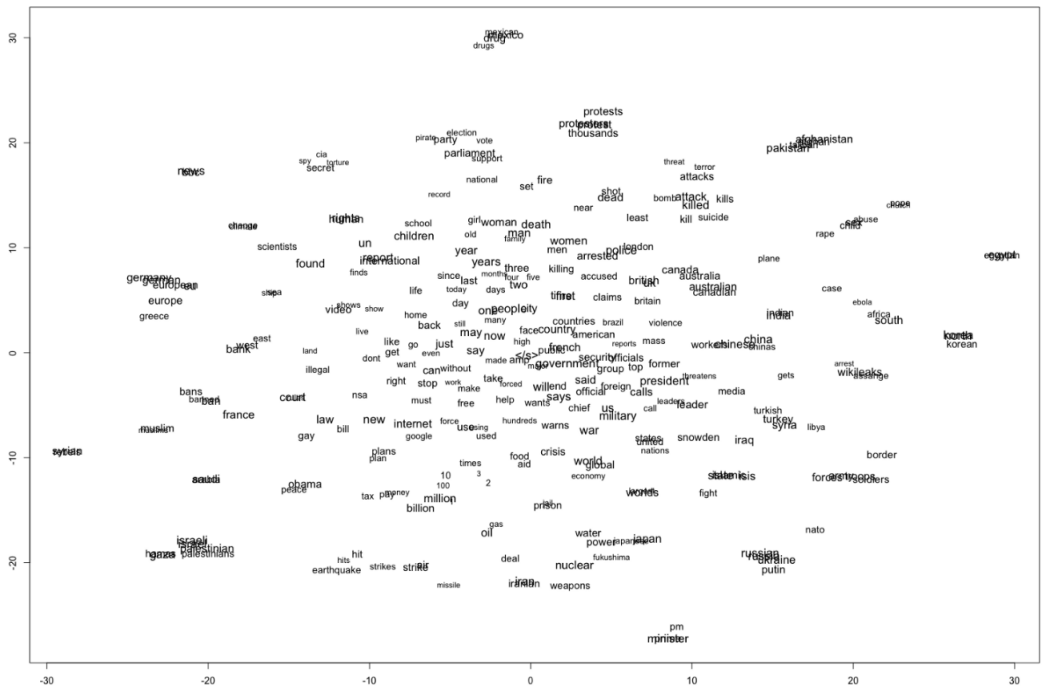
Como resultado, cada dimensão de um vetor é um número real que se refere implicitamente a uma característica sintática ou semântica, e palavras com contexto similar estão próximas no espaço, como exemplificado na Figura 2.7. Logo, essa representação permite fazer operações baseadas na distância entre as palavras no espaço e gerar resultados como similaridade entre duas palavras, palavra análoga resultante de uma soma ou subtração entre palavras, etc. Todas essas características tornam essa representação bem mais eficiente e expressiva.

Figura 2.6 Arquiteturas CBoW e *Skip-Gram*.



Fonte: https://deeplearning4j.org/word2vec

Figura 2.7 Exemplo de representação bidimensional da distância entre vetores de palavras aprendidos em textos de notícias.



Fonte: https://hackernoon.com/predict-stock-market-with-daily-top-news-8c8db25bef8d

2.3 Trabalhos Relacionados

Durante os anos próximos à virada da século XX, houve um bom desenvolvimento nas áreas de Mineração de Opinião e Análise de Sentimento com uma sequência de estudos que exploravam a subjetividade [24, 25, 26], polaridade [27, 28, 29] e perspectiva [30] em sentenças, documentos ou na linguagem natural de modo geral. Seguindo essa sucessão, surgiu o primeiro trabalho de detecção automática de posicionamento como uma relação entre perspectivas.

De 2006, este trabalho introduziu a definição atual da tarefa com classes básicas de apoio (concordância) e oposição (discordância). Utilizando transcrições de debates com múltiplos locutores sobre propostas de lei ocorridos no congresso americano, são discutidos métodos que separam discursos em segmentos, extraem relações entre eles e usam essas informações para a classificação com uma *Support Vector Machine* [31].

A maior parte dos trabalhos seguintes se inspiraram no crescimento da *web* e se focaram em debates realizados em fóruns *online* sobre ideologias [32, 33, 34, 35, 36, 37], produtos [38], e outros temas [39, 40]. Todos estes trabalhos fazem a classificação em conjuntos de dados com tópicos bem definidos. Isto é, não se tem no mesmo conjunto debates entre marcas de celulares e marcas de carros, apenas entre marcas de um desses produtos, por exemplo.

Nesses estudos ocorre a exploração tanto da detecção direta do posicionamento quanto do posicionamento como derivado de pontos de vistas individuais, alguns utilizando informações prévias sobre o alvo a que se refere uma opinião e outros não. Entre as abordagens, domina o uso de *Support Vector Machine*, *Naive Bayes*, Programação Linear Inteira e grafos. Nesta última, os nós são discursos ou usuários e as arestas indicam a relação entre esses nós.

Há estudos que associam a perspectiva individual a nível de discurso e outros a nível de usuário. Por isso, existe uma diversidade na análise de *features*, com pesquisas se concentrando nas línguisticas e relações entre discursos ou usuários. Entre esses estudos, nota-se o início da popularização da palavra *stance*[1] para nomeação da tarefa [34, 38], e a apresentação do posicionamento neutro [33].

A tarefa de detecção de posicionamento já foi aplicada a redações de alunos [41], e mais recentemente, a *posts* do Twitter [42], de onde surgiram pela primeira vez abordagens *deep learning*, cujos resultados foram relativamente fracos. Essas abordagens foram empregadas para aprendizagem de representações, transferência de aprendizado e para a classificação propriamente dita [43, 44, 45, 46], mas nenhuma delas está puramente associada a notícias.

A detecção de posicionamento em notícias somente foi proposta em 2016 [47], onde se era feito detecção com as três classes básicas utilizando uma regressão logística e o posicionamento era extraído entre manchete e alguma alegação curta, atingindo 73% de acurácia. Nele foi apresentado o primeiro conjunto de dados para detecção de posicionamento em notícias.

De modo geral, os estudos sobre detecção de posicionamento mostram resultados instáveis entre conjuntos de dados diferentes, não havendo, portanto, uma abordagem que sempre seja a melhor. Apesar de *features* linguísticas serem mais frequentes, *features* da relação entre os discursos ou usuários devem ser extraídas quando possível, pois são mais úteis para a classificação. Os resultados também demonstram uma maior dificuldade em se classificar a classe

[1] *Stance* é a tradução inglesa da palavra posicionamento no contexto de opiniões. *Position*, apesar de também ser uma tradução válida, é mais utilizada para se referir à localização.

neutra do que as demais.

Inspiração para este trabalho, o desafio *Fake News Challenge* objetiva criar um sistema de checagem de notícias falsas [8] através da detecção de posicionamento entre notícias. Os organizadores do desafio ampliaram o conjunto de dados de [47] e criaram a abordagem *baseline*. Esse desafio foi o primeiro a propor uma classe para o caso em que os pontos de vistas não estão relacionados de nenhuma forma.

É importante destacar que existem pesquisas sobre notícias falsas. Elas detalham os tipos de notícias falsas presentes, possíveis *features* que podem ser extraídas para identificá-las e métodos de classificação que podem ser usados [48, 49, 50]. Todavia, a identificação automatizada de notícias falsas, motivação deste trabalho e do desafio, foi alcançada por poucos trabalhos [51, 52], devido a dificuldade de se criar uma coleção de dados para tal tarefa e aos baixos resultados que haviam sido alcançados até pouco tempo atrás.

CAPÍTULO 3

Abordagens

Neste capítulo, apresentamos duas abordagens para o problema proposto: uma *deep learning* e uma clássica, que será o *baseline*. São detalhados o pré-processamento do texto, a geração de features e o algoritmo de classificação usado em cada. São apresentados também um conjunto de *features* adicionais usadas nos experimentos.

O código está escrito em Python e bibliotecas externas auxiliaram no desenvolvimento. O uso delas é citado ao longo deste capítulo. Tarefas sem citação de biblioteca foram escritas em Python puro.

3.1 *Deep Learning*

A abordagem *deep learning* idealizada por este trabalho envolve o uso de vetores de palavras e uma rede neural convolucional. Primeiro, manchete e corpo são pré-processados para serem convertidos em vetores de *tokens*. Esses vetores são, então, transformados para índices de *word embeddings* e tratados para que sejam dados como entrada na rede neural.

3.1.1 Pré-processamento

O pré-processamento textual acontece igualmente para manchete e corpo em três etapas:

1. *Case-folding* de todos os caracteres.
2. Tokenização feita pelo spaCy [53]. Ela é baseada em sinais de pontuação e espaços em branco, com remoção dos espaços e manutenção de palavras hifenizadas e sinais de pontuação.
3. Substituição de sequências de algarismos por um único algarismo zero.

3.1.2 Conversão em Índices

Os *tokens* precisam ser convertidos em índices inteiros de *word embeddings*, que serão substituídos por vetores propriamente ditos na primeira camada da arquitetura. Isso é necessário por conta da implementação da biblioteca *deep learning* usada no desenvolvimento, o Keras [54]. Os *word embeddings* utilizados neste trabalho já existiam e foram treinados em páginas do Wikipédia[1]. São 870215 mapeamentos possíveis que incluem palavras, sinais de pontuação, algarismos, etc., e cujos vetores têm 30 dimensões de tamanho.

[1] https://www.wikipedia.org/

Tokens existentes no conjunto de *embeddings* são simplesmente convertidos para os índices inteiros. Contudo, existem tratamentos a serem feitos durante a conversão, dado que nem todo *token* estará presente no conjunto. Esses *tokens* são substituídos por `<unk>`, um *token* especial que está presente no conjunto e simboliza uma palavra desconhecida. Essa parte do processo é a mesma para manchete e corpo.

Como textos têm tamanho variável, consequentemente o número de *tokens* criados também varia. Logo, para manter o espaço de entrada uniforme, é necessário fazer tratamento com *zero padding* no final do vetor de índices. Adiciona-se o índice de valor zero – simbolizado por `<pad>`, cujo vetor é formado apenas por zeros – até que a quantidade de índices do vetor de índices se iguale ao maior vetor de *tokens* existente no corpus. Isso é feito separadamente para manchete e corpo. Ou seja, o vetor de índices de uma manchete fica com tamanho igual ao máximo tamanho entre os vetores de *tokens* de manchetes, e o vetor de índices de um corpo iguala seu tamanho ao maior entre os vetores de *tokens* de corpos.

Por fim, adicionam-se `Z` novos índices zeros no começo do vetor de índices, e outros `Z` no final a fim de que as convoluções não sejam estreitas. O cálculo é feito por `Z=(J+1)/2`, onde `J` é o tamanho da maior janela de convolução aplicada na arquitetura. Esse último tratamento é feito igualmente para os vetores de índices de manchete e de corpo.

3.1.3 Arquitetura

A arquitetura proposta foi desenvolvida com o Keras e teve o Theano como backend [54, 55]. A ideia básica dela é gerar *features* de similaridade com similaridade de cosseno[2] entre *features* convolucionais produzidas na manchete e nas do corpo. No primeiro nível, temos uma camada de *embeddings* não-treináveis, que fazem a conversão do vetor de índices em inteiros nos vetores de vetores de números reais.

O segundo nível da arquitetura é composto por várias camadas horizontais de convoluções unidimensionais de *stride* 1, onde cada uma produz 200 filtros e consequentemente 200 mapas de *features*. São três camadas de convolução para a manchete e mais três para o corpo. Elas tem janelas de tamanho 3, 4 e 5. Portanto, o `J` da subseção anterior é 5, e o `Z` é 3. Após cada convolução dessas, é feito ativação por ReLU para introduzir não-linearidades nos resultados.

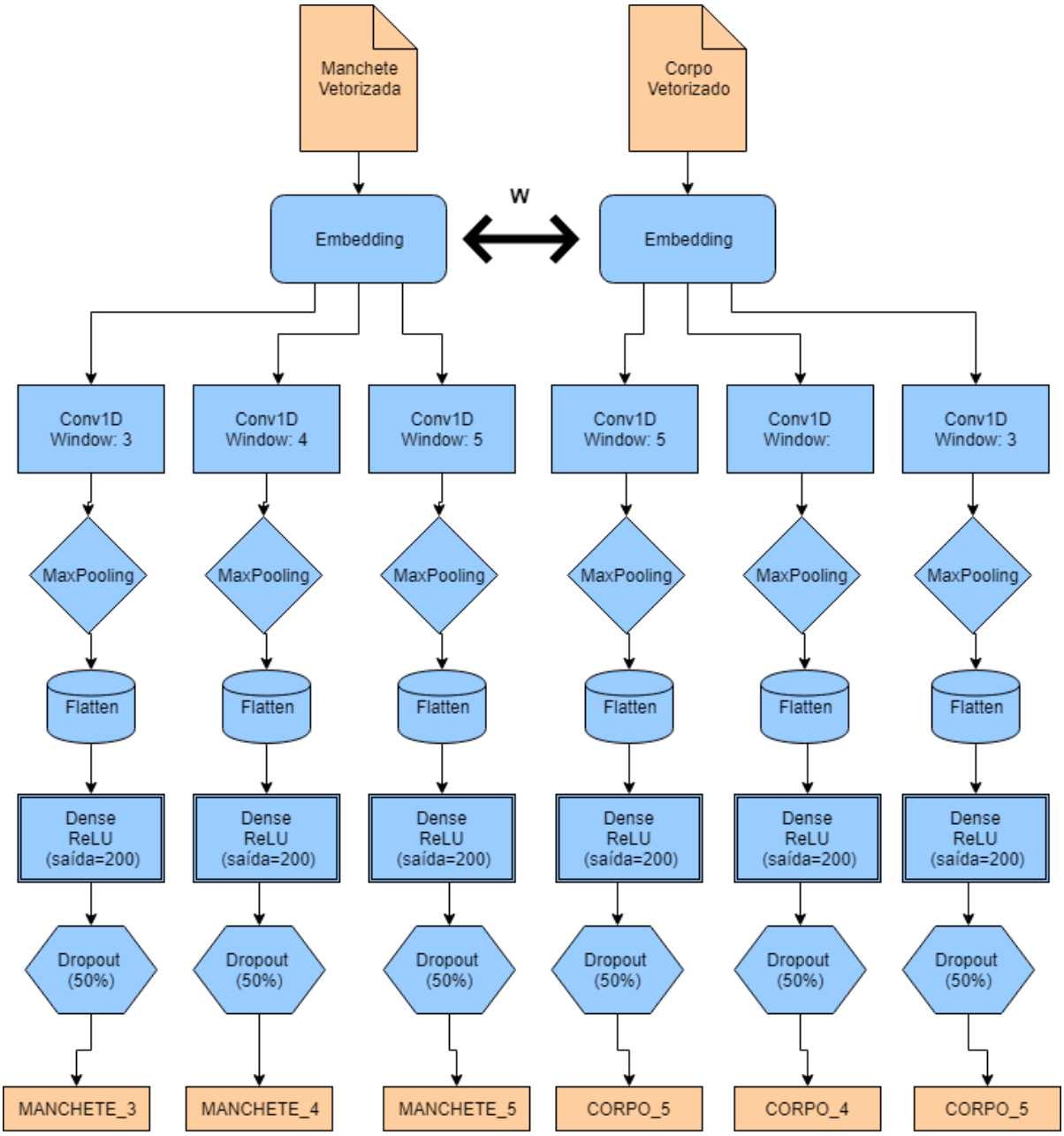
No terceiro nível é aplicado *pooling* unidimensional de máximo com *stride* 2 e uma janela de tamanho 2. Isso reduz o número de *features* pela metade e deve extrair aquelas mais importantes nessas regiões. E então, as *features* restantes são reduzidas e concatenadas em um vetor unidimensional pelas camadas de `Flatten` do Keras [54].

Esses vetores passam por camadas completamente conectadas, geram 200 neurons de saída e são ativadas ou não por ReLU. Isso é necessário para uniformizar a quantidade de neurons dessa fase da arquitetura, já que uma manchete e um corpo tem tamanhos diferentes inicialmente e precisamos de quantidades iguais de *features* para calcular a similaridade.

Depois, esses neurons de saída sofrem *dropout* de 50%. *Dropout* é uma técnica de regularização aplicada durante o treinamento em que aleatoriamente anula-se uma fração dos neurons de saída, principalmente de camadas totalmente conectadas como a `Dense`. Essa primeira parte da arquitetura pode ser visualizada pela Figura 3.1.

[2]Similaridade de cosseno é uma medida de distância em que calcula-se o cosseno do ângulo entre dois vetores.

Figura 3.1 Parte inicial da arquitetura. Em azul, suas camadas. Em amarelo, entradas e saídas.

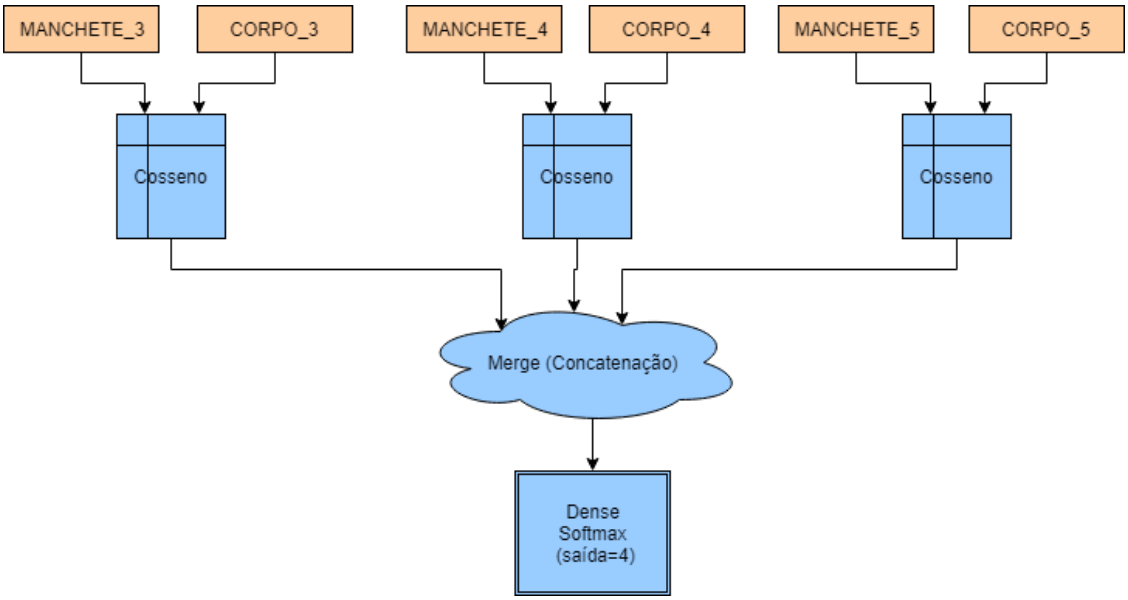


Fonte: Elaborado pelo autor.

Em seguida, como pode se ver na Figura 3.2, calcula-se a similaridade de cosseno entre o vetor de saída da manchete e o vetor do corpo que foram gerados por convoluções de janelas de tamanhos iguais. Os resultados são concatenados em um vetor unidimensional que contém 3 *features* porque as convoluções são feitas com três tamanhos de janelas. Essas *features* são chamadas de `Simil`.

Para a classificação final, esse vetor é dado como entrada em uma camada completamente conectada com 4 saídas, visto que se está trabalhando com 4 classes. A ativação é feita por *softmax*. Essa arquitetura é treinada por 5 épocas com *batches* de tamanho 80. A função de custo é a *cross-entropy* e a otimização é feita pelo algoritmo de *Adam*.

Figura 3.2 Parte final da arquitetura. Em azul, suas camadas. Em amarelo, entradas e saídas.



Fonte: Elaborado pelo autor.

`MANCHETE_j` e `CORPO_j` são vetores de saída pós-*dropout*, onde o `j` refere-se ao tamanho da janela aplicada por uma prévia camada de convolução. Esses vetores formam o conjunto de *features* `Rede`. Em alguns experimentos, esse conjunto e o de *features* complementares são incorporadas à essa arquitetura. Essa agregação é feita logo antes da última camada, concatenando-se essas *features* ao vetor de *features* `Simil`.

3.2 *Baseline*

Na abordagem *baseline*, apresentada recentemente [8, 14], tanto a manchete quanto o corpo de uma notícia passam por um pré-processamento textual. Depois, são extraídas *features* individuais e do par. Por fim, essas *features* são concatenadas em um vetor numérico, que é passado como entrada para o algoritmo de classificação.

3.2.1 Pré-processamento

O pré-processamento textual básico ocorre independente de *feature* e é feito em quatro etapas nesta ordem:

1. Substituição por espaço em branco de todos os caracteres que não sejam letras, números ou espaços em branco.
2. Remoção de espaços em branco do início e do fim da *string*.
3. Remoção de espaços extras em sequências contínuas de espaços em branco.
4. *Case-folding* dos caracteres restantes.

Dependendo da *feature*, outras transformações como a tokenização, a lematização, e remoção de *stopwords* também ocorrem. Elas são explicitamente mencionadas na próxima subseção e foram feitas utilizando funções disponíveis no spaCy [53].

3.2.2 *Features*

As informações extraídas da manchete e do corpo foram idealizadas pelo processo de engenharia de *features*, em que pessoas com conhecimento sobre o domínio do problema se juntam para pensar e escolher as *features* que mais contribuiriam para a acurácia do classificador.

Resultando em um vetor numérico de 44 dimensões, esses 6 grupos de *features* escolhidas formam o conjunto de *features* `Base`. Seus cálculos são explicados abaixo:

- Contagem de char-gramas[3] co-ocorrentes
  - Todos os char-gramas de tamanho `T` são extraídos da manchete pré-processada e sem *stopwords*[4] presentes. Calcula-se quantos char-gramas aparecem nos primeiros `C` caracteres do corpo pré-processado. Esse resultado é a *feature* desejada.
  - Isso é feito para quatro valores de `T`: 2, 4, 8 e 16. E também para três valores de `C`: 100, 255 e corpo completo. Essas combinações geram, portanto, 12 *features*.
- Contagem de n-gramas[5] co-ocorrentes
  - Todos os n-gramas de tamanho `N` são extraídos da manchete pré-processada. Conta-se quantos n-gramas aparecem nos primeiros `C` caracteres do corpo pré-processado. Essa quantidade é a *feature* desejada.
  - Isso é feito para cinco valores de `N`: 2, 3, 4, 5 e 6. E também para dois valores de `C`: 255 e corpo completo. Essas combinações geram mais 10 *features*.

[3] Char-grama é uma sequência de `T` caracteres consecutivos de um texto.

[4] *Stopwords* são palavras de pouco significado e muito frequentes na linguagem natural, como artigos e preposições.

[5] N-grama é uma sequência de `N` *tokens* consecutivos de um texto.

- Contagem de *tokens* co-ocorrentes
  - Tokeniza-se a manchete pré-processada e calcula-se quantos desses *tokens* estão presente no corpo pré-processado completo e nos seus 255 primeiros caracteres.
  - Isso é feito para os *tokens* da manchete com e sem *stopwords* presentes, gerando 4 *features* no total.
- Percentual de lemas co-ocorrentes
  - Extrai-se todos os *tokens* lematizados da manchete e do corpo pré-processados, criando-se dois conjuntos. Divide-se a quantidade de *tokens* na interseção desses conjuntos pela quantidade de *tokens* na união deles. Essa é a única *feature* calculada.
- Polaridade
  - Extrai-se todos os *tokens* lematizados de um texto pré-processado. Utilizando uma lista de 15 palavras refutativas que pode ser encontrada na Tabela 3.1, calcula-se quantas vezes cada palavra dessa está entre os lemas do texto. Então, somam-se essas quantidades em um valor único e calcula-se o resto da divisão por 2. Se este valor for 1, o texto é considerado negativo; se for 0, é considerado positivo.
  - Isso é feito tanto para a manchete quanto para o corpo, logo, são 2 *features*.
- Presença de palavras refutativas
  - Extrai-se todos os *tokens* lematizados da manchete pré-processada. Cria-se um vetor binário em que o valor 1 significa que uma palavra refutativa está entre os lemas da manchete, e o valor 0 que não está.
  - Como isso é feito para todas as palavras da lista de palavras refutativas da Tabela 3.1, são geradas 15 *features*.

Tabela 3.1 Palavras refutativas, escolhidas com base na coleção de dados citada na seção 4.

bogus	debunk	denies
deny	despite	doubt
doubts	fake	false
fraud	hoax	nope
not	pranks	retract

3.2.3 Classificação

É usada a classe `GradientBoostingClassifier` do scikit-learn [56], que é uma implementação de *Gradient Boosted Trees*. Os hiperparâmetros foram ajustados para haverem 200 estágios de *boosting*. Nesta implementação, são geradas 4 árvores de regressão por estágio, uma para cada classe do problema, totalizando 800 árvores no final do processo.

A função de custo é a logística, e para manter um bom nível de generalização, novos nós somente são criados quando a qualidade do *split* é maior que um determinado *threshold*, ou a profundidade do novo nó é menor ou igual a três.

3.3 *Features* Complementares

Outros 4 grupos de *features* foram extraídas das manchetes e dos corpos independente de abordagem e são detalhados abaixo. Elas formam o conjunto de *features* `Extra` e foram pensadas para enriquecer os experimentos, com um total de 923 dimensões:

- *Bag-of-words* clássico
  - Manchetes e corpos são transformadas com *case-folding*, tokenização e remoção de *stopwords* feitos pelo scikit-learn [56]. Então, descobre-se os 200 *tokens* mais frequentes entre todos esses textos.
  - Cria-se um vetor com a frequência desses *tokens* para manchete e outro para corpo, resultando em 400 *features*.
  - Uma *feature* extra é criada pela aplicação da similaridade de cosseno entre o vetor da manchete e o do corpo.
- *Bag-of-words* com tf-idf[6]
  - Funciona como o *bag-of-words* clássico, mas em vez de frequências puras, é calculado o tf-idf. Portanto, são mais 400 *features* individuais e uma da similaridade de cosseno.
- *Bag-of-POS*[7]
  - Pelo spaCy [53], todos os *tokens* das manchetes e corpos são convertidos para suas respectivas partes de discurso.
  - Com o scikit-learn [56], cria-se um vetor de frequências de POS (entre todos achados na coleção) em um dado documento.
  - Como existem 50 POS possíveis, e isso é feito para manchete e corpo separadamente, geram-se 100 *features*.

[6] Tf-idf, ou *term frequency–inverse document frequency*, é uma medida que indica a importância de uma palavra em um documento com relação ao corpus.

[7] POS, *part-of-speech*, ou partes de discurso, são categorias gramaticais dadas às palavras de um texto.

  - Uma *feature* extra é criada por similaridade de cosseno entre o vetor da manchete e o do corpo.

- Tópicos

  - Manchetes e corpos são transformadas com *case-folding*, tokenização e remoção de *stopwords* feitos pelo scikit-learn [56]. Palavras que aparecem em apenas um documento são excluídas.
  - Utilizando o Latent Dirichlet Allocation[8] do scikit-learn [56], gera-se um vetor de 10 dimensões para manchete e outro para corpo, totalizando 20 *features*.

[8]Latent Dirichlet Allocation é um algoritmo que aprende uma representação vetorial de probabilidades para um documento. Cada probabilidade indica se o documento pertence a um dado tópico, e o tamanho do vetor é a quantidade de tópicos escolhidos para a representação. O aprendizado é feito pela distribuição de palavras.

CAPÍTULO 4

Avaliação

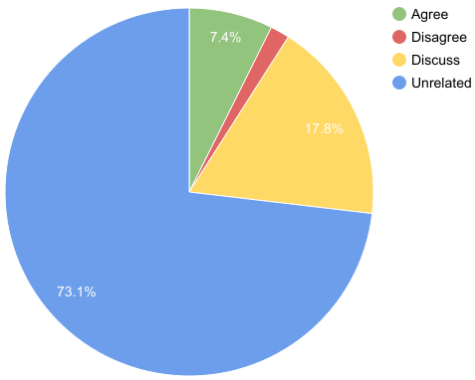
Neste capítulo, são primeiramente descritas a coleção de dados utilizada, a estruturação dos experimentos e as métricas para a avaliação. No fim, os resultados são apresentados e discutidos.

4.1 Coleção de Dados

A coleção utilizada nos experimentos é composta por 49972 pares manchete-corpo, onde cada par tem apenas um rótulo entre quatro possíveis. Eles são:

- *Agree* - quando a manchete tem uma posição de concordância à posição do corpo.
- *Disagree* - quando a manchete tem uma posição de discordância à posição do corpo.
- *Discuss* - quando a manchete está relacionada ao corpo, mas não tem uma posição.
- *Unrelated* - quando a manchete não está relacionada ao corpo.

Figura 4.1 Distribuição dos rótulos na coleção de dados.



Fonte: Elaborada pelo autor.

Esses pares foram gerados através da combinação de 1648 manchetes e 1669 corpos de notícias escritas na língua inglesa, os quais foram coletados e rotulados por jornalistas [47], e estão disponíveis publicamente nos arquivos `train_bodies.csv` e `train_stances.csv` de um repositório da *web* [13].

Nenhuma manchete ou corpo dessa coleção é proveniente de fontes que satirizam e humorizam informações, sejam essas verdadeiras ou falsas. Editoriais ou artigos explícitos de opinião sobre um tópico também foram excluídos pois não se encaixam no próposito do trabalho [8].

4.2 Organização dos Experimentos

Inicialmente, os dados são embaralhados e separados em três conjuntos: um de treinamento, um de validação e um de teste, que tem respectivamente 70%, 10% e 20% do total de instâncias da coleção completa. Cada conjunto desse é montado de forma a não haver repetição de instâncias entre eles e aproximadamente manter a proporção das classes da coleção completa, como demonstrado na Tabela 4.1

Tabela 4.1 Conjuntos criados para os experimentos, mostrando o número de instâncias de cada classe dentro do conjunto indicado.

Conjunto	***Agree***	***Disagree***	***Discuss***	***Unrelated***
Treinamento	2576	588	6237	25583
Validação	368	84	891	3654
Teste	734	168	1781	7308

O aprendizado é feito sobre o conjunto de treinamento em ambas abordagens. O conjunto de validação é usado apenas nos experimentos com *deep learning* para que ao final de todas as épocas obtenha-se o modelo de maior acurácia sobre este conjunto. Para o *baseline*, o conjunto de validação é simplesmente ignorado.

Por se tratar de um problema multiclasse, são extraídas do conjunto de teste a matriz de confusão, e as métricas de macro precisão, macro cobertura, macro f-measure e micro f-measure. Essas métricas permitem avaliar a capacidade de robustez das soluções ao grande desbalanceamento das classes presentes e à sub-representação das classes mais importantes do contexto em que o problema está inserido.

Esse processo é realizado para duas variações da abordagem *baseline* e três variações da *deep learning*, com detalhes apresentados na Tabela 4.2 e Tabela 4.3.

Tabela 4.2 Detalhamento das abordagens e *features* nos experimentos.

ID	**Abordagem**	**Grupos de *features***	**Nº de *features***
`BL`	*baseline*	`Base`	44
`BL_BEXT`	*baseline* modificada	`Base` e `Extra`	967
`DL`	*deep learning*	`Simil`	3
`DL_BEXT`	*deep learning* modificada	`Simil`, `Base` e `Extra`	970
`DL_OUTRO`	*deep learning* modificada	`Simil` e `Rede`	1203

Tabela 4.3 Detalhamento dos tempos de treinamento e teste nos experimentos, aproximando a casa dos segundos. Os tempos mostrados não incluem a produção das *features* `Extra`, que leva cerca de 1 hora.

ID	Treinamento	Teste
`BL`	42s	< 1s
`BL_BEXT`	41min e 55s	< 1s
`DL`	29h 28min e 1s	17min e 12s
`DL_BEXT`	30h 4min e 48s	16min e 21s
`DL_OUTRO`	29h 53min e 24s	16min e 48s

4.3 Resultados

A Tabela 4.4 mostra que os melhores resultados são dos experimentos `DL_BEXT` seguido do `BL_BEXT`. Eles são primeiro e segundo lugares em todas as métricas gerais extraídas. Essa superioridade claramente foi obtida pelas *features* `Extra`, não usadas nos outros experimentos. Elas aparentam ser capazes de aumentar bastante o acerto nas classes subrepresentadas e também melhorar o nível de acerto nas sobrerepresentadas. Isso pode ser ao visto ao se comparar as matrizes de confusão de `BL` com `BL_BEXT`, e de `DL` com `DL_BEXT`.

Tabela 4.4 Resultados gerais dos experimentos, em uma escala de 0 a 100 com aproximação de 2 casas decimais.

	Macro precisão	Macro cobertura	Macro F1	Micro F1
`BL`	67.68	50.59	50.57	88.00
`BL_BEXT`	81.18	70.29	73.82	93.97
`DL`	60.06	57.31	58.46	87.90
`DL_BEXT`	83.85	74.19	76.67	95.36
`DL_OUTRO`	78.18	67.81	72.09	89.33

O terceiro melhor resultado é obtido por `DL_OUTRO`. Comparando-a com `DL`, percebe-se que as *features* `Rede` foram as responsáveis por acertar mais as classes subrepresentadas. E à exceção da micro F1, suas métricas gerais superam consideravelmente `DL` e `BL`. Enquanto isso, estas duas possuem os piores resultados, equiparáveis pelos prós e contras que apresentam em suas matrizes de confusão.

`BL`, visto na Tabela 4.5, tem resultados fracos nas classes *agree* e *disagree*, e bons nas outras. É isto que faz com que seu micro F1 não seja tão distante do resto, mas que seu macro F1 seja péssimo. Ela confunde muitos pares *agree* e *disagree* com *discuss*. É visível que o classificador não aprende o que é tomar ou não uma posição. Ele aprende pela distribuição e estima que a maioria dos pares não *unrelated* são *discuss*. As *features* `Base` sozinhas não distinguem o suficiente e precisam ser complementadas com outras *features*, como é o caso de `BL_BEXT`.

Tabela 4.5 Matriz de confusão para BL. Os rótulos verdadeiros são indicados pelas linhas.

	Agree	**Disagree**	**Discuss**	**Unrelated**
Agree	128	1	523	82
Disagree	24	4	117	23
Discuss	56	1	1508	216
Unrelated	8	2	146	7152

BL_BEXT tem o segundo menor erro para a classe *disagree*, um ótimo resultado com *agree* e uma melhoria nas confusões com a classe *discuss*. Todavia, pela Tabela 4.6, essas confusões ainda existem. Posto que isso não ocorre nos experimentos *deep learning*, é possível que uma parcela desse erro aconteça por conta do algoritmo de classificação e aprendizado do *baseline*.

Tabela 4.6 Matriz de confusão para BL_BEXT. Os rótulos verdadeiros são indicados pelas linhas.

	Agree	**Disagree**	**Discuss**	**Unrelated**
Agree	434	15	241	44
Disagree	48	55	53	12
Discuss	78	8	1609	86
Unrelated	10	2	65	7231

No DL, apesar de que vários pares *agree* são classificados como *unrelated*, os resultados com *agree* são bons e superam bastante os de BL. Também é interessante que apenas três *features* consigam uma equivalência aos resultados gerais de BL. Contudo, existe uma confusão entre instâncias de *discuss* e *unrelated* provada pela Tabela 4.7. Além disso, a incapacidade de conseguir classificar corretamente qualquer par *disagree* torna essa abordagem realmente fraca. As *features* Simil não são satisfatoriamente informativas a esse nível, provavelmente porque estão em pequena quantidade.

Tabela 4.7 Matriz de confusão para DL. Os rótulos verdadeiros são indicados pelas linhas.

	Agree	**Disagree**	**Discuss**	**Unrelated**
Agree	394	0	38	302
Disagree	93	0	10	65
Discuss	23	0	1431	327
Unrelated	69	0	282	6957

O que deu os melhores resultados a DL_BEXT foi a classe *agree*. Qualquer outro experimento teve acertos muito inferiores para essa classe, tanto que a diferença entre ela e o segundo

lugar (DL_OUTRO) é de aproximadamente 150 pares, como mostram a Tabela 4.8 e Tabela 4.9. O DL_BEXT também tem a melhor taxa de verdadeiros positivos para *discuss* e *unrelated*. Se houvesse menos classificações de *disagree* como *agree*, e *discuss* como *unrelated*, seu macro F1 ultrapassaria os 80%.

Tabela 4.8 Matriz de confusão para DL_BEXT. Os rótulos verdadeiros são indicados pelas linhas.

	Agree	**Disagree**	**Discuss**	**Unrelated**
Agree	603	19	68	44
Disagree	98	40	22	8
Discuss	32	1	1630	118
Unrelated	19	2	33	7254

Tabela 4.9 Matriz de confusão para DL_OUTRO. Os rótulos verdadeiros são indicados pelas linhas.

	Agree	**Disagree**	**Discuss**	**Unrelated**
Agree	456	36	31	211
Disagree	47	62	2	57
Discuss	11	2	1346	422
Unrelated	95	6	146	7061

O experimento DL_OUTRO confirmou o a tendência de sucesso da classe *agree* nas abordagens *deep learning*. Foi o segundo melhor acerto de pares *agree* e o primeiro de pares *disagree*. O erro entre *unrelated* e *discuss* continuou existindo como no DL, piorando para pares *discuss* classificados como *unrelated*. Este fator impediu sua performance de ser equiparada às melhores. Ainda assim, as *features* Rede provaram-se aproveitáveis.

CAPÍTULO 5

Conclusão

Este trabalho apresentou arquiteturas *deep learning* para o problema de detecção de posicionamento entre manchetes e corpos de notícias *online*, motivado pelo desafio *Fake News Challenge* [8]. Vimos que essa tarefa nunca havia sido aplicada antes no contexto de notícias, e que a proposta de usá-la na detecção de notícias falsas é corroborada por profissionais checadores de fatos. Fora isso, foi a primeira vez que a classe *unrelated* foi inclusa.

Avaliados com uma coleção de dados desbalanceada pelo método *holdout*, a arquitetura que continha apenas *features* de similaridade alcançou resultados ligeiramente inferiores ao *baseline*, por não conseguir acertar casos com oposição. E a que possuía outros tipos de *features* produzidas pela rede chegou a resultados ligeiramente superiores, provavelmente pelo grande número de *features*, quando comparada à outra arquitetura.

Features adicionais não originadas da rede *deep learning* foram concebidas e mostraram-se muito úteis. Resultados significantemente superiores ao *baseline* foram alcançados quando elas foram utilizadas em conjunto tanto do próprio *baseline* quanto da arquitetura *deep learning*. Os melhores resultados foram produzidos por esta última combinação e podem ser usados no mundo real em breve.

É possível perceber que cada abordagem tem suas vantagens e desvantagens em termos dos acertos sobre classes. Contudo, de forma geral, a classe *disagree* foi a mais difícil de se classificar e a classe *unrelated* foi a mais fácil, por conta da distribuição dos rótulos. Em termos de tempo, é notável a dificuldade de se treinar as redes *deep learning*.

5.1 Trabalhos Futuros

Além do que foi citado na introdução deste trabalho sobre estudos que liguem diretamente a detecção de posicionamento à detecção de notícias falsas, é necessário estudar o poder de generalização das abordagens criadas. Pode-se conseguir isso utilizando o método *k-fold* de validação cruzada ou outros conjuntos de dados, quando surgirem.

Também seria proveitoso um estudo sobre o impacto de cada *feature* proposta por este trabalho, assim como testes com redes neurais mais simples como *Multilayer Perceptron*. Elas poderiam usar apenas as *features* complementares e ajudariam a aproximadamente medir o nível de importância das *features* geradas por *deep learning*.

Visto que a arquitetura *deep learning* sem *features* complementares tem resultados próximos ao *baseline*, outros trabalhos podem explorar novas arquiteturas *deep learning* que produzam mais *features*, principalmente de similaridade. Novas combinações não experimentadas por este trabalho entre *features* e abordagens também podem ser analisadas, assim como a clas-

sificação em cascata com *deep learning* ou híbrida, desde que se tenham mais exemplos para as classes subrepresentadas.

E ainda que os melhores resultados encontrados neste trabalho sejam bons, eles não são ótimos para a classe *disagree*, então é preciso estudar maneiras de melhorar essa classe. Possíveis abordagens incluem a criação de novas *features* refutativas ou de polaridade, e a compensação dessa classe por sobreamostragem dela ou subamostragem das outras classes. Além disso, experimentos com vetores de palavras de tamanhos maiores, treináveis sobre o corpus, ou pré-treinados no contexto de notícias devem aumentar os resultados gerais.

Referências Bibliográficas

[1] Many Americans Believe Fake News Is Sowing Confusion. Disponível em: <http://www.journalism.org/2016/12/15/many-americans-believe-fake-news-is-sowing-confusion/>. Acesso em: 4 abr. 2017.

[2] Introducing Factmata — Artificial intelligence for automated fact-checking. Disponível em: <https://medium.com/factmata/introducing-factmata-artificial-intelligence-for-political-fact-checking-db8acdbf4cf1>. Acesso em: 4 abr. 2017.

[3] Nas redes, mentiras sobre pleito nos EUA superam notícias reais. Disponível em: <http://www1.folha.uol.com.br/mundo/2016/11/1833158-nas-redes-mentiras-sobre-pleito-nos-eua-superam-noticias-reais.shtml/>. Acesso em: 4 abr. 2017.

[4] Fake news: an insidious trend that's fast becoming a global problem. Disponível em: <https://www.theguardian.com/media/2016/dec/02/fake-news-facebook-us-election-around-the-world>. Acesso em: 4 abr. 2017.

[5] H. Allcott and M. Gentzkow, "Social media and fake news in the 2016 election," tech. rep., National Bureau of Economic Research, 2017.

[6] Can 'Fake News' Impact The Stock Market?. Disponível em: <https://www.forbes.com/sites/kenrapoza/2017/02/26/can-fake-news-impact-the-stock-market/#17700ba12fac>. Acesso em: 4 abr. 2017.

[7] What is fake news, what are Facebook and Google doing about it, why is the Government boycotting YouTube and how can I tell if a site is legitimate?. Disponível em: <https://www.thesun.co.uk/news/2188911/fake-news-inquiry-facebook-google-boycott/>. Acesso em: 4 abr. 2017.

[8] FNC - Fake News Challenge. Disponível em: <http://www.fakenewschallenge.org/>. Acesso em: 2 fev. 2017.

[9] L. Deng, D. Yu, *et al.*, "Deep learning: methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[10] IBM to cut more than 111,000 jobs in largest corporate lay-off ever. Disponível em: <http://www.ibtimes.co.uk/ibm-cut-more-111000-jobs-this-week-largest-corporate-lay-off-ever-1485128>. Acesso em: 1 jun. 2017.

[11] IBM denies plans to cut 26% of workforce. Disponível em: <https://www.theguardian.com/technology/2015/jan/26/ibm-technology-giant-110000-jobs-cut-claims>. Acesso em: 1 jun. 2017.

[12] IBM To Cut More Than 110,000 Jobs, Report Says. Disponível em: <http://news.sky.com/story/ibm-to-cut-more-than-110000-jobs-report-says-10373937>. Acesso em: 1 jun. 2017.

[13] Stance Detection dataset for FNC-1. Disponível em: <https://github.com/FakeNewsChallenge/fnc-1>. Acesso em: 7 fev. 2017.

[14] A baseline implementation for FNC-1. Disponível em: <https://github.com/FakeNewsChallenge/fnc-1-baseline>. Acesso em: 10 mar. 2017.

[15] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.

[16] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *Journal-Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[17] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean, "Boosting algorithms as gradient descent," in *Advances in neural information processing systems*, pp. 512–518, 2000.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[19] G. G. Chowdhury, "Natural language processing," *Annual review of information science and technology*, vol. 37, no. 1, pp. 51–89, 2003.

[20] C. D. Manning, P. Raghavan, H. Schütze, *et al.*, *Introduction to information retrieval*, vol. 1. Cambridge university press Cambridge, 2008.

[21] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986.

[22] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[24] J. Wiebe, "Learning subjective adjectives from corpora," *AAAI/IAAI*, vol. 20, no. 0, p. 0, 2000.

[25] J. Wiebe, T. Wilson, R. Bruce, M. Bell, and M. Martin, "Learning subjective language," *Computational linguistics*, vol. 30, no. 3, pp. 277–308, 2004.

[26] J. Wiebe and E. Riloff, "Creating subjective and objective sentence classifiers from unannotated texts.," in *CICLing*, vol. 5, pp. 486–497, Springer, 2005.

[27] H. Yu and V. Hatzivassiloglou, "Towards answering opinion questions: Separating facts from opinions and identifying the polarity of opinion sentences," in *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pp. 129–136, Association for Computational Linguistics, 2003.

[28] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up?: sentiment classification using machine learning techniques," in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pp. 79–86, Association for Computational Linguistics, 2002.

[29] T. Wilson, J. Wiebe, and P. Hoffmann, "Recognizing contextual polarity in phrase-level sentiment analysis," in *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pp. 347–354, Association for Computational Linguistics, 2005.

[30] W.-H. Lin, T. Wilson, J. Wiebe, and A. Hauptmann, "Which side are you on?: identifying perspectives at the document and sentence levels," in *Proceedings of the tenth conference on computational natural language learning*, pp. 109–116, Association for Computational Linguistics, 2006.

[31] M. Thomas, B. Pang, and L. Lee, "Get out the vote: Determining support or opposition from congressional floor-debate transcripts," in *Proceedings of the 2006 conference on empirical methods in natural language processing*, pp. 327–335, Association for Computational Linguistics, 2006.

[32] R. Malouf and T. Mullen, "Taking sides: User classification for informal online political discourse," *Internet Research*, vol. 18, no. 2, pp. 177–190, 2008.

[33] A. Murakami and R. Raymond, "Support or oppose?: classifying positions in online debates from reply activities and opinion expressions," in *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pp. 869–875, Association for Computational Linguistics, 2010.

[34] S. Somasundaran and J. Wiebe, "Recognizing stances in ideological on-line debates," in *Proceedings of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*, pp. 116–124, Association for Computational Linguistics, 2010.

[35] K. S. H. V. NG and K. Hasan, "Predicting stance in ideological debate with rich linguistic knowledge," in *24th International Conference on Computational Linguistics*, p. 451, 2012.

[36] K. S. Hasan and V. Ng, "Stance classification of ideological debates: Data, models, features, and constraints.," in *IJCNLP*, pp. 1348–1356, 2013.

[37] K. S. Hasan and V. Ng, "Extra-linguistic constraints on stance recognition in ideological debates.," in *ACL (2)*, pp. 816–821, 2013.

[38] S. Somasundaran and J. Wiebe, "Recognizing stances in online debates," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pp. 226–234, Association for Computational Linguistics, 2009.

[39] M. A. Walker, P. Anand, R. Abbott, and R. Grant, "Stance classification using dialogic properties of persuasion," in *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 592–596, Association for Computational Linguistics, 2012.

[40] P. Anand, M. Walker, R. Abbott, J. E. F. Tree, R. Bowmani, and M. Minor, "Cats rule and dogs drool!: Classifying stance in online debate," in *Proceedings of the 2nd workshop on computational approaches to subjectivity and sentiment analysis*, pp. 1–9, Association for Computational Linguistics, 2011.

[41] A. Faulkner, "Automated classification of stance in student essays: An approach using stance target information and the wikipedia link-based measure," *Science*, vol. 376, no. 12, p. 86, 2014.

[42] S. Mohammad, S. Kiritchenko, P. Sobhani, X.-D. Zhu, and C. Cherry, "Semeval-2016 task 6: Detecting stance in tweets.," in *SemEval@ NAACL-HLT*, pp. 31–41, 2016.

[43] I. Augenstein, A. Vlachos, and K. Bontcheva, "Usfd at semeval-2016 task 6: Any-target stance detection on twitter with autoencoders.," in *SemEval@ NAACL-HLT*, pp. 389–393, 2016.

[44] I. Augenstein, T. Rocktäschel, A. Vlachos, and K. Bontcheva, "Stance detection with bidirectional conditional encoding," *arXiv preprint arXiv:1606.05464*, 2016.

[45] W. Wei, X. Zhang, X. Liu, W. Chen, and T. Wang, "pkudblab at semeval-2016 task 6: A specific convolutional neural network system for effective stance detection.," in *SemEval@ NAACL-HLT*, pp. 384–388, 2016.

[46] G. Zarrella and A. Marsh, "Mitre at semeval-2016 task 6: Transfer learning for stance detection," *arXiv preprint arXiv:1606.03784*, 2016.

[47] W. Ferreira and A. Vlachos, "Emergent: a novel data-set for stance classification," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, ACL, 2016.

[48] Y. Chen, N. J. Conroy, and V. L. Rubin, "Misleading online content: Recognizing clickbait as false news," in *Proceedings of the 2015 ACM on Workshop on Multimodal Deception Detection*, pp. 15–19, ACM, 2015.

[49] V. L. Rubin, Y. Chen, and N. J. Conroy, "Deception detection for news: three types of fakes," *Proceedings of the Association for Information Science and Technology*, vol. 52, no. 1, pp. 1–4, 2015.

[50] N. J. Conroy, V. L. Rubin, and Y. Chen, "Automatic deception detection: Methods for finding fake news," *Proceedings of the Association for Information Science and Technology*, vol. 52, no. 1, pp. 1–4, 2015.

[51] V. L. Rubin, N. Conroy, and Y. Chen, "Towards news verification: Deception detection methods for news discourse," in *Hawaii International Conference on System Sciences*, 2015.

[52] V. L. Rubin, N. J. Conroy, Y. Chen, and S. Cornwell, "Fake news or truth? using satirical cues to detect potentially misleading news.," in *Proceedings of NAACL-HLT*, pp. 7–17, 2016.

[53] spaCy: Industrial-Strength Natural Language Processing in Python. Disponível em: <`https://spacy.io/`>. Acesso em: 25 mai. 2017.

[54] F. Chollet *et al.*, "Keras." `https://github.com/fchollet/keras`, 2015.

[55] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.

[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

Este volume foi tipografado em LaTeX na classe UFPEThesis (www.cin.ufpe.br/~paguso/ufpethesis).