UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Uma Proposta de Classificação de Tweets sobre Trânsito Utilizando Diferentes Técnicas de Deep Learning

Estevan Barbará Teixeira

Orientador

Dsc. Carlos Alberto Vieira Campos

Co-orientador

Dsc. Pedro Nuno de Souza Moura

RIO DE JANEIRO, RJ - BRASIL
DEZEMBRO DE 2020

Uma Proposta de Classificação de Tweets sobre Trânsito Utilizando Diferentes Técnicas de Deep Learning

ESTEVAN BARBARÁ TEIXEIRA

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

Dsc. Carlos Alberto Vieira Campos — UNIRIO

Dsc. Pedro Nuno de Souza Moura — UNIRIO

Dsc. Sidney Cunha de Lucena — UNIRIO

Dsc. Antônio Augusto de Aragão Rocha — UFF

Dsc. Jean-Pierre Briot — Sorbonne Université

RIO DE JANEIRO, RJ - BRASIL
DEZEMBRO DE 2020.

Teixeira, Estevan Barbará
T262 Uma Proposta de Classificação de Tweets sobre Trânsito Utilizando Diferentes Técnicas de Deep Learning / Estevan Barbará Teixeira. – Rio de Janeiro, 2020.
89p.

Orientador: Carlos Alberto Vieira Campos.
Coorientador: Pedro Nuno de Souza Moura.
Dissertação (Mestrado) - Universidade Federal do Estado do Rio de Janeiro, Programa de Pós-Graduação em Informática, 2020.

1. Sistemas de Informação. 2. Cidades Inteligentes. 3. Processamento de Linguagem Natural. 4. Deep Learning. I. Campos, Carlos Alberto Vieira, orient. II. Moura, Pedro Nuno de Souza, coorient. III. Título.

Dedico essa dissertação à nobre matriarca Vera Lúcia, que me apoiou incansavelmente em todos os altos e baixos dessa jornada.

Agradecimentos

Agradeço a minha mãe Vera Lúcia, que partilhou comigo todos os percalços e alegrias desse caminho e jamais deixou de me apoiar na jornada, e a meu pai Nelson que, de onde estiver, tenho certeza que me olha com o mesmo carinho e apoio de quando esteve entre nós.

Aos meus orientadores Carlos Alberto Vieira Campos e Pedro Nuno de Souza Moura, cuja experiência e didática me guiaram em todo esse processo, me estimulando e desafiando contra todos os limites que eu acreditava intransponíveis.

Aos amigos alunos e professores da Unirio, em particular aos companheiros Thiago Medeiros e Daniel Costa, por toda a troca de ideias e parceria nos estudos, alegrias e dificuldades.

Aos colegas presentes e passados do MP em Mapas, particularmente, Rhenan Bartels, Felipe Ferreira e Gabriel Barbier, por toda a compreensão e apoio mesmo enquanto eu me dividia entre as responsabilidades.

E a todos os amigos, muito numerosos para nomear cada um, que comigo dividiram algum passo desse caminho e possibilitaram chegar até aqui.

Teixeira, Estevan Barbará, **Uma Proposta de Classificação de Tweets sobre Trânsito Utilizando Diferentes Técnicas de Deep Learning**. UNIRIO, 2020. 89 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

RESUMO

Obter informação relevante de forma rápida, de modo a apoiar tomadas de decisão, é um dos mais importantes desafios da mobilidade urbana. Em tal contexto, rede sociais são fontes bastante interessantes para obtenção de conhecimento sobre situação de trânsito, muito embora, a própria heterogeneidade das informações postadas e da gramática usada exijam o uso de classificadores sobre os textos, de modo a garantir a utilidade das informações usadas. No tocante ao idioma português, há pouca pesquisa sobre classificação desse tipo de texto, particularmente explorando as capacidades de redes neurais. Assim, esta pesquisa apresenta um modelo para representar e classificar microtextos para o idioma português por meio de técnicas modernas de redes neurais, particularmente deep learning, visando obter informações relevantes de trânsito. Para este fim, analisou-se os resultados de diversas combinações entre arquiteturas de deep learning para representar e classificar textos, bem como os efeitos da redução dimensional sobre os sistemas de representação, alcançando resultados competitivos, acima de 94%, em acurácia, precisão e recall.

Palavras-chave: sistemas de transportes inteligentes, redes sociais, processamento de linguagem natural, deep learning, redução de dimensões, sensibilidade dimensional

ABSTRACT

To quickly obtain relevant information, as to support decision making, is one of the most important challenges of urban mobility. In this context, social networks are very interesting sources to gather knowledge about traffic situation, although heterogeneity itself from posted information and used grammar demands use of classifiers on the texts, as to ensure the usefulness of the gathered information. Regarding the Portuguese language, there is little research about classification of this kind of text, particularly by exploiting neural network capabilities. Thus, this research presents a model to represent and classify microtexts in the Portuguese language using modern neural network techniques, notedly deep learning, aiming to obtain relevant traffic information. To this end, we analised the results of many combinations with deep learning architectures to represent and classify texts, as well as the effects of dimension reduction on the representation systems, reaching competitive results, over 94% in accuracy, precision and recall.

Keywords: intelligent transport systems, social networks, natural language processing, deep learning, dimensional reduction, dimension sensibility

Sumário

1 Introdução **1**

1.1 Motivação 2

1.2 Formulação do Problema 4

1.3 Justificativa 4

1.4 Objetivo 5

1.5 Contribuições 5

1.6 Estrutura da Dissertação 6

2 Processamento de Linguagem Natural **8**

2.1 Classificação 8

2.1.1 Deep Learning 9

2.1.2 Redes Convolucionais 11

2.1.3 Redes Recorrentes 11

2.2 Representação 12

2.2.1 Bag-of-words 13

2.2.2 Word Embeddings 14

2.3 Dimensionalidade 14

2.3.1 Análise de Componente Principal 15

3 Trabalhos Relacionados **16**

4 Proposta de classificação de tweets de trânsito em português **20**

4.1 Problema 20

4.1.1 Classificação por Deep Learning 20

4.1.2 Características do idioma português 20

4.2 Metodologia 20

4.2.1 Entrada e pré-processamento 21

4.2.2 Vetorização 21

4.2.2.1 Redução de Dimensionalidade 21

4.2.3 Classificação 22

4.2.4 Saída 23

5 Experimentos e Resultados **24**

5.1 Ambiente 24

5.2 Modelos utilizados 25

5.3 Coleta e etiquetamento do Dataset 25

5.4 Processamento 27

5.4.1 Pré-processamento 27

5.4.2 Vetorização 27

5.4.2.1 Redução dimensional 29

5.4.3 Classificação 29

5.4.3.1 CNN 29

5.4.3.2 LSTM 32

5.4.3.3 Modelo Misto 32

5.5 Resultados 33

5.5.1 Tempo de Vetorização 34
5.5.2 Tempo de Treinamento 37
5.5.3 Acurácia 43
5.5.4 Precisão 51
5.5.5 Recall 57
5.5.6 Medida F1 57

6 Conclusão **64**

Lista de Figuras

1.1 Trajeto de um tufão obtido por redes sociais. Extraído de [30] 3

2.1 Esquema de um *multilayer perceptron*, mostrando as camadas e características obtidas, extraído de [8], originalmente publicado por [43] 10

2.2 Esquema de uma rede convolucional, extraído de [31]. 12

2.3 Esquema de uma rede LSTM, extraído de [9] 13

3.1 Exemplo de associação de tweets a congestionamentos. Extraído de [42] . 17

4.1 Sequência de etapas do modelo proposto para a representação e classificação de microtextos. 21

4.2 Modelo CNN + LSTM 23

5.1 Retângulo envolvente utilizado (Representação gerada no site `http://geojson.io/`) 26

5.2 Histograma da quantidade de palavras 27

5.3 Distribuição cumulativa da quantidade de palavras 28

5.4 Comparação do tempo de vetorização dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN. 34

5.5 Comparação do tempo de vetorização dos modelos Word2Vec, GloVe e fastText utilizando classificação LSTM 36

5.6 Comparação do tempo de vetorização dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN+LSTM 37

5.7 Comparação do tempo de vetorização do modelo em 300 dimensões com os modelos reduzidos via PCA, entre as diversas combinações de vetorização e classificação. 38

5.8 Comparação do tempo de treinamento dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização Word2Vec. 40

5.9 Comparação do tempo de treinamento dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização GloVe. 41

5.10 Comparação do tempo de treinamento dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização fastText. 41

5.11 Comparação do tempo de treinamento dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN. 42

5.12 Comparação do tempo de treinamento dos modelos Word2Vec, GloVe e fastText utilizando classificação LSTM. 42

5.13 Comparação do tempo de treinamento dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN+LSTM. 43

5.14 Comparação do tempo de treinamento do modelo em 300 dimensões com os modelos reduzidos via PCA, entre as diversas combinações de vetorização e classificação. 44

5.15 Comparação da porcentagem de acurácia dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização Word2Vec. 46

5.16 Comparação da porcentagem de acurácia dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização GloVe. 47

5.17 Comparação da porcentagem de acurácia dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização fastText. 47

5.18 Comparação da porcentagem de acurácia dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN. 48

5.19 Comparação da porcentagem de acurácia dos modelos Word2Vec, GloVe e fastText utilizando classificação LSTM 48

5.20 Comparação da porcentagem de acurácia dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN+LSTM 49

5.21 Comparação da porcentagem de acurácia do modelo em 300 dimensões com os modelos reduzidos via PCA, entre as diversas combinações de vetorização e classificação 50

5.22 Comparação da porcentagem de precisão dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização Word2Vec. 51

5.23 Comparação da porcentagem de precisão dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização GloVe. 53

5.24 Comparação da porcentagem de precisão dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização fastText. 54

5.25 Comparação da porcentagem de precisão dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN. 54

5.26 Comparação da porcentagem de precisão dos modelos Word2Vec, GloVe e fastText utilizando classificação LSTM 55

5.27 Comparação da porcentagem de precisão dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN+LSTM 55

5.28 Comparação da porcentagem de precisão do modelo em 300 dimensões com os modelos reduzidos via PCA, entre as diversas combinações de vetorização e classificação. 56

5.29 Comparação da porcentagem de recall dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização Word2Vec. 57

5.30 Comparação da porcentagem de recall dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização GloVe. 59

5.31 Comparação da porcentagem de recall dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização fastText. 60

5.32 Comparação da porcentagem de recall dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN. 60

5.33 Comparação da porcentagem de recall dos modelos Word2Vec, GloVe e fastText utilizando classificação LSTM 61

5.34 Comparação da porcentagem de recall dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN+LSTM 61

5.35 Comparação da porcentagem de recall do modelo em 300 dimensões com os modelos reduzidos via PCA, entre as diversas combinações de vetorização e classificação. 62

Lista de Tabelas

5.1 Estatísticas dos Corpora usados na vetorização. Adaptado de [10] 30

5.2 Tempos de vetorização sem PCA 35

5.3 Tempos de vetorização com PCA 36

5.4 Tempos de treinamento sem PCA 39

5.5 Tempos de treinamento com PCA 40

5.6 Percentagem de Acurácia sem PCA 45

5.7 Percentagem de Acurácia com PCA 46

5.8 Comparação das medidas de acurácia 49

5.9 Percentagem de Precisão sem PCA 52

5.10 Percentagem de Precisão com PCA 53

5.11 Percentagem de Recall sem PCA 58

5.12 Percentagem de Recall com PCA 59

5.13 Comparação das medidas F1 63

List of Algorithms

1 Algoritmo de pós-processamento (adaptado de [28]) 22
2 Algoritmo de redução dimensional (adaptado de [28]) 22
3 Algoritmo de vetorização 28
4 Algoritmo de configuração da CNN 29
5 Algoritmo de configuração da LSTM 32
6 Algoritmo de configuração CNN+LSTM 33

Lista de Nomenclaturas

CBOW	Continuous Bag-Of-Words
CNN	Convolutional Neural Network
CSV	Comma Separated Values
GloVe	Global Vectors
ITS	Intelligent Transportation Systems
LSTM	Long Short-Term Memory
NILC	Núcleo Interinstitucional de Linguística Computacional
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
PCA	Principal Component Analysis
PPA	Post-processing Algorithm
PyPI	Python Package Index
RNN	Recurrent Neural Network
Tf-Idf	Term Frequency/Inverse document frequency

1. Introdução

Uma parte bastante significativa da vida em qualquer grande centro urbano é gasto no deslocamento entre pontos. Considerando a grande população em tais centros, como cerca de 6,7 milhões de habitantes no Rio de Janeiro, ou 12,3 milhões em São Paulo [12], entende-se a necessidade de ordenação na movimentação de tal enorme contingente humano entre seus diversos pontos de interesse. Uma vez que as vias urbanas dificilmente atendem à demanda, é perfeitamente possível que algo entre um sexto e um quarto de um dia normal sejam dedicados à simples movimentação entre casa e trabalho. Isso sem levar em conta as dificuldades que o ambiente urbano desordenado impõe aos trabalhos que dependem de deslocamento ágil, como entregas, transporte de passageiros e mesmos serviços essenciais, como bombeiros, ambulâncias e patrulhamento policial. Mesmo o impacto financeiro de tais questões é prejudicial, com custos de manutenção das vias sobreutilizadas sendo repassadas ao público, e as próprias taxas de transporte e entrega crescendo continuamente. Assim faz-se natural que seja investido tempo e recursos para desenvolver tecnologias que possibilitem tais deslocamentos serem mais rápidos, seguros e interessantes.

A disciplina de Cidades Inteligentes surge como o campo de estudos que busca aprimorar o planejamento e administração das áreas urbanas através de pesquisas e tecnologias, objetivando a melhoria da qualidade de vida dos cidadãos. Dentre seus focos, o campo de Sistemas de Transporte Inteligentes, ou Intelligent Transportation Systems (ITS), em particular se interessa pelas questões de deslocamento em seus ambientes. ITS *compreendem a união de diversas tecnologias objetivando prover otimização abrangente da mobilidade urbana de uma cidade e gerar maior segurança aos motoristas e conforto e entretenimento aos passageiros*, como definido por Meneguette, De Grande e Loureiro, no livro *Intelligent Transport System in Smart Cities: Aspects and Challenges of Vehicular Networks and Cloud* [20].

Tais ferramentas integram dados absorvidos do ambiente de trânsito, tais como câme-

ras, radares, sensores de velocidade, etc., assim como informações obtidas diretamente de seus usuários, para compor bases de conhecimento da situação atual e histórica do trânsito, e oferecer uma ampla gama de serviços aos interessados. Esses serviços são em geral, agrupados em três categorias específicas, a saber:

- Segurança, que envolve aplicações focadas em proteger a integridade física dos usuários, de outros indivíduos envolvidos no trânsito, bem como de seus bens. Tais aplicações incluem alertas de risco de colisão, notificações sobre perigos na estrada e sistemas de alerta para serviços de emergência;
- Eficiência de tráfego, que tem por objetivo minimizar o gasto de recursos (tempo e combustível) nas viagens, melhorando, por extensão, a situação ambiental e econômica da região. Tais ferramentas incluem sistemas de navegação, sinais de trânsito adaptativos e centrais de informações aos motoristas; e
- Entretenimento, as quais visam tornar as viagens mais confortáveis aos passageiros. Tais aplicações podem incluir jogos eletrônicos, compartilhamento de conteúdos e sistemas de recomendação de serviços urbanos.

Cabe observar que, apesar de tais agrupamentos, nada impede que um mesmo software ofereça mais de um tipo de serviço, como por exemplo *Waze*[1] e *Google Maps*[2], que fornecem todas as três categorias.

1.1 Motivação

O reconhecimento de informações relativas a eventos prejudiciais ao trânsito requer uma poderosa infraestrutura de ITS, de modo a reconhecer e informar rapidamente ao público sobre tanto os congestionamentos e incidentes quanto as rotas alternativas. Numa situação de cidades em crescimento desordenado (como a maioria das grandes cidades brasileiras), tal infraestrutura é insuficiente ou mesmo inexistente. Assim, o desafio de coletar tais informações passa por buscar uma fonte alternativa.

O uso do próprio público interessado como fonte de informação, através de um sistema colaborativo, é ideal para tal situação. E, dada a natureza das redes sociais, em que informações de estado imediato são prontamente partilhadas, tem-se então fonte suculenta para obter tais dados. Portanto a busca nessas redes por informações que possam

[1] https://www.waze.com/
[2] https://www.google.com/maps

impactar em indivíduos similares é uma opção natural [39]. Contudo, é muito provável que indivíduos com interesse em informações específicas não conheçam quem as possui, e mesmo quando possuem esse acesso fácil, estarão limitados pela interface da rede social, bem como pelo volume de informações inúteis obtidos. Assim, diversos estudos foram feitos no objetivo agregar dados de redes sociais e exibir rapidamente, em uma interface amigável, as informações que se esteja buscando. Um exemplo já clássico, é o trabalho de [30], que informava em tempo real o deslocamento de um tufão ao longo das ilhas japonesas, baseado nos tweets feitos a respeito do evento.

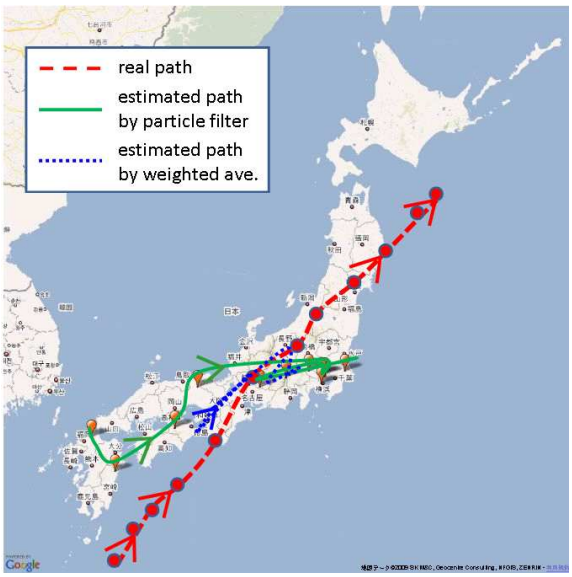

Figura 1.1: Trajeto de um tufão obtido por redes sociais.
Extraído de [30]

Para tal, uma estratégia eficaz é a leitura de informações postadas em redes sociais, como o *Twitter* ou *Facebook* [6]. Contudo, tal estratégia esbarra na enorme quantidade de informações dos mais diversos tipos em tais redes [33]. Sendo assim, faz-se imperativa uma forma de diferenciar os textos entre aqueles que importam e os que não importam para o contexto de trânsito.

Redes sociais apresentam características bastante particulares, como textos pouco formatados, altamente coloquiais, com grande quantidade de onomatopeias, abreviações e *emojis*. Particularmente no caso dos microblogs, como o *Twitter*[3] e o *Sina Weibo*[4], características específicas incluem textos curtos e bastante diretos. O *Twitter*, em específico, é muito usado como fonte para estudos envolvendo redes sociais por sua natureza aberta, visto que suas postagens são públicas por definição [40], diferentemente de redes como o *Facebook*, que exigem permissão explícita para o acesso a postagens. No entanto, estudos

[3]https://twitter.com/
[4]https://www.weibo.com/

focados em classificação de microtextos raramente trabalham com português, sendo eles em sua maioria concentrados na língua inglesa [38]. Igualmente, há grande dificuldade em levantar bases de dados prontas de tweets em português para utilizar em pesquisas desse gênero.

Assim, decidiu-se realizar esse estudo, aplicando técnicas de deep learning ao reconhecimento de informações em *tweets* escritos em língua portuguesa. Nosso trabalho foi particularmente influenciado pela pesquisa publicada em 2019 por Dabiri e Heaslip [4], que propõe metodologias muito similares às propostas ao longo da pesquisa e cujos avanços foram prontamente incorporados.

1.2 Formulação do Problema

Textos de redes sociais demandam uma análise bastante particular para serem classificados, devido às suas características intrínsecas, tais como pouca formalidade, uso de abreviações, alto uso de *links*, *emojis*, onomatopeias e termos com significado menos perceptível, omissão de termos menos significantes, como preposições e artigos, e heterogeneidade. O uso de técnicas de *Deep Learning* para tal tarefa vem sendo feito com sucesso nos últimos anos, no entanto, ainda restam diversas questões particulares a serem respondidas, em especial, se os classificadores que atendem bem ao idioma Inglês serão igualmente satisfatórios em outras línguas. Particularmente no domínio de trânsito, foco da equipe de pesquisa, temos detalhes a considerar, como uso de entidades nomeadas para nomenclatura de pontos de interesse, terminologias próprias, e necessidade de tratamento ágil para atender situações em tempo real.

Da mesma forma é bem pouco explorada a questão se tais classificadores se beneficiam de uma entrada com maior informação dimensional, ou seja se ao transformar os textos em vetores numéricos para inseri-los nas redes, esses vetores serão melhor compreendidos pelo sistema caso tenham mais dimensões.

1.3 Justificativa

Na revisão bibliográfica realizada, foi possível observar a existência de diversos trabalhos envolvendo a extração de informações sobre trânsito a partir de textos de redes sociais. Contudo, poucos desses trabalhos envolvem o uso de *deep learning*, e igualmente poucos lidam com textos em língua portuguesa (ou, a bem da verdade, com idiomas di-

ferentes do inglês e mandarim). Não pudemos encontrar nenhum trabalho que envolva os três aspectos simultaneamente, isto é, aprendizagem profunda, língua portuguesa e domínio de trânsito.

Analogamente, não encontramos trabalhos detalhando a sensibilidade dos classificadores à variação de dimensionalidade da entrada. De fato, em todos os trabalhos de linguagem natural usando *deep learning* que foram levantados, a entrada é vetorizada de uma única forma, com uma única quantidade de dimensões. Assim, o questionamento do quanto uma classificação melhora ou piora baseado no número de dimensões da entrada permanece em aberto.

1.4 Objetivo

Este trabalho tem por objetivo estudar o desempenho de classificadores baseados em *deep learning* para microtextos de língua portuguesa no domínio de trânsito. Adicionalmente pretendemos analisar o quão influente é a dimensionalidade da entrada em tais classificadores. Para tal finalidade, são apresentadas análises de tempo de vetorização e treinamento das redes, bem como de métricas tradicionais de avaliação (acurácia, precisão e *recall*). De modo a estudar melhor as variações envolvidas, são utilizadas diversas combinações de vetorizadores e classificadores, bem como variações tanto na aplicação direta da vetorização em múltiplas dimensões, quanto na aplicação de redutores dimensionais sobre os mesmos, analisando o resultado de tal redução.

1.5 Contribuições

Como visto nas seções anteriores, existem desafios não explorados na aplicação de *deep learning* à classificação de microtextos, de modo a se obter a melhor combinação entre resultados adequados e tempo de uso aceitável. Assim, elencamos as seguintes contribuições no trabalho:

- Estudo dos resultados de classificação de microtextos em língua portuguesa, especificamente no domínio de trânsito, avaliando o desempenho da combinação de três arquiteturas diferentes de classificação (Redes convolucionais, redes recorrentes e a combinação de ambas) com três ferramentas diferentes de vetorização (Word2Vec, GloVe e fastText);
- Estudo do impacto da variação de dimensões, através da análise da performance e

tempo dos classificadores mediante cinco possibilidades dimensionais dos vetorizadores (50, 100, 300, 600 e 1000 dimensões)

- Estudo comparativo do resultado das classificações utilizando vetorizadores em 300 dimensões versus o uso de vetorizadores de 600 e 1000 dimensões cujo resultado é submetido à redução dimensional via PCA para as mesmas 300 dimensões, também nas métricas de tempo e performance; e
- Adicionalmente, o conjunto de textos reunido para os testes dessa pesquisa é disponibilizado gratuitamente para quaisquer interessados em pesquisas com tweets em língua portuguesa.

A pesquisa realizada apresenta como frutos dois artigos, um apresentado no 10º Simpósio Brasileiro de Engenharia de Sistemas Computacionais (SBESC), em novembro de 2020 [36], e outro submetido para o IEEE International Conference on Communications e atualmente em avaliação.

1.6 Estrutura da Dissertação

O Capítulo 2 explora as tecnologias envolvidas no trabalho apresentado. Dentre tais tecnologias são estudadas as ferramentas para representação de textos em forma de vetores numéricos e as ferramentas para análise dos mesmos textos com a finalidade de classificá-los quanto à sua relevância para o tema de eventos de trânsito, com foco particular nas técnicas de Deep Learning.

O Capítulo 3 trata do estado da arte das pesquisas sobre extração de informações de textos em Redes sociais, com foco em eventos de trânsito, apresentando os principais artigos levantados, publicados desde 2017 e avaliando suas metodologias, pontos fortes e áreas nas quais deixam a desejar.

No Capítulo 4 falamos sobre as metodologias empregada, em um sistema de quatro passos (entrada e pré-processamento, vetorização, classificação e saída) e apontando em alto nível as decisões do algoritmo seguido.

O Capítulo 5 apresenta a implementação metodológica, em termos de configurações, variáveis e fórmulas envolvidas, bem como os experimentos realizados e as interpretações de seus resultados, com a tabulagem dos retornos numéricos e representação gráfica dos comparativos efetuados.

No Capítulo 6 é trazida a conclusão obtida a partir das pesquisas e experimentos e sugestões para a continuidade da pesquisa, em termos de novas tecnologias a estudar e aplicações a desenvolver a partir da metodologia.

2. Processamento de Linguagem Natural

Dá-se o nome de Processamento de Linguagem Natural, ou *Natural Language Processing* (NLP) à disciplina que estuda as interações de sistemas computacionais com linguagens humanas, buscando extrair significados e conhecimento diretamente a partir de textos escritos, com o mínimo de intervenção de um leitor que ofereça interpretação humana [14].

Particularmente a extração de conhecimento, ou seja, a obtenção de insumos para tomada de decisão, a partir das informações contidas nos textos a serem analisados, é uma das principais áreas da NLP e, muito embora múltiplas ferramentas e abordagens existam para tal finalidade, permanece com inúmeras questões em aberto [7].

2.1 Classificação

O primeiro desafio de qualquer extração de conhecimento a partir de um texto é justamente identificar se no texto existem informações relevantes ao domínio, das quais o conhecimento possa ser extraido. Assim, é preciso usar alguma técnica para classificá-los. A forma mais básica (e provavelmente mais segura) seria a leitura por um especialista humano. No entanto, a viabilidade de tal ação é bastante limitada, principalmente em sistemas em tempo real, ou com grande volume de textos. Dessa forma faz-se necessário reconhecer de forma automatizada a presença de informações relevantes.

Naturalmente isso pode ser feito inserindo no sistema um banco de *features* (características) relevantes a serem buscadas no texto. Tal abordagem apresenta graves falhas, como a necessidade de um especialista para apresentar as *features*, a dificuldade de representar e capturar essas características no texto, e as diferenças e evoluções linguísticas que podem tornar as características menos relevantes, ou insuficientes.

Outra forma de tratar o problema é deixar que o próprio sistema computacional iden-

tifique as características a serem analisadas, usando técnicas de Aprendizado de Máquina. Para esse fim, há trabalhos documentados com diversos classificadores, tais como Naïve Bayes, SVM, árvores de decisão e K-Nearest Neighbors [27]. Ao longo da última década, contudo, uma grande evolução tem sido apontada no uso de redes profundas, ou *Deep Learning*, para o tratamento de classificação de textos [8].

2.1.1 Deep Learning

Goodfellow, Bengio e Courville definem *Deep Learning* como uma abordagem à inteligência artificial em que os neurônios representando os diversos conceitos e características avaliados são empilhados uns sobre os outros em múltiplas camadas [8]. Diferente de uma rede neural comum, devido à grande quantidade de neurônios, conexões e camadas, não há a necessidade de apontar diretamente *features* para cada análise, sendo que a rede se torna capaz de aprender a identificá-las e extraí-las por conta própria.

A base de uma rede profunda é a arquitetura conhecida como *Multilayer Perceptron*, que em sua definição mais simples é uma rede composta por multiplas camadas de neurônios em *feed-forward*, ou seja, em que cada neurônio da camada n alimenta todos os neurônios da camada $n+1$ e não recebe nenhuma forma de *feedback*. Um exemplo desse tipo de rede é mostrado na figura 2.1, que apresenta uma arquitetura em cinco camadas, em que cada uma, de acordo com a sua profundidade, vai extraindo características de níveis de abstração incrementalmente maiores da imagem fornecida, começando com o conjunto de *pixels*, e identificando limites, contornos, objetos e, por fim, a identidade dos objetos.

Outras arquiteturas bastante comuns de redes profundas incluem:

- Autoencoders, que agregam, ou melhor, codificam as informações recebidas em pacotes cada vez menores, e os decodificam de modo a recuperar a informação original completa;
- Redes de crenças profundas, ou Deep Belief Networks (DBN), em que cada conexão entre camadas é tratada como uma máquina de Boltzman restrita, sendo treinada separadamente;
- Redes recorrentes, que recebem entradas sequenciais e a cada novo passo dessa sequência usam tanto a entrada quanto a informação acumulada de contexto para a análise total do objeto; e
- Redes convolucionais, são redes que usam entradas em topologia de grade e em que

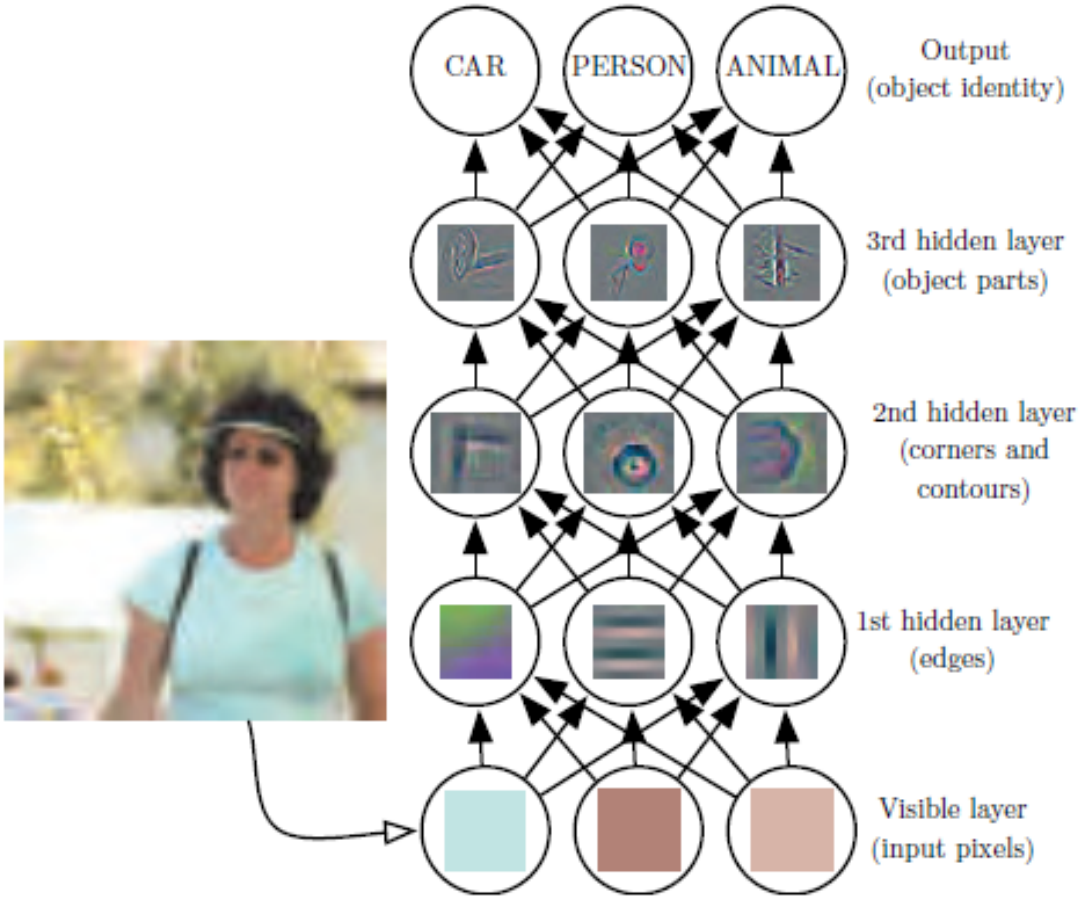

Figura 2.1: Esquema de um *multilayer perceptron*, mostrando as camadas e características obtidas, extraído de [8], originalmente publicado por [43]

.

a passagem entre duas ou mais camadas utiliza uma função convolucional em vez de uma multiplicação matricial, como um filtro de moda ou mapeamento subsequente.

Dentre essas diversas arquiteturas de redes profundas, as propostas em Redes Convolucionais e Redes Recorrentes têm se mostrado particularmente interessantes para a tarefa de classificação de textos [3, 4, 16].

2.1.2 Redes Convolucionais

As Redes Convolucionais, identificadas pela sigla CNN (*Convolutional Neural Networks*), foram propostas em [17] e consistem em uma arquitetura de redes neurais composta por uma secão de extração de *features*, e uma de classificação. O uso de CNN é reconhecido como uma das principais ferramentas para detecção automatizada de features, sendo particularmente usado em processamento de imagens, mas também em processamento de linguagem natural [16].

A primeira seção da rede, seu grande diferencial, é composta por uma sequência de pares de elementos. Uma camada de convolução seguida por uma de concentração (*pooling*). Na camada de convolução, a matriz de entrada é submetida a um filtro, chamado de função de convolução, célula a célula. Tal filtro é aplicado sobre a célula e um conjunto de células adjacentes (formando um quadrado 3×3, 5×5, etc.). A operação corresponde a um produto interno entre o filtro e a região onde está sendo aplicado, de modo que o resultado é um único escalar, que é colocado na célula correspondente na camada de concentração. Dessa forma, as informações de um conjunto de células são concentradas em uma única, possibilitando uma melhor análise do classificador. Tal processo pode ser repetido diversas vezes, adicionando-se pares de camadas de convolução e concentração.

A segunda seção da rede é responsável pela classificação dos dados. Geralmente essa seção é composta por: (i) uma camada de achatamento, que reduz as dimensões das informações obtidas a partir da camada de convolução, e (ii) uma camada de neurônios *fully-connected*, que pontua os elementos e seleciona a classe com maior semelhança aos resultados da rede [19]. Esse processo é mostrado na Figura 2.2, extraída do trabalho de [31] onde dois níveis de convolução são usados para extrair características de uma imagem, as quais são usadas para classificá-la.

2.1.3 Redes Recorrentes

Redes Neurais Recorrentes, ou Recurrent Neural Networks (RNN), são uma arquitetura de deep learning em que o resultado do passo n é usado como entrada para o passo

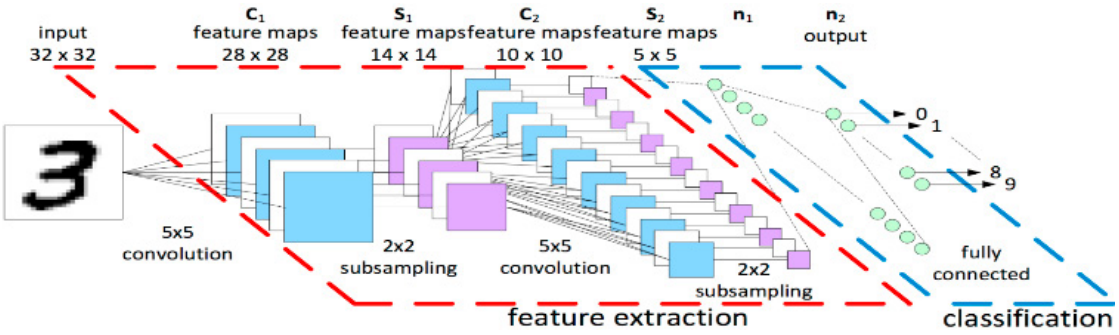

Figura 2.2: Esquema de uma rede convolucional, extraído de [31].

$n+1$. Dessa forma, a camada de recorrência se retro-alimenta, justificando o nome. Essa retroalimentação faz com que a rede possua uma "memória" dos eventos passados, que influenciam nos resultados seguintes. São bastante eficazes no reconhecimento de entidades nomeadas e *speech tagging*, situações em que o contexto carrega informação necessária ao reconhecimento [3].

Uma questão importante sobre a arquitetura RNN tradicional é o fato de que as informações não são "esquecidas", fazendo com que os resultados iniciais influenciem todo o processo. Para abordar essa questão, foi proposta a rede *Long Short-Term Memory Network* (LSTM, ou rede de memória longa de curta-duração) [11]. Nessa arquitetura, a recorrência é composta por quatro camadas de ativação, ou "portões" que a cada novo passo, selecionam se:

1. A informação contida no neurônio deve ser esquecida (*Forget Gate*);
2. A informação recebida deve ser lembrada (*Input Gate*); e
3. A informação recebida deve ser enviada para a função de ativação (*Output Gate*).

A quarta camada é a função de ativação padrão da rede neural, que computa a informação e envia para o classificador. Dessa forma, dependendo das condições, uma informação ao entrar na rede pode demandar que a "memória" seja apagada, não sendo avaliada com base nas informações passadas [8]. A Figura 2.3, extraída de [9], mostra um neurônio típico de uma rede LSTM, com os portões e suas entradas, saídas e comunicações.

2.2 Representação

O uso de tais ferramentas para classificação de textos geralmente demanda a representação dos mesmos em vetores numéricos, uma vez que a maioria dos classificadores,

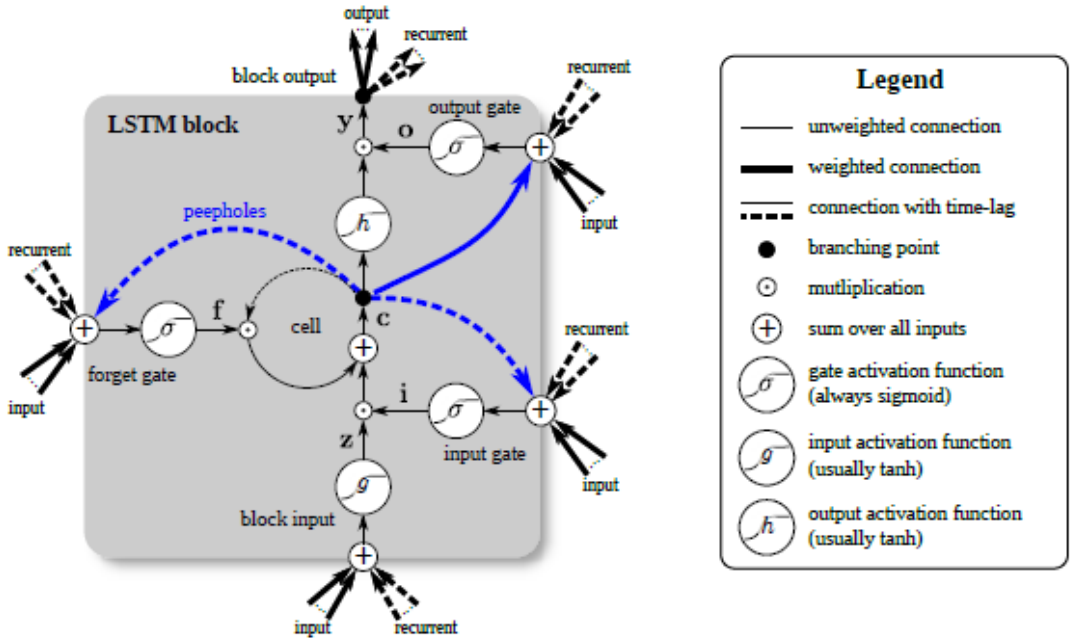

Figura 2.3: Esquema de uma rede LSTM, extraído de [9]

particularmente redes neurais, são focados no trabalho com números. Nas próximas seções, serão brevemente descritas algumas técnicas de representação, desde as mais básicas, como Bag-of-Words e Tf-Idf, até mais avançadas, como aquelas baseadas em *Word Embedding*, tais como *Word2Vec* e *GloVe*.

2.2.1 Bag-of-words

As técnicas básicas de vetorização são coletivamente chamadas de *Bag-of-Words*, por essencialmente armazenarem todas as palavras de um docuemnto em uma "bolsa", desprezando sua posição relativa. O método mais simples consiste em basicamente contar as ocorrências de uma palavra no texto. Nesse sistema, cada palavra presente no *corpus* como um todo é considerada uma dimensão de um vetor, e o seu valor será o número de vezes que a palavra aparece. Assim, um documento é representado por um vetor em um espaço n-dimensional, em que n é o número de palavras no *corpus* [35].

Uma evolução desse método é o chamado *Term Frequency/Inverse document frequency* (Tf-Idf). Nesse método, a freqüência dos termos em seus documentos é normalizada pelo inverso de sua frequência no *corpus*, segundo a fórmula $w_{x,y} = tf_{x,y} * log(\frac{N}{df_x})$. Assim como na contagem básica, esse método tende a gerar vetores esparsos e, apesar de carregar alguma informação semântica, o mesmo não considera as semelhanças entre os termos, além de obviamente ignorar suas posições.

2.2.2 Word Embeddings

Para atacar tais problemas, são usados métodos de *deep learning* para vetorização dos termos, os quais têm por objetivo projetar as palavras em um espaço n-dimensional, através do uso de redes profundas, onde proximidade vetorial se traduza em proximidade semântica. Esses sistemas são coletivamente chamados de *Word Embeddings*.

Word2Vec foi uma das primeiras propostas de *Word Embeddings*, desenvolvidas por um grupo de cientistas da Google liderados por Tomas Mikolov [21], e compreende duas implementações distintas do conceito. A primeira, *Continuous Bag-Of-Words*, ou CBOW, recebe uma sequência de palavras, buscando prever uma palavra (geralmente de posição central) do grupo, a partir das outras e, no processo, gera sua a representação vetorial. A outra implementação, *Skip-Gram*, propõe o oposto, tentando obter o contexto a partir da palavra analisada. Em ambos os casos, o conjunto de palavras é submetido a um *AutoEncoder*, e o resultado da rede é comparado com o gabarito, utilizando uma função de *back-propagation* para aprender os padrões.

A implementação *fastText* foi desenvolvida por uma equipe ligada ao Facebook, também liderada por Mikolov [22], e é de modo geral uma evolução do Word2Vec. Seu funcionamento é bastante similar à proposta anterior, sendo também implementada através das técnicas CBOW e SkipGram. Contudo, o principal diferencial é que a rede não trata as palavras como unidades atômicas, mas sim como sequências de letras, e as analisa através de comparações de subconjuntos de letras. Por exemplo, o termo fastText pode ser visto como um conjunto de {'fas', 'ast', 'stT', 'tTe', 'Tex', 'ext'}, e dessa forma carregar significados próximos de *fast* ({'fas', 'ast'}) e *text* ({'tex', 'ext'}). Assim, mesmo uma palavra com pouca presença nos corpora usados para treinar a rede poderá ser traduzida pelas suas similaridades internas.

Global Vectors, ou *GloVe*, são uma terceira implementação de *Word Embeddings*, proposta por uma equipe da universidade de Stanford [25]. Essa implementação identifica as palavras por meio de uma matriz de co-ocorrência, que avalia a probabilidade de duas palavras aparecerem no mesmo texto. Seus resultados são similares ao Word2Vec, embora com uma distribuição vetorial mais abrangente.

2.3 Dimensionalidade

Um aspecto importante da representação vetorial é a dimensionalidade dos vetores resultantes. Como através do uso de *Word Embeddings*, é possível traduzir as palavras para

vetores em qualquer espaço n-dimensional (respeitando, é claro, a capacidade computacional do sistema usado), é necessário atentar para a quantidade de dimensões. De fato, o aumento do espaço dimensional tende a levar a um aumento quadrático ou mesmo exponencial dos recursos computacionais necessários para o treinamento das redes neurais, problema esse tão conhecido que é apelidado "Maldição da Dimensionalidade" (*Curse of Dimensionality*). [13].

Em alguns contextos, como na Estatística e no Aprendizado de Máquina, como forma de mitigar esse problema, foram criadas diversas técnicas para reduzir o espaço dimensional dos dados sem perder informação significativa, dentre as quais, uma das mais simples e performáticas é a Análise de Componente Principal, ou PCA (Principal Component Analysis) [41].

2.3.1 Análise de Componente Principal

A técnica de PCA foi proposta em 1901, por Karl Pearson [24], no escopo de análises estatísticas. O cerne do modelo é a projeção dos dados em um subespaço linear de menor dimensionalidade, capturando as dimensões que melhor explicam a variância do dado. Tal processo é feito por meio do cálculo da matriz de covariância do dado, cujos n autovetores associados aos n maiores autovalores carregam a maior semântica do dado. Baseado nisso, os n autovetores correspondentes a esses n autovalores serão a base do novo espaço vetorial n-dimensional [41].

3. Trabalhos Relacionados

Neste capítulo, apresentamos alguns trabalhos elencados em nossa pesquisa bibliográfica, de modo a determinar o estado da arte de trabalhos envolvendo a obtenção de informações de trânsito a partir de redes sociais, fazendo ou não uso de Aprendizado de Máquina para cumprir esse objetivo.

Em 2017, Wang et tal. publicaram um estudo sobre a combinação de tweets com informações de GPS e mapeamento urbano e climático [42]. O trabalho relata a construção de um mapa de congestionamentos na cidade Chicago, a partir de cerca de dois milhões e meio de dados de GPS carregando informações de instante temporal, velocidade e localização. Tais pontos geram manchas de congestionamento a partir de análises de velocidade, associados às posições temporais para determinar início e fim dos eventos. Os tweets são por sua vez, obtidos por meio de perfis de autoridades públicas e usuários focados em informações de trânsito, sendo eles bem formatados e padronizados. Além destes, tweets do público geral foram detectados por meio de palavras-chave e adicionados ao banco de dados. Os posts foram, então, associados a partir de suas posições espaço-temporais aos locais de congestionamento, servindo como explicação das paralisações, como ilustrado na figura 3.1. Um ponto a ser observado é que os tweets apenas trazem informações adicionais a congestionamentos conhecidos, não sendo o sistema capaz de detectar outros tipos de evento, nem usar a rede social como fonte primária.

No ano seguinte, Zheng et al. apresentam um framework com o mesmo propósito de extrair informações de posts do Weibo e associá-las a eventos anômalos de trânsito [44]. Os autores usam dados de GPS instalados em taxi da região de Shenzen na China, coletados em outubro de 2014 e projetados sobre mapas da rede de transporte urbano. Dados considerados anômalos, no caso, com velocidades abaixo da média local ou tempos de deslocamento acima da média são mantidos, enquanto o resto é descartado. Eventos no Weibo são detectados geocodificando entidades nomeadas e pareando-as espaço-temporalmente com as anomalias de deslocamento dos taxis. Isso significa que, diferente

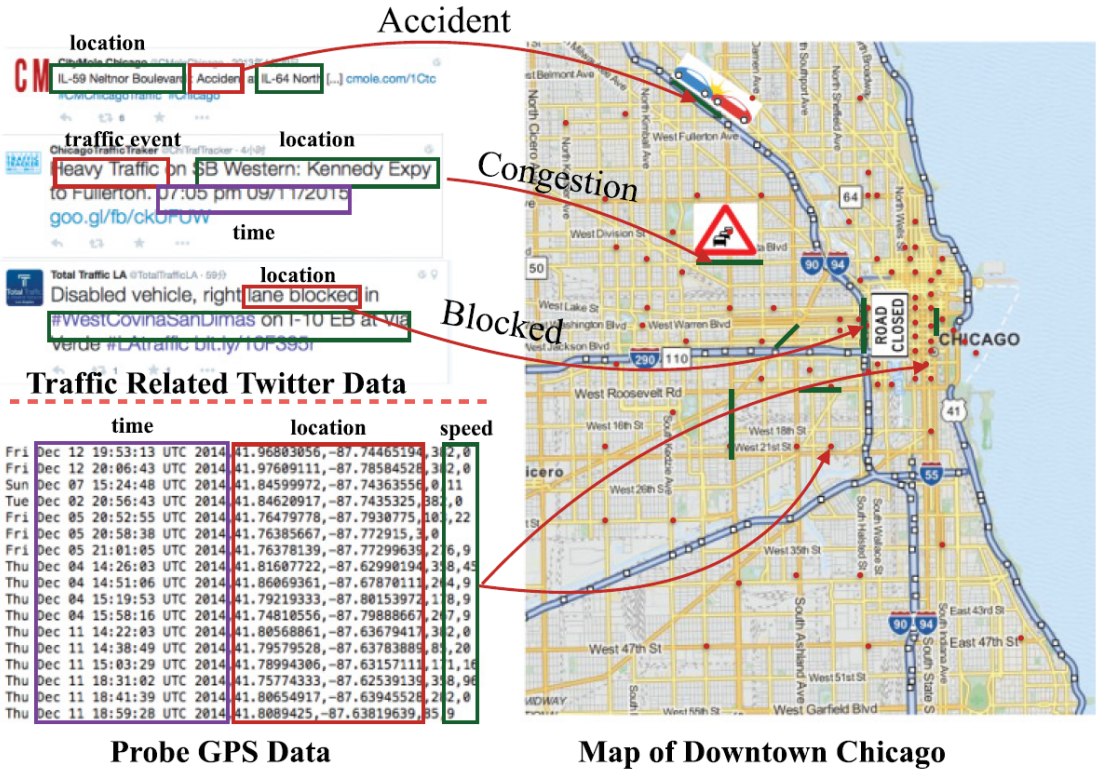

Figura 3.1: Exemplo de associação de tweets a congestionamentos. Extraído de [42]

do trabalho anterior, é possível detectar diversos tipos de eventos, embora a fonte primária das informações não seja o Weibo. Além disso, cabe mencionar a ser mencionado é que o uso de entidades nomeadas é limitado a localizações a geocodificar, significando que o sistema não depende de palavras-chave para identificação de eventos.

Em 2019 diversos trabalhos na área foram publicados, como o de Rettore et al., numa parceria entre pesquisadores da UFMG e UNICAMP, que constrói a aplicação T-Incident, capaz de unir dados veiculares com mídias sociais com o propósito de mapear eventos relevantes relacionados ao trânsito, como shows, acidentes, locais para hospedagem, etc [29]. Com uma abordagem diferente dos trabalhos anteriormente citados, a pesquisa partiu de incidentes relatados nas plataformas Here WeGo[1] e Bing Maps[2] e vinculando informações de tweets na mesma área espaço-temporal. Assim, o trabalho é incapaz de detectar um evento a partir de um tweet, necessitando que o sistema já o conheça das plataformas de mapas. Além disso, o uso de palavras-chave para classificar a relevância do post apresenta uma dificuldade com relação a erros de digitação e palavras desconhecidas.

No mesmo ano, Tenório et al., uma equipe conjunta da UFMG com a UFAL propôs a detecção de eventos no Twitter através do uso de grafos de visibilidade natural [37]. A

[1]https://wego.here.com/
[2]https://www.bing.com/maps

partir de datasets pré-classificados de tweets focados no campeonato de futebol inglês e nas eleições norte-americanas, o sistema identifica eventos nesses contextos a partir de da distribuição de Poisson dos bigramas mais comuns, sem necessidade de pré-definir palavras-chave. Esses eventos são, então, agrupados por meio de grafos de visibilidade, em que os vértices são palavras e as arestas são conexões formando os bigramas, e particionados por um processo de clusterização Markoviano. Como o sistema utilizou tweets pré-classificados, aparentemente não seria adequado para detectar eventos em um conjunto real não temático de postagens em tempo real, embora sua combinação com um classificador ágil poderia resolver tal pendência.

Ainda em 2019, a equipe de Cidades Inteligentes do PPGI/UNIRIO publicou o trabalho de Leonardo Tetéo e Elton Soares [39], utilizando um framework para detectar eventos de trânsito em tempo real, baseado no modelo estatístico Conditional Random Fields, que modela entidades nomeadas na forma de grafos. Enquanto que boa parte do código e todo o dataset tenham ficado inacessíveis após o desligamento de membros da equipe, a pesquisa seguiu com a proposta já feita à época de usar deep learning para classificação dos tweets em tempo real, de modo que possam ser aplicados a um sistema de georreferenciamento e cadastro de eventos.

Finalmente, no mesmo ano, Dabiri e Heaslip, na Universidade de Virgina, propuseram um modelo para classificar relevância de eventos de trânsito em tweets, por meio de deep learning [4]. Sua proposta se assemelha à da nossa equipe de tal forma que optamos por incorporar a sua metodologia em nossa pesquisa, aplicando o algoritmo usado à língua portuguesa e testando os limites e possibilidades ao redor do sistema pronto.

Já em 2020, Shi et al. [32] propuseram uma rede baseada no modelo CNN, chamada *Wide-grained capsule network* para detecção de eventos meteorológicos a partir de posts no Sina Weibo. A rede desenvolvida parte de uma camada convolucional baseada em *N-gramas* (ao contrário da maioria dos trabalhos, que usam *Bag-of-words*), conectado a uma sequência de camadas de encapsulamento, além de uma segunda camada convolucional. Além disso, outro ponto a ser observado é o uso do sistema Google BERT para representação vetorial dos posts. Devido à novidade do trabalho, assim como as outras pesquisas de 2020, não pudemos estudá-lo mais a fundo para determinar como poderia ser usado para melhorar nossa pesquisa, ainda que não pertença ao domínio de trânsito.

Também em 2020, Dabiri et al. [5] apresentaram um modelo para classificação de veículos por uma combinação de tamanho e peso, baseado na sua trajetória com o uso de GPS, aplicando um modelo de redes convolucionais sobre os dados de deslocamento. Enquanto que a pesquisa em si não envolve redes sociais, é mais um degrau no uso de

deep learning para modelagem de trânsito.

Além disso, Bencke et al. [1], uma equipe combinada da UFSC com a Universidad de Valparaíso, propuseram em 2020 um sistema para classificar posts do *Twitter* e do *Colab.re*[3] com respeito a áreas de *Smart Cities* em que possam se encaixar. O trabalho envolveu o uso de técnicas clássicas de *machine learning* para a execução da classificação, e ocorreu muito mais como uma prova de conceito da recuperação de informações de ambas as redes do que de um algoritmo focado em tal extração.

Finalmente, Li et al. [18] trouxeram o uso de redes adversariais generativas (GAN, ou *Generated Adversarial Networks*) combinadas com redes convolucionais para executar detecções de modos de viagem sobre dados de GPS, obtendo bons resultados e levantando a possibilidade do uso de GAN para outras pesquisas na área. As redes adversariais foram usadas primariamente para melhorar a qualidade das amostras de GPS, sendo a classificação executada via CNN, contudo.

[3] https://www.colab.re/

4. Proposta de classificação de tweets de trânsito em português

4.1 Problema

4.1.1 Classificação por Deep Learning

4.1.2 Características do idioma português

4.2 Metodologia

De modo a tratar o problema de classificação dos microtextos, seguimos uma metodologia fundamentada nos conceitos apresentados, de Word Embeddings e Deep Learning. Tal modelo é diretamente baseado no trabalho proposto por [4], e enquanto que os experimentos envolvem modificações e análises não apresentadas pelos pesquisadores citados, a metodologia central é a mesma.

O modelo é composto por cinco etapas, detalhadas nas seções abaixo. A figura 4.1 mostra a sequencia de passos de forma visual:

1. 4.2.1 Entrada e Pré-processamento
2. 4.2.2 Vetorização
3. 4.2.2.1 Redução de Dimensionalidade
4. 4.2.3 Classificação
5. 4.2.4 Saída

Figura 4.1: Sequência de etapas do modelo proposto para a representação e classificação de microtextos.

4.2.1 Entrada e pré-processamento

O sistema recebe como entrada um tweet por vez, contendo a id e o texto. O texto sofre um primeiro pré-processamento através da remoção de caracteres especiais, conversão em caixa baixa e transformação do texto em uma lista de tokens. Após esse passo, são removidas todas as palavras reconhecidas como stopwords, ou seja, palavras destituídas de semântica relevante, tais como artigos, preposições e verbos de ligação. Ao final desse passo, obtemos um conjunto de palavras semanticamente significantes, que é enviada para o vetorizador.

4.2.2 Vetorização

O passo seguinte transforma o conjunto de tokens em um conjunto de vetores *n*-dimensionais, o número de dimensões do vetorizador escolhido. O processo de vetorização não pode ser feito em tempo real, uma vez que a cada novo conjunto de tokens, a semântica seria alterada pelas mudanças de contexto, levando a tokens de mesmo significado possuírem representação vetorial diferente caso sejam inseridos em contextos diferentes. Assim o sistema deve fazer uso de um dicionário pré-montado, onde cada palavra é traduzida para o vetor correspondente, ou para um vetor de zeros, caso não seja encontrada.

4.2.2.1 Redução de Dimensionalidade

Como uma análise adicional, foi proposta uma redução de dimensionalidade, de modo a avaliar a sensibilidade dos classificadores a esse tipo de ferramenta. Redutores de dimensionalidade operam pelo princípio de condensar informações em menos dimensões, como um Autoencoder, ou descartar dimensões com baixa semântica, como um PCA, tal como visto na seção 2.3. Assim como na vetorização, optou-se pela aplicação antecipada dos redutores, uma vez que não faria sentido aplicação em tempo real da análise sobre um dicionário estático.

Tal redução sobre *word embeddings* pré-vetorizados foi proposta por Raunak et al. em 2019 [28]. Seu algoritmo, aplicado no trabalho, consiste em uma combinação do

PCA clássico com o algoritmo de pós-processamento (PPA, *Post-Processing Algorithm*) proposto por Mu e Viswanath [23], sendo este mostrado no algoritmo 1 e a redução completa no algoritmo 2.

Algorithm 1 Algoritmo de pós-processamento (adaptado de [28])

Dados: Matriz de Word Embedding X, Parâmetro de Limiar D
Resultado: Matriz de Word Embedding Pós-processada X

function PPA(X, D)
    $X \leftarrow X - \overline{X}$ ▷ Subtrair a média da matriz
    $u_i \leftarrow PCA(X)$, sendo $i = 1, 2, ..., D$ ▷ Computar os principais componentes via PCA
    for all v in X **do** ▷ Remover os D componentes mais relevantes
        $v = v - \sum_{i=1}^{D} (u_i^T \cdot v) u_i$
    return X

Algorithm 2 Algoritmo de redução dimensional (adaptado de [28])

Dados: Matriz de Word Embedding X, Nova Dimensão N, Parâmetro de Limiar D
Resultado: Matriz de Word Embedding com N Dimensões X

$X \leftarrow PPA(X, D)$ ▷ Aplicar algoritmo PPA
$X \leftarrow PCA(X)$ ▷ Transformar X usando PCA
$X \leftarrow PPA(X, D)$ ▷ Aplicar algoritmo PPA

A ideia por trás do algoritmo é de que, segundo Mu e Viswatch [23], todos os principais *word embeddings* apresentam um vetor médio responsável pela maior parte de sua variância e, após a sua remoção, concentram sua energia em um espaço de em média 8 dimensões. Assim, a remoção de ambos fortaleceria o *embedding*, espalhando sua variância pelo resto do espaço dimensional. Baseado nessa proposta, Raunak et al. propõem a utilização do PPA como um "purificador"da representação, antes e depois da redução via PCA, garantindo que o *embedding* obtido possua uma variância espalhada por seu espaço dimensional e possa carregar melhor suas informações semânticas.

4.2.3 Classificação

Os textos vetorizados foram classificados segundo o algoritmo de [4], utilizando as três implementações propostas: CNN pura, LSTM pura e ambas as redes combinadas. Tal combinação é proposta como uma complementação entre as características das redes, a excelente capacidade de extração de características do CNN e a análise de contexto do LSTM, tratando as frases como sequências de tokens. Tal abordagem vem sendo utilizada com maior frequência em outros trabalhos de NLP [3, 16].

Optou-se por seguir as configurações propostas por [4], no tangente a tamanho de n-grama, algoritmo de otimização, e outros hiper-parâmetros, os quais serão melhor explicados no Capítulo 5. Apesar disso, houve alterações significativas no truncamento de tokens e adição de novas variáveis a serem estudadas, também explicadas no referido capítulo.

Especificamente sobre a combinação de redes, o processamento é feito em sequencia por uma CNN e uma LSTM, sem o uso de camada de acumulação entre elas, como mostrado na Figura 4.2, a qual retrata os conjuntos de tokens em suas respectivas janelas, sendo tratados pela função de convolução e compondo a recorrência, sendo cada janela, um passo.

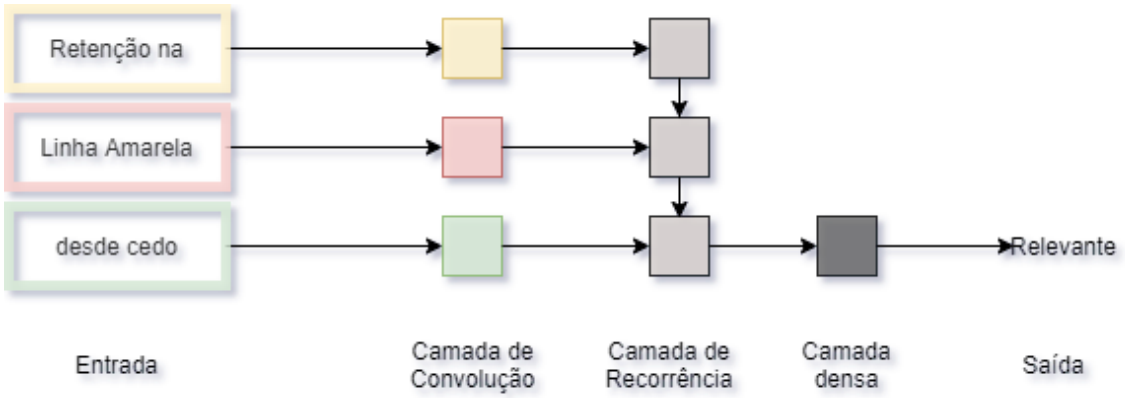

Figura 4.2: Modelo CNN + LSTM

4.2.4 Saída

Os resultados das redes neurais são submetidos a um classificador softmax, que combina os diversos pontos levantados em um único score e aponta de forma unívoca a classe mais adequada ao texto. A saída final do sistema é, assim, a classificação binária do texto entre relevante e não-relevante, de modo a avaliar se tal informação pode ou não ser utilizada em um sistema online de informações de trânsito.

5. Experimentos e Resultados

Buscando analisar o modelo proposto, foram feitos uma série de experimentos relativos à extração de conhecimento em microtextos. Este capítulo descreve tais experiências e os resultados obtidos.

5.1 Ambiente

Para os experimentos, foi usado um sistema Windows 10, montado sobre um hardware composto por

- Processador Intel Core i5-7400 (7ª Geração), com 4 núcleos e velocidade de 3 GHz
- 16 GB de RAM DDR4 Dual Channel, disposta em 2 pares de pentes
- SSD para armazenamento do sistema operacional, ambiente programacional e swap
- HD SATA3 para armazenamento dos dados e modelos analisados

O sistema também inclui uma placa gráfica, mas que não foi utilizada nos experimentos. Com respeito a software, foi usado um ambiente Python 3.7 implementado via Anaconda 4.8 e isolado através de Virtual Environment. Foi usado o seguinte conjunto de pacotes Python (todos os quais disponíveis no repositório PyPi):

- gensim 3.8.0
- Keras 2.2.5
- matplotlib 3.1.2
- nltk 3.4.5

- numpy 1.17.1
- python-decouple 3.3
- sklearn 0.21.3
- tensorflow 1.15.2

5.2 Modelos utilizados

As vetorizações usadas foram obtidas do Repositório de Word Embeddings do NILC, cujo trabalho é detalhadamente relatado em [10]. Para os experimentos, foram usadas as vetorizações criadas em Word2Vec, fastText (ambas na implementação SkipGRAM) e GloVe, geradas pelo NILC para o idioma português. Tais vetorizações são disponibilizadas gratuitamente no próprio site do laboratório [1], como arquivos de texto organizado por vírgulas (CSV). Os modelos foram disponibilizados como vetores *n-dimensionais* cujos valores nas *n* dimensões variam entre -1 e 1. Os valores de n usados foram:

- 50
- 100
- 300
- 600
- 1000

As vetorizações foram produzidas baseadas em um *corpus* de diversas fontes, incluindo Wikipedia, Google News, obras literárias de domínio público, jornais e textos científicos [10].

As redes neurais de classificação usados (CNN e LSTM) foram montadas através do Keras, utilizando algoritmos baseados nos propostos por Dabiri e Heaslip [4].

5.3 Coleta e etiquetamento do Dataset

Para os experimentos, foi coletado um dataset composto por tweets cuja tag *languages* equivalia a pt-BR (Portanto definidos como português brasileiro), e filtrados por um re-

[1] http://nilc.icmc.usp.br/nilc/index.php/repositorio-de-word-embeddings-do-nilc

tângulo envolvente correspondente às latitudes 5,88 N e 34,31 S e longitudes 34,97 W e 74,62 W. Tal área, como visto na figura 5.1 cobre praticamente todo o território brasileiro.

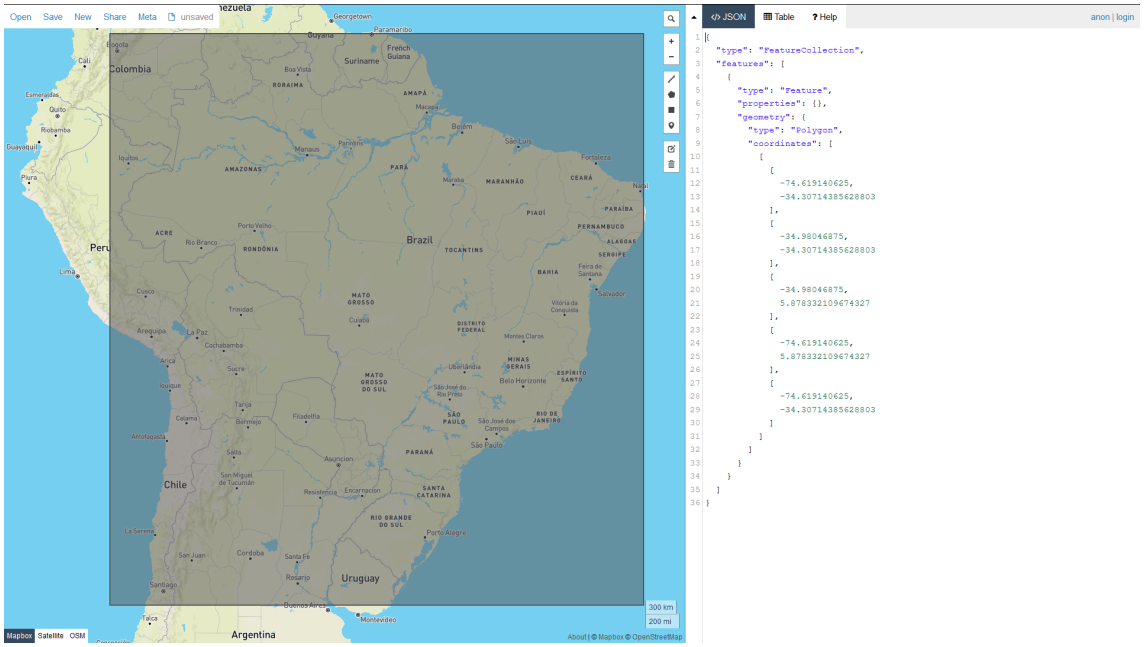

Figura 5.1: Retângulo envolvente utilizado (Representação gerada no site `http://geojson.io/`)

Além destes filtros na API de coleta, o sistema também foi programado para descartar *tweets* que fossem compostos por menos de quatro palavras, após o descarte de sinais de pontuação, quebras de linha, urls e *stopwords*. Dentre os *tweets* obtidos, 18.793 foram coletados do histórico postado por perfis tradicionalmente vinculados ao domínio de trânsito até 13 de março de 2020, sendo estes: @OperacoesRio, @CETRIO_ONLINE, @transitorj, @CETSP_, @TransitoSampaSP e @radiotransitofm. Já os outros 24.359 foram obtidos pelo método *streaming*, sendo tweets postados em 14 de março de 2020 que atendiam aos filtros supracitados.

Os textos obtidos foram classificados manualmente, após uma pré-classificação por meio de um dicionário das palavras mais frequentes nos *tweets* dos perfis @CETRIO_ONLINE e @CETSP_. Dessa forma, chegou-se a um conjunto de 16.315 *tweets* relevantes ao domínio de trânsito, e outros 26.837 não-relevantes. O *dataset* foi então dividido aleatoriamente em conjuntos de treino e teste, através do uso da ferramenta *scikit-learn*, respeitando uma proporção de 4:1 para treino e teste, respectivamente.

5.4 Processamento

5.4.1 Pré-processamento

O pré-processamento fez uso de funções nativas da linguagem *Python*, bem como da lista de *stopwords* em português provida pelo pacote NLTK [2].

Tais vetores são truncados no 30º token, levando a um *input* para o classificador de uma matriz 30 vezes o número de dimensões do vetorizador.

Tal truncamento foi feito de modo a garantir um *input* de tamanho fixo, necessário para as redes implementadas no pacote *Keras*. A escolha do número 30 se deu mediante uma análise do número de palavras dos *tweets*, sendo que tal número contempla 98.5% dos textos, após a remoção das *stopwords* e *urls*, como mostrado na Figura 5.3, sem causar *overflows* de memória.

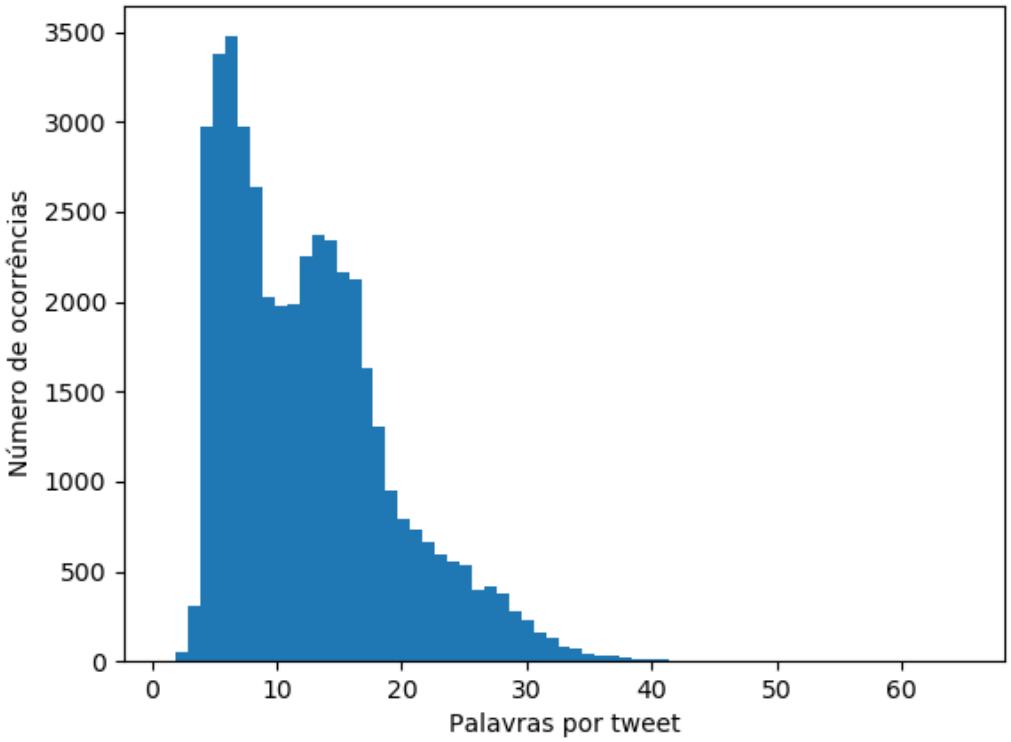

Figura 5.2: Histograma da quantidade de palavras

5.4.2 Vetorização

A representação vetorial foi gerada usando um sistema de pré-vetorização, onde uma análise prévia de um corpus similar (porém não necessariamente o mesmo, e nesse caso

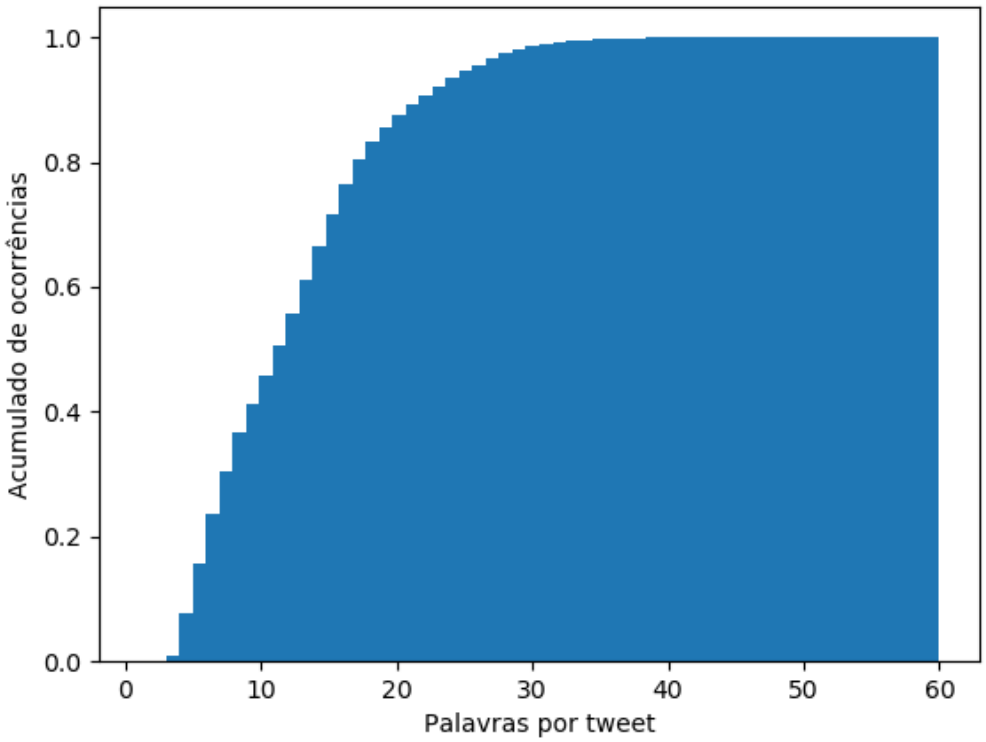

Figura 5.3: Distribuição cumulativa da quantidade de palavras

realmente não o mesmo) é analisado para a obtenção de um dicionário palavra-vetor. O pseudo-código é mostrado no Algoritmo 3.

Algorithm 3 Algoritmo de vetorização

```
1:  function VECTORIZE(tokenLimit, text, dictionary)
2:      vector ← zeroVector()
3:      for token ∈ text do
4:          index ← getIndexOfToken(token, text)
5:          if index < tokenLimit then
6:              if token ∈ dictionary then
7:                  vector[index] ← dictionary[token]        ▷ Se o token não constar do
    dicionário, não precisa fazer nada, o vetor já vem zerado por default
8:          else
9:              break        ▷ Se o index não for menor que o limite de tokens, sai do loop
10:     return vector
```

No caso desta pesquisa, foram utilizadas vetorizações executadas pelo Núcleo Interinstitucional de Linguística Computacional da Universidade Federal de São Paulo (NILC-

USP), documentado por Hartmann et al em [10]. Tal vetorização foi feita sobre os corpora elencados na tabela 5.1, traduzida do artigo supracitado.

Foram usados os dicionários de Word2Vec/SkipGram, GloVe e fastText, que foram adaptados para a língua portuguesa e disponibilizados pelos autores[2]. Os dicionários apresentam vetorizações em 50, 100, 300, 600 e 1000 dimensões, variável essa usada na pesquisa para determinar o conjunto mais adequado de dimensões.

5.4.2.1 Redução dimensional

Como mencionado na seção 2.3, trabalhar com dimensões muito elevadas pode ser extremamente pesado sobre os recursos. Específicamente no caso de word embeddings pré-vetorizados, como o que trabalhamos nessa pesquisa, Raunak et al. [28] propuseram uma técnica

FALTOU TUDO!

5.4.3 Classificação

Foram feitos experimentos utilizando os três modelos propostos por [4] (CNN, LSTM e o modelo combinado). Os códigos utilizados encontram-se no repositório GitHub[3].

5.4.3.1 CNN

O modelo de CNN foi implementado como no Algoritmo 4.

Algorithm 4 Algoritmo de configuração da CNN

```
1:  function CNN(tweets, dimensions)
2:      vectorLength ← tweets.length()
3:      model ← Sequential()
4:      model.add(Conv2D(filters = 200, kernel_size = (2, dimensions), activation = relu)
5:      model.add(MaxPooling2D(pool_size = convolution_output)
6:      model.add(Dropout(rate = 0.5))
7:      model.add(Flatten())
8:      model.add(Dense(units = 2, activation = softmax))
9:      model.compile(optimizer = Adam, loss = categorical_crossentropy)
10:     return model
```

O primeiro passo do algoritmo é a própria função de convolução, configurada com os seguintes parâmetros:

- Quantidade de filtros: 200

[2] http://nilc.icmc.usp.br/nilc/index.php/repositorio-de-word-embeddings-do-nilc
[3] https://github.com/EBarbara/twitter-research

Tabela 5.1: Estatísticas dos Corpora usados na vetorização. Adaptado de [10]

Corpus	Tokens	Tipos	Gênero	Descrição
LX-Corpus [Rodrigues et al. 2016]	714,286,638	2,605,393	Gêneros mistos	Uma enorme coleção de textos de 19 fontes. A maior parte dos quais escritos em Português Europeu.
Wikipedia	219,293,003	1,758,191	Enciclopédico	Dump da Wikipedia em 20/10/2016
GoogleNews	160,396,456	664,320	Informativo	Notícias raspadas do serviço GoogleNews
SubIMDB-PT	129,975,149	500,302	Linguagem falada	Legendas raspadas do website IMDb
G1	105,341,070	392,635	Informativo	Notícias raspadas do portal G1 entre 2014 e 2015.
PLN-Br [Bruckschen et al. 2008]	31,196,395	259,762	Informativo	Grande corpus do Projeto PLN-BR com textos amostrados de 1994 a 2005.
Obras literárias de domínio público	23,750,521	381,697	Prosa	Uma coleção de 138,268 obras literárias do website Domínio Público
Lacio-web [Aluísio et al. 2003]	8,962,718	196,077	Gêneros mistos	Textos de diversos gêneros, como literários, informativos, científicos, legais e didáticos
Portuguese e-books	1,299,008	66,706	Prosa	Coleção de livros clássicos de ficção escritos em Português do Brasil raspado do website Literatura Brasileira
Mundo Estranho	1,047,108	55,000	Informativo	Textos raspados da revista Mundo Estranho
CHC	941,032	36,522	Informativo	Textos raspados do website Ciência Hoje das Crianças(CHC)
FAPESP	499,008	31,746	Comunicação Científica	Textos de divulgação científica da revista Pesquisa FAPESP
Textbooks	96,209	11,597	Didatico	Textos para crianças entre a 3ªe 7ªsérie do ensino fundamental
Folhinha	73,575	9,207	Informativo	Notícias escritas para crianças, raspadas em 2015 do suplemento Folhinha do jornal Folha de São Paulo
NILC subcorpus	32,868	4,064	Informativo	Textos escritos para crianças da 3ªe 4ªsérie do ensino fundamental
Para Seu Filho Ler	21,224	3,942	Informativo	Notícias escritas para crianças, do jornal Zero Hora
SARESP	13,308	3,293	Didático	Textos de questões de Matemática, Ciências Humanas e Naturais e redação escritas para avaliação de estudantes
Total	1,395,926,282	3,827,725		

- Tamanho da janela de convolução: $2 \times n$
- Função de ativação: relu

Significando que a função de ativação a ser aplicada será Unidade Linear Retificada (Rectified Linear Unit, ou ReLU), cuja fórmula é $f_x = max_{0,x}$, que 200 filtros para extração de características serão passados em cada aplicação de convolução, e que a função de convolução será aplicada em janelas compostas por dois tokens (sendo n o número de dimensões do token).

O passo seguinte é a camada de acumulação (ou Max Pooling), que reduz o resultado de cada janela de convolução a um valor único, diminuindo a espacialidade da aplicação. Em seguida é aplicado um Dropout de 50%, descartando aleatoriamente metade dos resultados da acumulação, de modo a generalizar o modelo e reduzir a chance de overfitting. Tal descarte é, por definição, somente feito em tempo de treinamento, sendo a camada ignorada na aplicação da rede treinada sobre dados de teste ou aplicação.

O próximo passo é um mero achatamento do *output* para um resultado único, que possa ser avaliado na camada seguinte, de adensamento. Nela, os resultados das camadas anteriores são submetidos à função de ativação, e baseado nisso, apontados para uma das classes disponíveis. Neste caso, a função usada foi softmax, usada para transformar as pontuações das características relativas às classes em probabilidades de pertencimento, e definida por $\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$, onde z_i é o i-ésimo elemento do vetor z de pontuações, K é o total de classes e z_j é a probabilidade do resultado representado pelo vetor z pertencer à j-ésima classe.

Toda a sequência é compilada com o uso de otimizador Adam e função de erro de Entropia Categórica Cruzada. O otimizador é a função que atualiza os pesos dos neurônios ao longo da rede. No caso da implementação Adam (nome derivado de adaptative moment estimation, ou Estimativa adaptativa de momento) trata-se de uma derivação do otimizador clássico de gradiente estocástico, proposto por Kingma e Ba, em 2014 [15]. Sua principal característica é a taxa de aprendizado variável, obtida da segunda derivada dos gradientes, em vez da primeira como no gradiente estocástico. Isso possibilita uma performance particularmente boa em gradientes esparsos, como linguagem natural e visão computacional, tornando-o bastante atraente para aplicação em CNNs.

A função de erro Entropia Categórica Cruzada (categorical crossentropy) pertence à categoria probabilística, ou seja, sua minimização significa a redução da chance de erro do modelo. No caso da função usada, é comparada a probabilidade da distribuição dos eventos entre as categorias e a aproximação dessa probabilidade obtida com o modelo,

como na fórmula $H(P,Q) = -\sum_{x \in X} P(x) log Q(x)$, onde P é a probabilidade real do evento x pertencer à classe X, e Q é a probabilidade do mesmo pertencimento calculada no modelo.

5.4.3.2 LSTM

O segundo modelo utiliza uma rede LSTM, definida segundo o Algoritmo 5. A primeira camada é a própria recorrência, definida com 50 unidades dimensionais, significando que 50 recorrências correm em paralelo, extraindo informações diversas. Diferente do CNN, a função de ativação usada é a tangente hiperbólica (tanh), definida por $tanhx = \frac{e^{2x}-1}{e^{2x}+1}$, o padrão para LSTM no Keras.

A sequencia é um Dropout de 50%, exatamente como no modelo CNN. Como o resultado da recorrência já é um vetor linear, então podemos ignorar o achatamento e aplicar diretamente a camada densa, igualmente com duas classes e softmax. No caso específico dos modelos envolvendo LSTM, uma função restritiva é aplicada aos pesos de kernel, regularizando-os. No caso foi usada a função L2, definida pela adição de λw^2 ao peso usado na camada densa, sendo w o valor original do peso de kernel, e λ uma constante que por padrão (como no nosso caso) é $0,01$.

Algorithm 5 Algoritmo de configuração da LSTM

```
1: function LSTM(tweets, dimensions)
2:     vectorLength ← tweets.length()
3:     model ← Sequential()
4:     model.add(LSTM(units = 50)
5:     model.add(Dropout(rate = 0.5))
6:     model.add(Dense(units = 2, activation = softmax, kernel_regularizer = l2))
7:     model.compile(optimizer = Adam, loss = categorical_crossentropy)
8:     return model
```

5.4.3.3 Modelo Misto

O terceiro modelo utiliza uma combinação de ambas as redes, submetendo o *input* primeiro a um CNN e, em seguida, a uma rede LSTM de 100 unidades. O pseudo-código usado é mostrado no Algoritmo 6.

A sequência, como mencionado, combina os aspectos dos dois outros modelos, começando com uma camada de CNN, com as mesmas configurações da rede original, seguido por um dropout de 25%. O resultado segue para uma camada de LSTM com 100 unidades, mais um dropout de 50%, e uma camada densa usando softmax e regularização de kernel l2.

Algorithm 6 Algoritmo de configuração CNN+LSTM

```
1: function MIXED(tweets, dimensions)
2:     vectorLength ← tweets.length()
3:     model ← Sequential()
4:     model.add(Conv2D(filters = 200, kernel_size = (2, dimensions), activation =
   relu)
5:     model.add(Dropout(rate = 0.25))
6:     model.add(LSTM(units = 100)
7:     model.add(Dropout(rate = 0.5))
8:     model.add(Dense(units = 2, activation = softmax, kernel_regularizer = l2))
9:     model.compile(optimizer = Adam, loss = categorical_crossentropy)
10:    return model
```

5.5 Resultados

Os experimentos foram rodados dez vezes para cada combinação de vetorizador (Word2Vec, GloVe e fastText) $\times$ classificador (CNN, LSTM, CNN+LSTM) $\times$ dimensões (50, 100, 300, 600, 1000). Além disso, foram executada dez repetições para as combinações vetorizador $\times$ classificador com 1000 e 600 dimensões, reduzidas via PCA para 300, gerando um total de 63 combinações.

Como métricas para análise, foram escolhidas as métricas padrão (acurácia, precisão e recall), a medida f1 e os tempos de vetorização e treinamento, sendo que cada um é particularmente definido nas análises das próximas sub-sessões. Os resultados obtidos foram compilados nas tabelas 5.2 a 5.12, que apresentam os valores mínimo, máximo e médio para as métricas usadas. Tais valores também são vistos nos gráficos mostrados nas **??**–5.35, que são de dois tipos. Os de variação dimensional (**??????????????** e figuras 5.4 a 5.6, 5.8 a 5.13, 5.15 a 5.20, 5.22 a 5.27 e 5.29 a 5.34) apresentam a variação da métrica associada em função das dimensões, e têm como curvas os classificadores ou vetorizadores, dependendo do caso, como melhor discutido nas sub-sessões abaixo. Tais gráficos apresentam a média dos resultados como curva, e os valores máximo e mínimo na forma de barras de erro, permitindo a análise visual da variação entre as 10 rodadas. Já os gráficos de comparação de PCA (figuras 5.7, 5.14, 5.21, 5.28 e 5.35) mostram a comparação dos resultados de 300 dimensões com os reduzidos por PCA a partir de 600 e 1000 dimensões. Diferente do anterior, este gráfico trabalha apenas com as médias, não apresentando as variações máxima e mínima

As métricas temporais foram calculadas mediante uma análise do sistema usado para rodar os experimentos. Como um *disclaimer* inicial, reconhecemos que estes valores não são absolutos, dependendo de diversas variáveis conhecidas e desconhecidas, como pro-

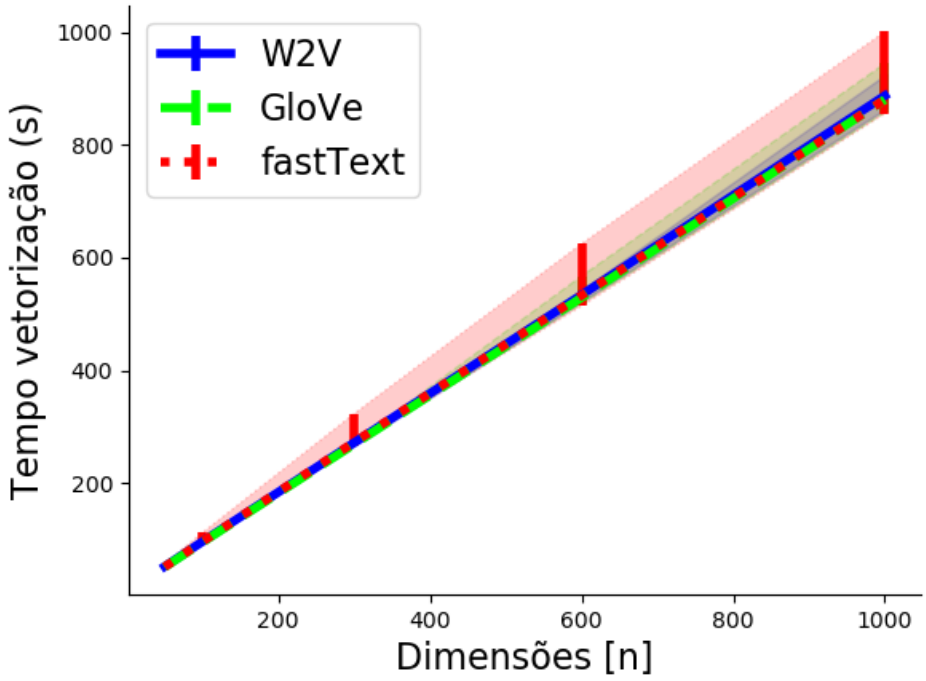

Figura 5.4: Comparação do tempo de vetorização dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN.

gramas rodando simultaneamente, tempo de funcionamento da máquina, atualização do sistema operacional, entre tantas outras. Contudo, optamos por analisá-las por poderem ser comparadas entre si e apresentar uma boa visão do impacto das variações nos experimentos, em particular na dimensionalidade sobre o desempenho do sistema como um todo.

5.5.1 Tempo de Vetorização

O tempo de vetorização é definido nesse trabalho como o tempo passado entre a inicialização da classe de vetorização (onde os parâmetros são configurados) e o penúltimo passo da execução da função vetorizadora (mostrada no algoritmo 3), logo antes do seu retorno. Os tempos são mostrados na tabela 5.2 para a versão sem PCA, e na tabela 5.3 para os resultados afetados por PCA.

Os gráficos exibidos nas figuras 5.4 a 5.6 mostram os comparativos desses tempos de vetorização. Por meio deles podemos observar o crescimento linear dos tempos de vetorização em relação ao número de dimensões. Enquanto a variação dos tempos efetivos é significativa, particularmente na combinações envolvendo o vetorizador GloVe (extremamente pronunciado na combinação com CNN+LSTM), isso não altera a linearidade do

Tabela 5.2: Tempos de vetorização sem PCA

Vetorizador	Classificador	Dimensões	Mínimo	Máximo	Média
Word2Vec	CNN	50	50.73s	56.01s	52.50s
Word2Vec	CNN	100	95.08s	98.12s	96.78s
Word2Vec	CNN	300	265.92s	277.01s	271.97s
Word2Vec	CNN	600	522.70s	542.96s	534.04s
Word2Vec	CNN	1000	862.98s	923.99s	886.95s
Word2Vec	LSTM	50	50.73s	54.66s	52.06s
Word2Vec	LSTM	100	94.95s	103.27s	97.25s
Word2Vec	LSTM	300	267.37s	279.16s	273.18s
Word2Vec	LSTM	600	524.55s	541.16s	535.94s
Word2Vec	LSTM	1000	876.25s	1002.82s	904.71s
Word2Vec	CNN+LSTM	50	50.48s	57.38s	52.79s
Word2Vec	CNN+LSTM	100	92.94s	100.82s	95.93s
Word2Vec	CNN+LSTM	300	261.46s	288.81s	268.93s
Word2Vec	CNN+LSTM	600	515.59s	590.54s	533.04s
Word2Vec	CNN+LSTM	1000	860.02s	884.87s	870.94s
GloVe	CNN	50	50.16s	53.04s	51.35s
GloVe	CNN	100	94.46s	96.63s	95.60s
GloVe	CNN	300	266.27s	272.56s	269.40s
GloVe	CNN	600	518.48s	567.27s	529.29s
GloVe	CNN	1000	857.26s	943.90s	880.06s
GloVe	LSTM	50	50.28s	59.24s	53.00s
GloVe	LSTM	100	93.88s	105.12s	96.86s
GloVe	LSTM	300	266.28s	291.10s	273.11s
GloVe	LSTM	600	515.28s	578.96s	535.82s
GloVe	LSTM	1000	847.73s	930.44s	879.05s
GloVe	CNN+LSTM	50	49.35s	55.52s	51.55s
GloVe	CNN+LSTM	100	92.29s	102.41s	94.74s
GloVe	CNN+LSTM	300	261.72s	287.46s	269.78s
GloVe	CNN+LSTM	600	517.65s	814.55s	571.18s
GloVe	CNN+LSTM	1000	841.01s	1001.68s	877.36s
fastText	CNN	50	49.93s	58.13s	51.91s
fastText	CNN	100	94.01s	112.73s	96.82s
fastText	CNN	300	266.45s	323.58s	272.88s
fastText	CNN	600	514.37s	624.81s	533.15s
fastText	CNN	1000	855.09s	1000.48s	880.12s
fastText	LSTM	50	50.09s	56.30s	51.22s
fastText	LSTM	100	92.65s	106.83s	95.90s
fastText	LSTM	300	260.29s	294.33s	271.88s
fastText	LSTM	600	512.58s	601.15s	531.48s
fastText	LSTM	1000	848.13s	952.23s	873.96s
fastText	CNN+LSTM	50	49.76s	55.73s	51.24s
fastText	CNN+LSTM	100	92.76s	105.13s	95.39s
fastText	CNN+LSTM	300	261.82s	302.79s	270.01s
fastText	CNN+LSTM	600	524.17s	643.68s	550.66s
fastText	CNN+LSTM	1000	853.71s	1030.02s	909.70s

Tabela 5.3: Tempos de vetorização com PCA

Vetorizador	Classificador	Dimensões	Redução	Mínimo	Máximo	Média
Word2Vec	CNN	600	300	287.50s	304.50s	292.07s
Word2Vec	CNN	1000	300	287.41s	295.70s	292.52s
Word2Vec	LSTM	600	300	284.74s	509.59s	309.58s
Word2Vec	LSTM	1000	300	284.88s	298.45s	291.77s
Word2Vec	CNN+LSTM	600	300	267.88s	332.14s	278.75s
Word2Vec	CNN+LSTM	1000	300	267.07s	272.15s	269.13s
GloVe	CNN	600	300	294.69s	300.16s	297.01s
GloVe	CNN	1000	300	270.65s	339.67s	302.90s
GloVe	LSTM	600	300	262.89s	267.38s	264.50s
GloVe	LSTM	1000	300	261.53s	666.15s	307.66s
GloVe	CNN+LSTM	600	300	261.92s	281.47s	266.74s
GloVe	CNN+LSTM	1000	300	263.56s	306.07s	271.47s
fastText	CNN	600	300	263.16s	337.02s	278.97s
fastText	CNN	1000	300	262.86s	269.78s	266.17s
fastText	LSTM	600	300	262.36s	323.68s	283.72s
fastText	LSTM	1000	300	263.53s	321.80s	290.01s
fastText	CNN+LSTM	600	300	263.02s	313.53s	275.80s
fastText	CNN+LSTM	1000	300	269.54s	496.15s	343.72s

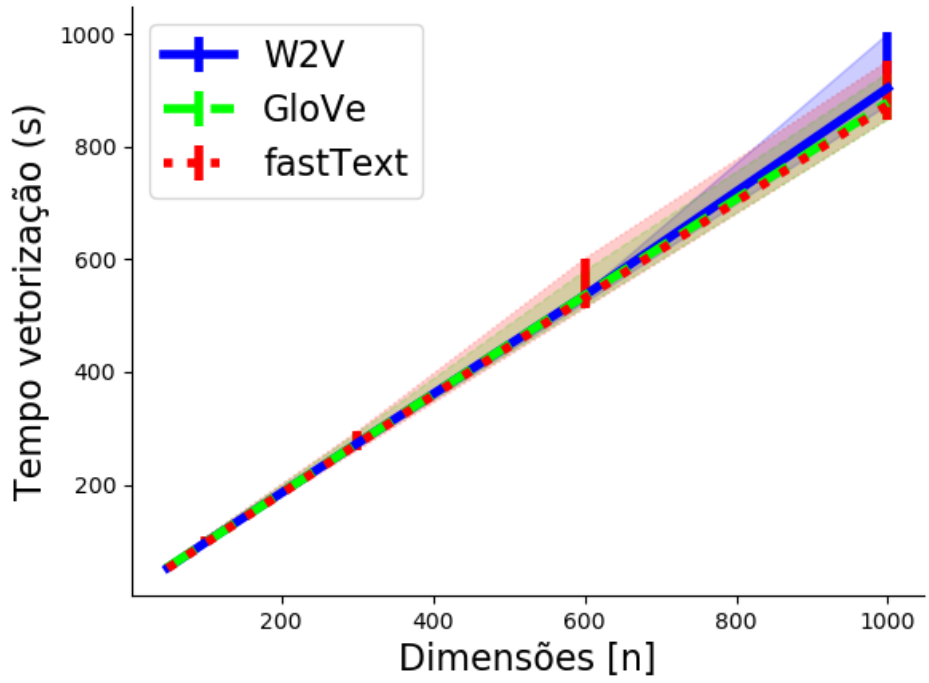

Figura 5.5: Comparação do tempo de vetorização dos modelos Word2Vec, GloVe e fastText utilizando classificação LSTM

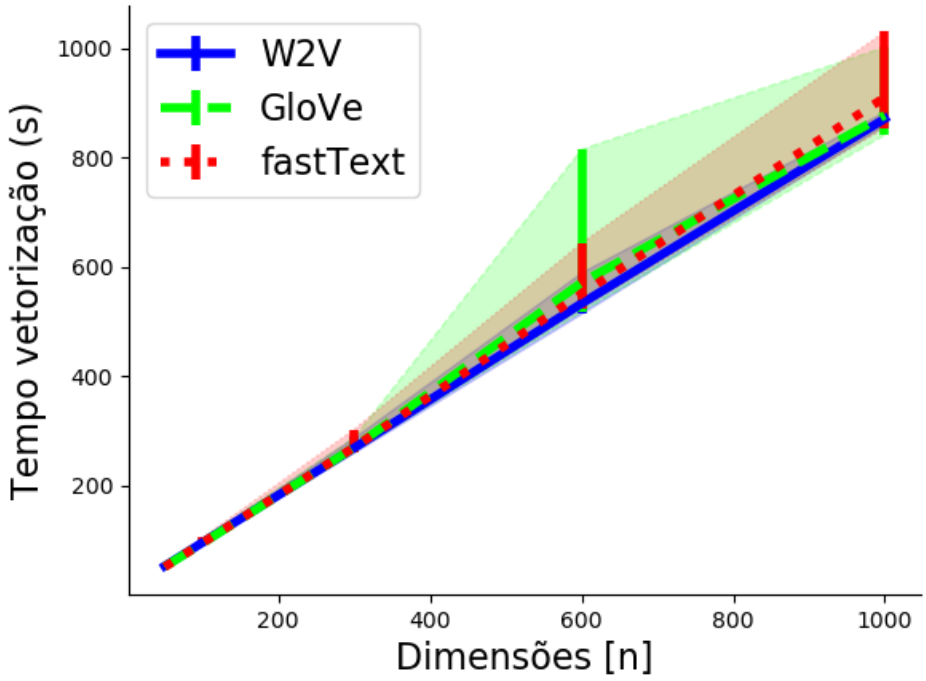

Figura 5.6: Comparação do tempo de vetorização dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN+LSTM

gráfico. As comparações com PCA, explicitadas na figura 5.7 mostram que os resultados com PCA são bastante estáveis, diferente das bem mais perceptíveis variações na média sem PCA.

5.5.2 Tempo de Treinamento

Definimos o tempo de treinamento como o tempo entre a entrada e a saída da função de treinamento de cada rede neural (mostradas nos algoritmos 4 a 6), e é relatada nas tabelas 5.4, para a versão sem PCA, e 5.5, com PCA.

A partir dos gráficos, exibidos nas figuras 5.8 a 5.13, podemos observar alguns pontos importantes. Primeiramente, percebe-se que os tempos de treinamento da rede CNN apresentam uma maior estabilidade e mesmo linearidade, além de serem treinados muito mais rápido que as redes envolvendo LSTM, o que é particularmente perceptível nos gráficos focados em vetorizadores (figuras 5.8 a 5.10). O gráfico 5.14 compara as médias de tempo para as implementações com e sem PCA, e mantém a tendência de CNNs mais rápidos e PCA mais estáveis.

Percebe=se uma clara interferência de variáveis referentes ao uso de processador e memória nos tempos de treinamento. Particularmente, quanto maior a sequência de aces-

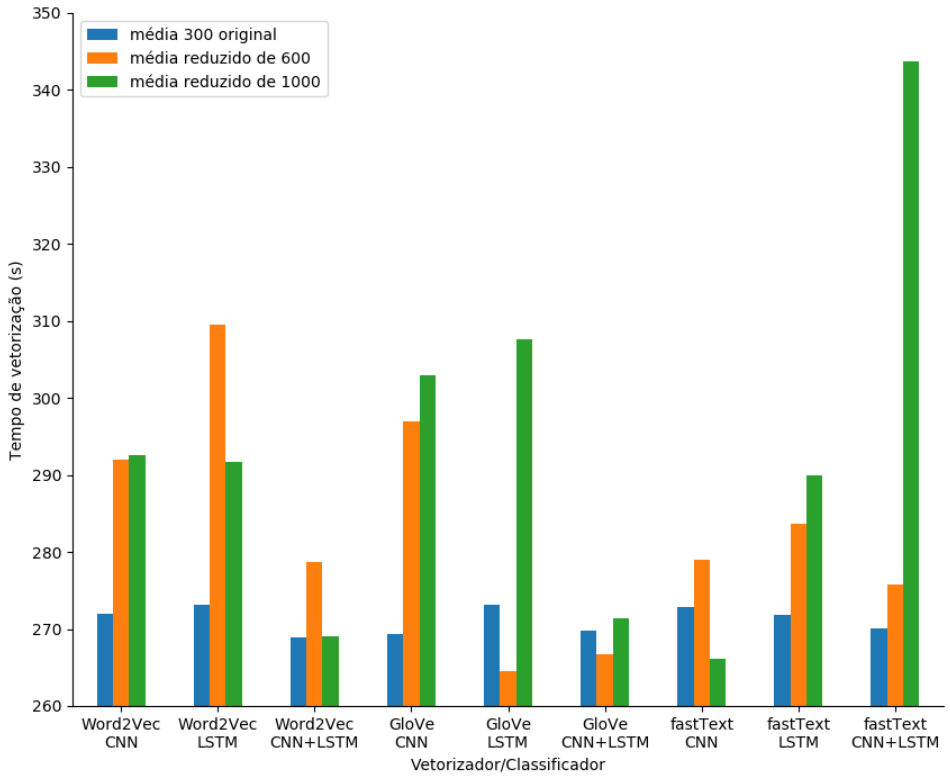

Figura 5.7: Comparação do tempo de vetorização do modelo em 300 dimensões com os modelos reduzidos via PCA, entre as diversas combinações de vetorização e classificação.

Tabela 5.4: Tempos de treinamento sem PCA

Vetorizador	Classificador	Dimensões	Mínimo	Máximo	Média
Word2Vec	CNN	50	57.24s	69.86s	61.26s
Word2Vec	CNN	100	72.62s	80.57s	74.22s
Word2Vec	CNN	300	135.87s	145.10s	140.21s
Word2Vec	CNN	600	235.84s	252.34s	243.58s
Word2Vec	CNN	1000	356.44s	383.68s	370.23s
Word2Vec	LSTM	50	148.73s	186.54s	171.31s
Word2Vec	LSTM	100	155.62s	215.32s	195.07s
Word2Vec	LSTM	300	220.96s	703.94s	536.98s
Word2Vec	LSTM	600	373.48s	2065.83s	1555.45s
Word2Vec	LSTM	1000	457.06s	3760.47s	2508.16s
Word2Vec	CNN+LSTM	50	358.01s	2107.89s	1213.79s
Word2Vec	CNN+LSTM	100	356.41s	470.91s	405.63s
Word2Vec	CNN+LSTM	300	427.20s	661.98s	536.02s
Word2Vec	CNN+LSTM	600	517.25s	718.38s	598.14s
Word2Vec	CNN+LSTM	1000	639.62s	968.61s	764.91s
GloVe	CNN	50	60.42s	82.92s	70.75s
GloVe	CNN	100	74.94s	102.45s	88.47s
GloVe	CNN	300	135.15s	175.86s	157.75s
GloVe	CNN	600	216.37s	276.69s	254.40s
GloVe	CNN	1000	325.90s	425.21s	381.39s
GloVe	LSTM	50	136.47s	210.74s	178.46s
GloVe	LSTM	100	153.87s	338.76s	211.02s
GloVe	LSTM	300	202.04s	867.30s	484.84s
GloVe	LSTM	600	308.49s	1111.20s	613.69s
GloVe	LSTM	1000	489.83s	1992.52s	1150.06s
GloVe	CNN+LSTM	50	364.19s	1204.78s	639.08s
GloVe	CNN+LSTM	100	365.91s	1324.49s	778.43s
GloVe	CNN+LSTM	300	409.75s	1491.74s	999.96s
GloVe	CNN+LSTM	600	510.36s	1569.45s	1085.64s
GloVe	CNN+LSTM	1000	643.66s	1942.99s	1128.28s
fastText	CNN	50	57.70s	84.09s	63.03s
fastText	CNN	100	71.63s	106.18s	80.74s
fastText	CNN	300	132.64s	180.06s	142.78s
fastText	CNN	600	224.47s	289.00s	246.47s
fastText	CNN	1000	350.99s	413.24s	374.49s
fastText	LSTM	50	143.71s	189.56s	172.47s
fastText	LSTM	100	152.67s	242.25s	204.44s
fastText	LSTM	300	195.31s	776.10s	478.27s
fastText	LSTM	600	312.28s	1192.49s	533.89s
fastText	LSTM	1000	481.74s	2016.53s	943.66s
fastText	CNN+LSTM	50	398.96s	1111.35s	691.61s
fastText	CNN+LSTM	100	450.29s	1245.04s	905.94s
fastText	CNN+LSTM	300	538.46s	1514.04s	1126.39s
fastText	CNN+LSTM	600	520.87s	1922.92s	1080.33s
fastText	CNN+LSTM	1000	642.55s	2114.90s	890.78s

Tabela 5.5: Tempos de treinamento com PCA

Vetorizador	Classificador	Dimensões	Redução	Mínimo	Máximo	Média
Word2Vec	CNN	600	300	172.88s	211.13s	181.58s
Word2Vec	CNN	1000	300	165.05s	176.65s	171.07s
Word2Vec	LSTM	600	300	596.02s	1221.51s	802.95s
Word2Vec	LSTM	1000	300	313.67s	566.50s	442.88s
Word2Vec	CNN+LSTM	600	300	2296.01s	3182.93s	2734.29s
Word2Vec	CNN+LSTM	1000	300	1898.05s	2490.90s	2138.32s
GloVe	CNN	600	300	493.23s	525.84s	511.28s
GloVe	CNN	1000	300	466.83s	511.01s	487.99s
GloVe	LSTM	600	300	178.71s	2112.21s	943.90s
GloVe	LSTM	1000	300	1806.82s	2455.82s	1975.29s
GloVe	CNN+LSTM	600	300	457.83s	1242.81s	733.52s
GloVe	CNN+LSTM	1000	300	421.18s	593.85s	478.99s
fastText	CNN	600	300	154.74s	249.74s	219.51s
fastText	CNN	1000	300	148.68s	238.70s	199.77s
fastText	LSTM	600	300	272.10s	683.46s	405.56s
fastText	LSTM	1000	300	199.56s	437.02s	313.20s
fastText	CNN+LSTM	600	300	1339.73s	1830.65s	1498.93s
fastText	CNN+LSTM	1000	300	1138.94s	1582.41s	1381.41s

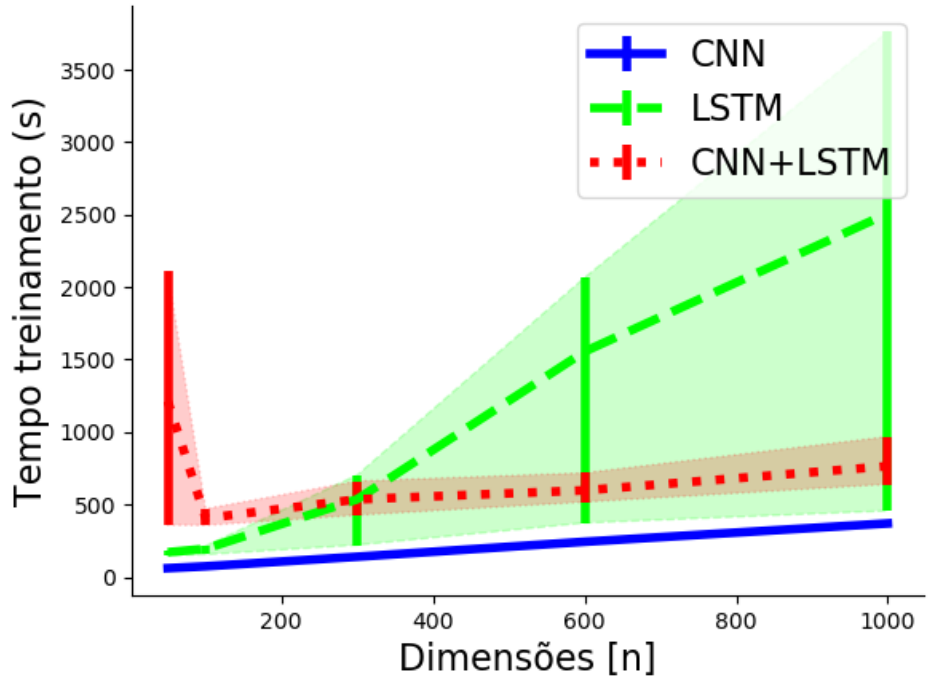

Figura 5.8: Comparação do tempo de treinamento dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização Word2Vec.

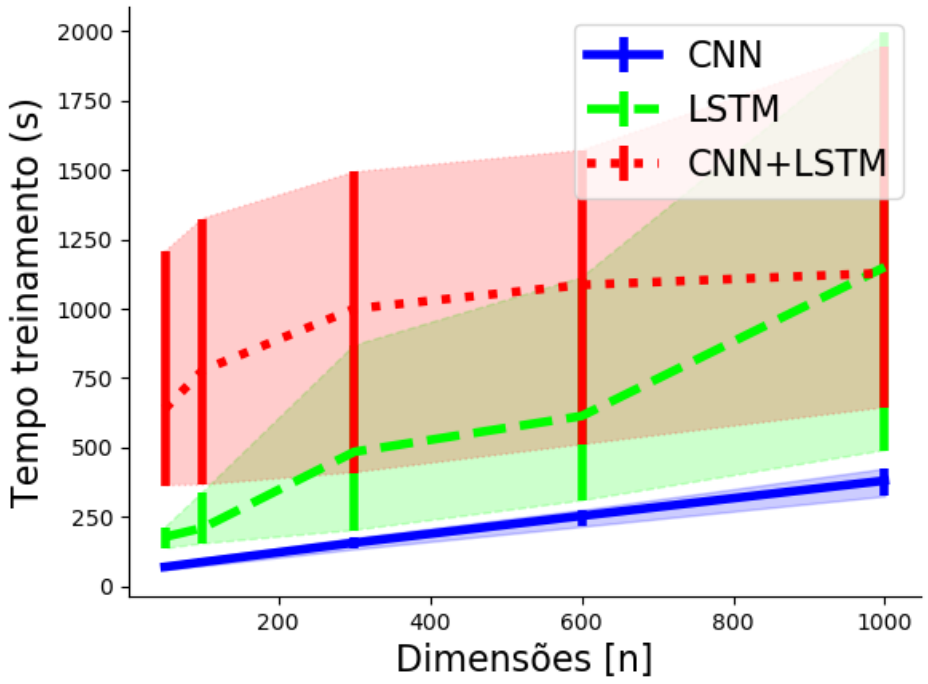

Figura 5.9: Comparação do tempo de treinamento dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização GloVe.

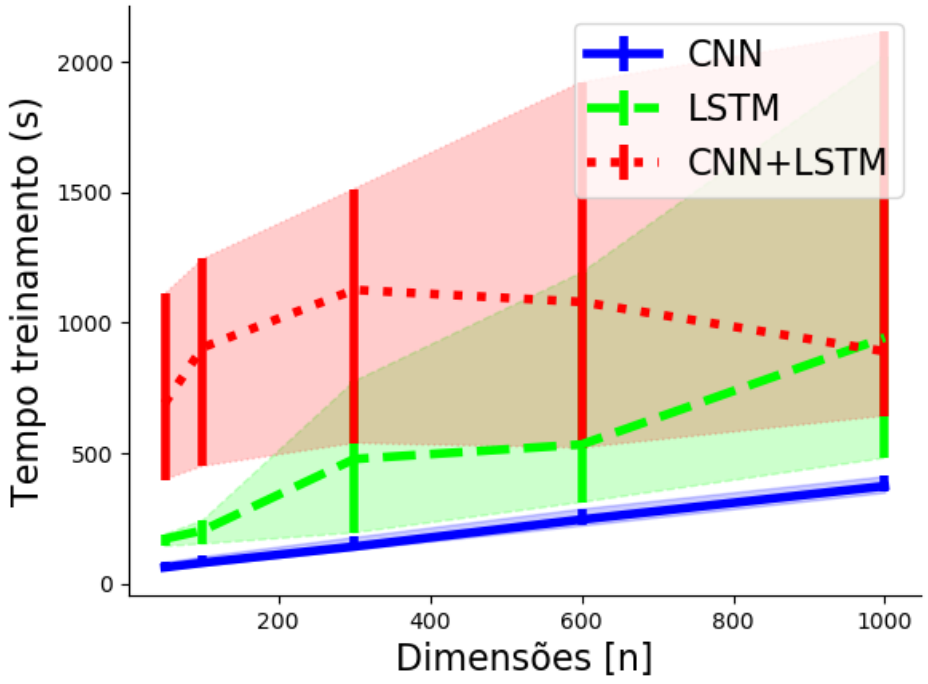

Figura 5.10: Comparação do tempo de treinamento dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização fastText.

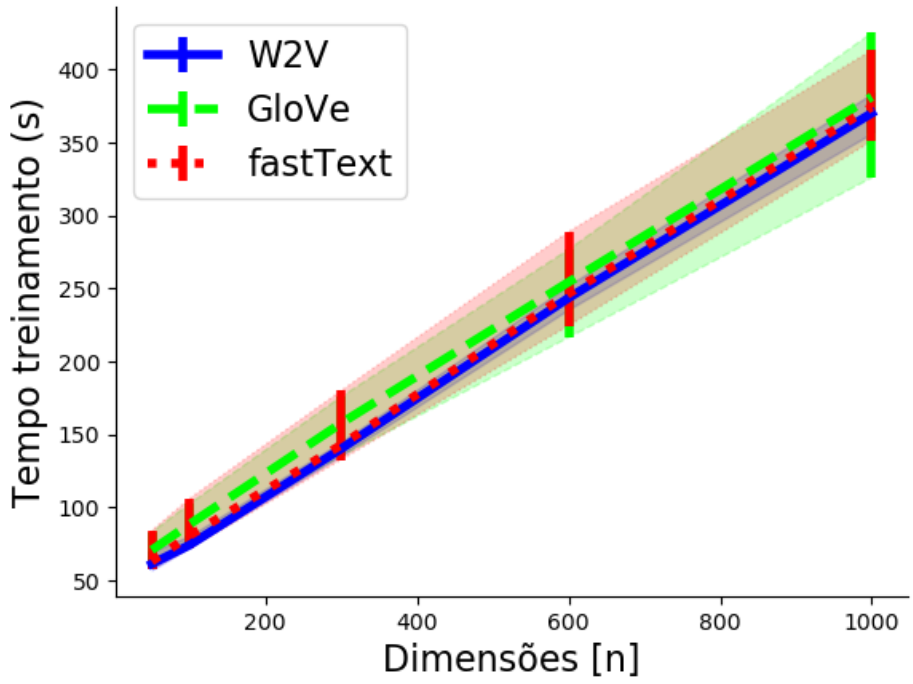

Figura 5.11: Comparação do tempo de treinamento dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN.

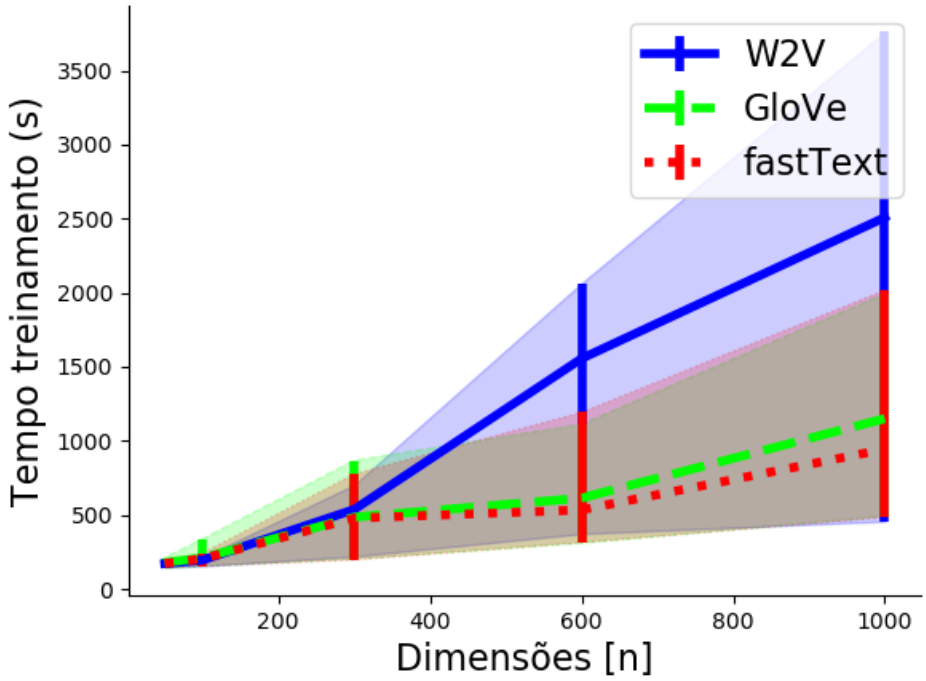

Figura 5.12: Comparação do tempo de treinamento dos modelos Word2Vec, GloVe e fastText utilizando classificação LSTM.

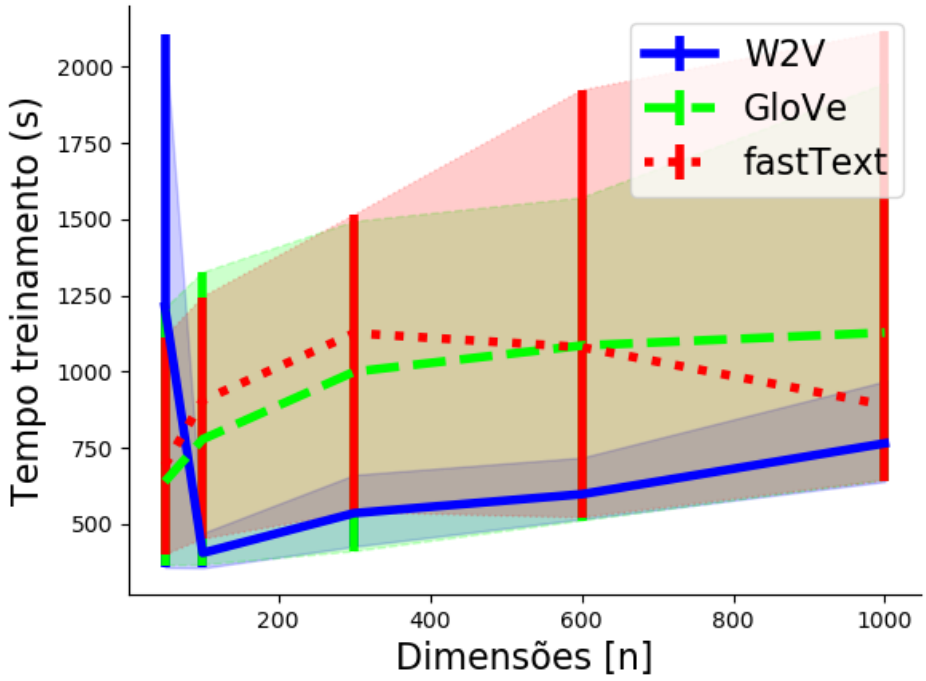

Figura 5.13: Comparação do tempo de treinamento dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN+LSTM.

sos ao *TensorFlow* sem limpeza da memória, mais lentos ficam os treinamentos seguintes, como evidenciado na diferença dos tempos para a rede CNN + LSTM para os vetorizadores Word2Vec e GloVe.

5.5.3 Acurácia

A métrica de acurácia se refere ao número de acertos do classificador, ou seja, os resultados positivos e negativos corretamente avaliados [26]. Sua fórmula é definida como $\frac{TP+TN}{TP+FP+TN+FN}$, onde TP são os resultados corretamente avaliados como positivos, TN corretamente negativos, FP falsamente positivos e FN falsamente negativos. Neste trabalho, os valores obtidos de acurácia estão mostrados nas tabelas 5.6 e 5.7, referindo-se aos resultados sem e com PCA.

Os gráficos referentes à acurácia (representados nas figuras 5.15 a 5.20) demonstram uma variação entre 96% e 98%, e uma grande estabilidade dos resultados, demonstrando o pouco impacto da variação dimensional, e entre as combinações de vetorizadores/classificadores na acurácia do sistema. Em geral a combinação fastText/CNN+LSTM apresenta uma ligeira superioridade na métrica, e o uso de PCA não apresenta variação significativa, como visto no gráfico da figura 5.21.

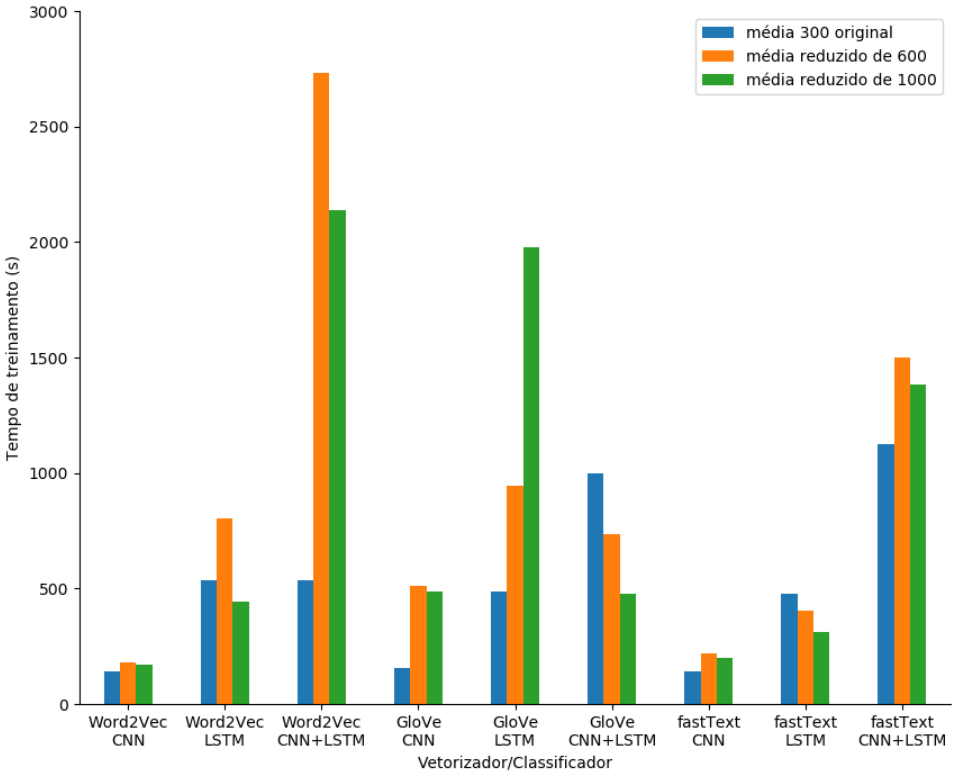

Figura 5.14: Comparação do tempo de treinamento do modelo em 300 dimensões com os modelos reduzidos via PCA, entre as diversas combinações de vetorização e classificação.

Tabela 5.6: Percentagem de Acurácia sem PCA

Vetorizador	Classificador	Dimensões	Mínimo	Máximo	Média
Word2Vec	CNN	50	96.49%	96.64%	96.56%
Word2Vec	CNN	100	96.69%	96.85%	96.78%
Word2Vec	CNN	300	96.75%	96.89%	96.82%
Word2Vec	CNN	600	96.84%	96.91%	96.87%
Word2Vec	CNN	1000	96.77%	96.93%	96.86%
Word2Vec	LSTM	50	96.38%	96.61%	96.50%
Word2Vec	LSTM	100	96.54%	96.83%	96.73%
Word2Vec	LSTM	300	96.63%	96.88%	96.71%
Word2Vec	LSTM	600	96.56%	96.83%	96.70%
Word2Vec	LSTM	1000	96.66%	96.89%	96.76%
Word2Vec	CNN+LSTM	50	96.72%	96.99%	96.79%
Word2Vec	CNN+LSTM	100	96.90%	97.04%	97.00%
Word2Vec	CNN+LSTM	300	96.82%	96.98%	96.93%
Word2Vec	CNN+LSTM	600	96.76%	97.05%	96.90%
Word2Vec	CNN+LSTM	1000	96.84%	97.01%	96.92%
GloVe	CNN	50	96.54%	96.64%	96.59%
GloVe	CNN	100	96.68%	96.83%	96.74%
GloVe	CNN	300	96.81%	96.91%	96.85%
GloVe	CNN	600	96.66%	96.92%	96.77%
GloVe	CNN	1000	96.67%	96.76%	96.72%
GloVe	LSTM	50	96.51%	96.79%	96.59%
GloVe	LSTM	100	96.61%	96.77%	96.69%
GloVe	LSTM	300	96.72%	96.89%	96.78%
GloVe	LSTM	600	96.58%	96.83%	96.71%
GloVe	LSTM	1000	96.57%	96.89%	96.75%
GloVe	CNN+LSTM	50	96.69%	96.90%	96.78%
GloVe	CNN+LSTM	100	96.73%	96.94%	96.82%
GloVe	CNN+LSTM	300	96.70%	96.93%	96.87%
GloVe	CNN+LSTM	600	96.75%	97.09%	96.90%
GloVe	CNN+LSTM	1000	96.76%	97.05%	96.90%
fastText	CNN	50	96.76%	96.89%	96.81%
fastText	CNN	100	96.79%	96.93%	96.89%
fastText	CNN	300	96.90%	97.10%	97.00%
fastText	CNN	600	96.96%	97.07%	97.02%
fastText	CNN	1000	96.98%	97.10%	97.05%
fastText	LSTM	50	96.49%	96.76%	96.61%
fastText	LSTM	100	96.57%	97.02%	96.76%
fastText	LSTM	300	96.79%	97.06%	96.95%
fastText	LSTM	600	96.83%	97.15%	96.98%
fastText	LSTM	1000	96.81%	97.06%	96.94%
fastText	CNN+LSTM	50	96.64%	96.93%	96.80%
fastText	CNN+LSTM	100	96.71%	97.02%	96.89%
fastText	CNN+LSTM	300	96.92%	97.19%	97.04%
fastText	CNN+LSTM	600	96.92%	97.19%	97.07%
fastText	CNN+LSTM	1000	97.00%	97.17%	97.08%

Tabela 5.7: Percentagem de Acurácia com PCA

Vetorizador	Classificador	Dimensões	Redução	Mínimo	Máximo	Média
Word2Vec	CNN	600	300	96.76%	96.87%	96.82%
Word2Vec	CNN	1000	300	96.70%	96.81%	96.76%
Word2Vec	LSTM	600	300	96.41%	96.68%	96.60%
Word2Vec	LSTM	1000	300	96.57%	96.90%	96.67%
Word2Vec	CNN+LSTM	600	300	96.71%	96.88%	96.79%
Word2Vec	CNN+LSTM	1000	300	96.76%	96.91%	96.82%
GloVe	CNN	600	300	96.75%	96.99%	96.81%
GloVe	CNN	1000	300	96.72%	96.86%	96.76%
GloVe	LSTM	600	300	96.57%	96.87%	96.74%
GloVe	LSTM	1000	300	96.69%	96.85%	96.75%
GloVe	CNN+LSTM	600	300	96.65%	96.91%	96.82%
GloVe	CNN+LSTM	1000	300	96.73%	96.99%	96.84%
fastText	CNN	600	300	96.98%	97.20%	97.10%
fastText	CNN	1000	300	96.95%	97.06%	97.00%
fastText	LSTM	600	300	96.76%	97.01%	96.85%
fastText	LSTM	1000	300	96.51%	96.84%	96.74%
fastText	CNN+LSTM	600	300	96.86%	97.05%	96.96%
fastText	CNN+LSTM	1000	300	96.86%	97.07%	96.96%

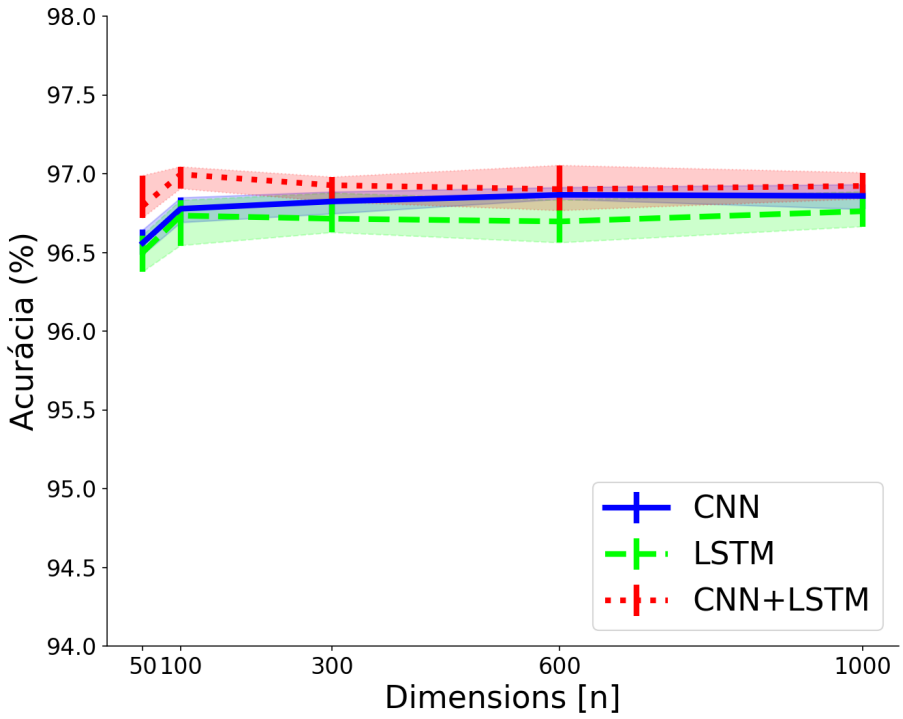

Figura 5.15: Comparação da porcentagem de acurácia dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização Word2Vec.

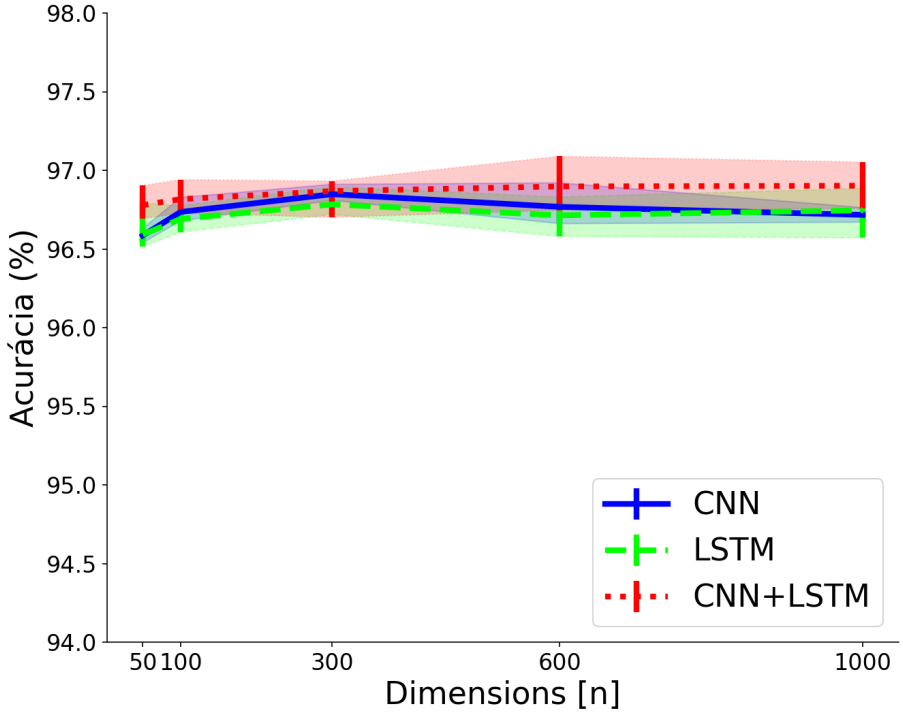

Figura 5.16: Comparação da porcentagem de acurácia dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização GloVe.

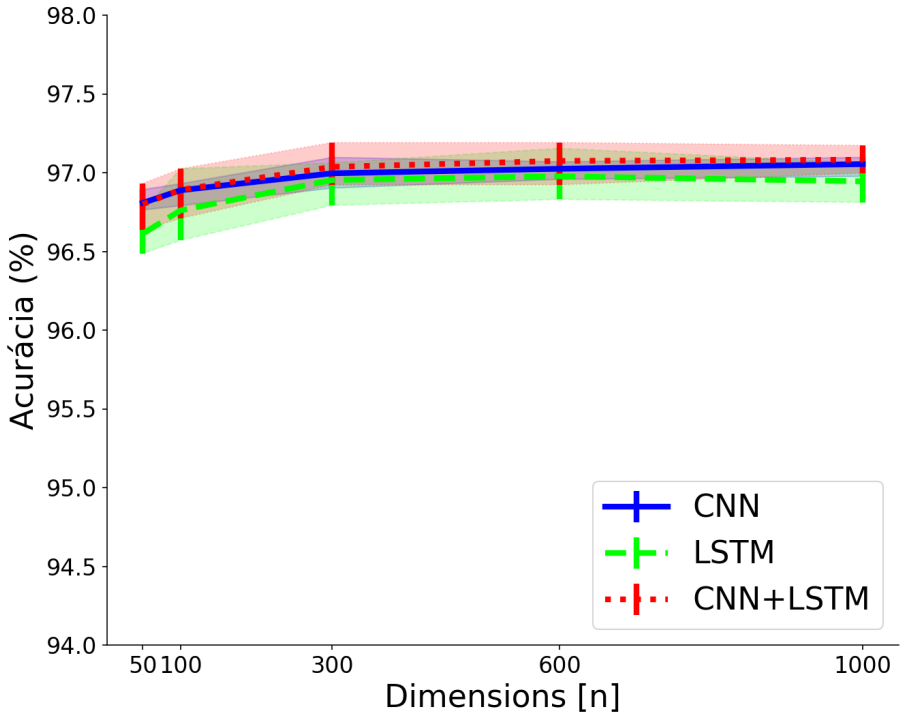

Figura 5.17: Comparação da porcentagem de acurácia dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização fastText.

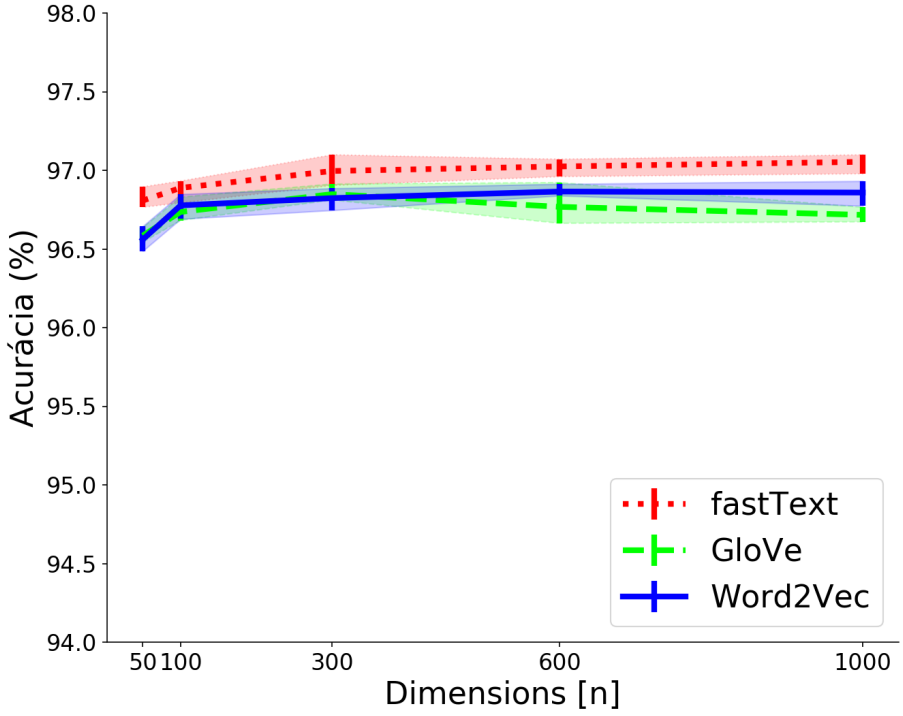

Figura 5.18: Comparação da porcentagem de acurácia dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN.

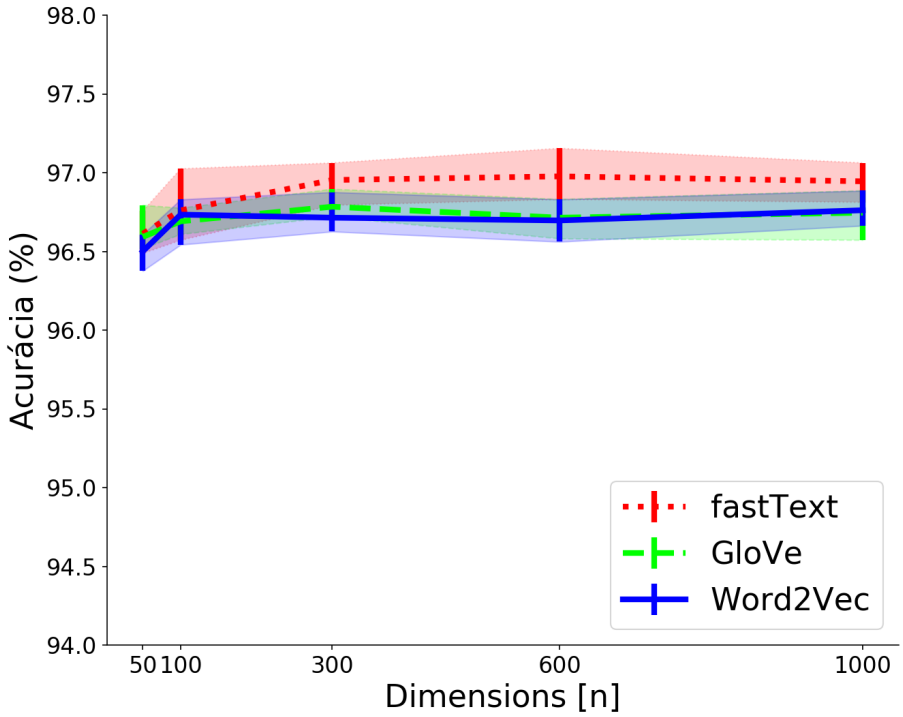

Figura 5.19: Comparação da porcentagem de acurácia dos modelos Word2Vec, GloVe e fastText utilizando classificação LSTM

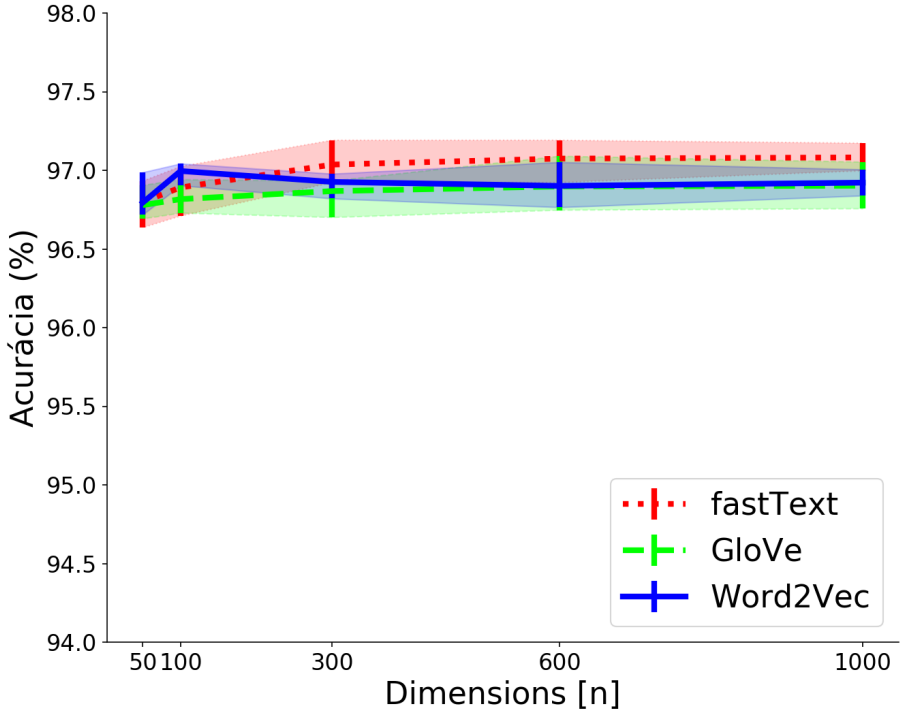

Figura 5.20: Comparação da porcentagem de acurácia dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN+LSTM

Tabela 5.8: Comparação das medidas de acurácia

Vetorizador	Classificador	100 original	300 original	300 reduzido de 1000	200 (Dabiri e Heaslip)
Word2Vec	CNN	96.8%	96.8%	96.8%	98.6%
Word2Vec	LSTM	96.7%	96.7%	96.7%	98.4%
Word2Vec	CNN+LSTM	97.0%	96.9%	96.8%	98.5%
fastText	CNN	96.9%	97.0%	97.0%	98.6%
fastText	LSTM	96.8%	97.0%	96.7%	98.5%
fastText	CNN+LSTM	96.9%	97.0%	97.0%	98.6%

Na tabela 5.8 comparamos a acurácia da aplicação em língua portuguesa com a proposta original de Dabiri e Heaslip, de forma indireta, uma vez que não só a proposta original usa vetorizadores de 200 dimensões, grandeza não explorada nesse trabalho, como cada trabalho foi avaliado em um dataset diferente. Para mitigar os efeitos da diferença dimensional, foi feita uma comparação com 100 e 300 dimensões, além de 1000 dimensões reduzido para 300 via PCA. Cabe ainda observar que o trabalho de Dabiri e Heaslip não utilizou o vetorizador GloVe, motivo pelo qual ele foi omitido da comparação.

Nota-se uma defasagem de pouco menos de 2% da nossa aplicação em relação ao modelo original, com ou sem redução dimensional, o que parece indicar a necessidade de melhoria nos parâmetros, melhor adequando o modelo às características da língua

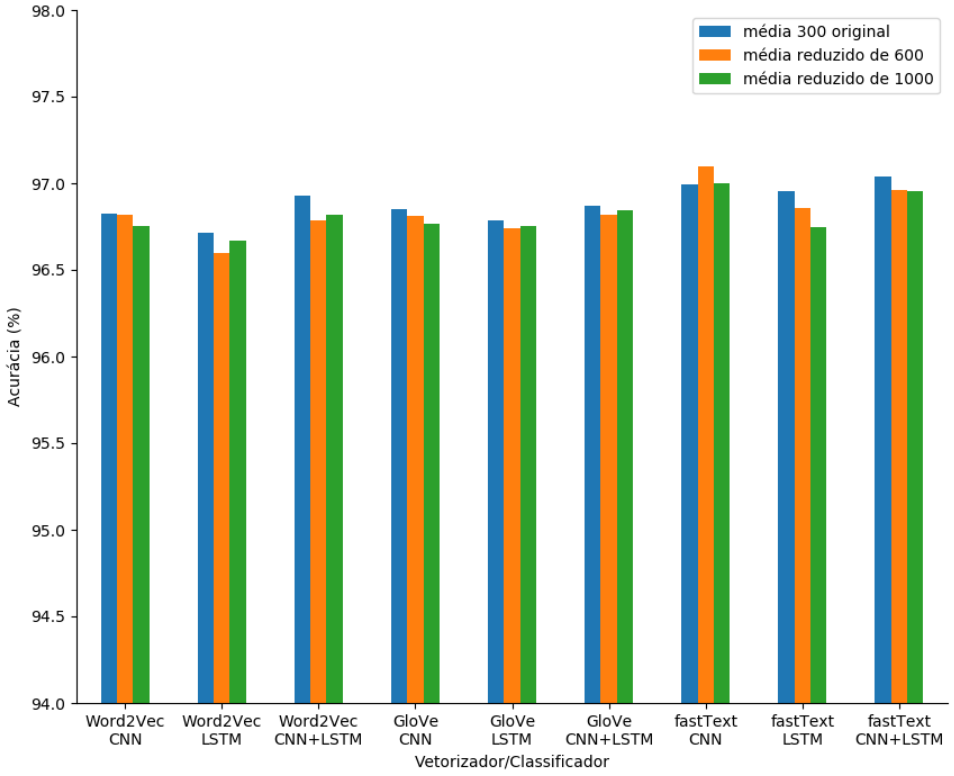

Figura 5.21: Comparação da porcentagem de acurácia do modelo em 300 dimensões com os modelos reduzidos via PCA, entre as diversas combinações de vetorização e classificação

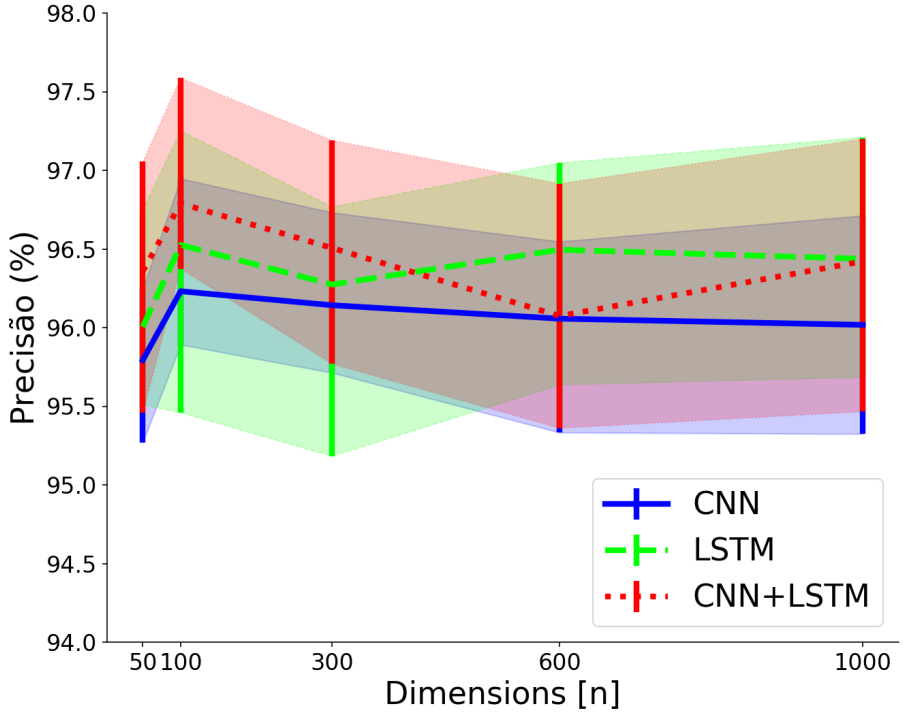

Figura 5.22: Comparação da porcentagem de precisão dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização Word2Vec.

portuguesa.

5.5.4 Precisão

Precisão é a métrica definida como a taxa de acertos dentre os resultados assumidos como positivos [26]. A fórmula que a define é $\frac{TP}{TP+FP}$. As tabelas 5.9 e 5.10 registram os resultados de precisão, sem e com PCA.

Os gráficos referentes à precisão (representados nas figuras 5.22 a 5.27) demonstram uma variação entre 95% e 98%, e resultados um pouco menos estáveis, embora superiores à acurácia. Da mesma forma que o com a acurácia, a interferência da variação dimensional é mínima, e a diferença entre a aplicação ou não de PCA (figura 5.28) também muito pouco significativa. Os resultados envolvendo o vetorizador Word2Vec e o classificador LSTM demonstram uma ligeira inferioridade.

Tabela 5.9: Percentagem de Precisão sem PCA

Vetorizador	Classificador	Dimensões	Mínimo	Máximo	Média
Word2Vec	CNN	50	95.27%	96.24%	95.79%
Word2Vec	CNN	100	95.89%	96.95%	96.23%
Word2Vec	CNN	300	95.71%	96.73%	96.14%
Word2Vec	CNN	600	95.33%	96.55%	96.06%
Word2Vec	CNN	1000	95.32%	96.71%	96.02%
Word2Vec	LSTM	50	95.51%	96.75%	96.00%
Word2Vec	LSTM	100	95.46%	97.25%	96.53%
Word2Vec	LSTM	300	95.18%	96.77%	96.27%
Word2Vec	LSTM	600	95.64%	97.05%	96.50%
Word2Vec	LSTM	1000	95.68%	97.21%	96.44%
Word2Vec	CNN+LSTM	50	95.46%	97.05%	96.36%
Word2Vec	CNN+LSTM	100	96.37%	97.59%	96.79%
Word2Vec	CNN+LSTM	300	95.77%	97.19%	96.51%
Word2Vec	CNN+LSTM	600	95.36%	96.92%	96.07%
Word2Vec	CNN+LSTM	1000	95.47%	97.20%	96.42%
GloVe	CNN	50	95.07%	96.19%	95.67%
GloVe	CNN	100	95.77%	96.87%	96.22%
GloVe	CNN	300	95.32%	97.00%	96.08%
GloVe	CNN	600	95.61%	96.84%	96.21%
GloVe	CNN	1000	95.90%	97.03%	96.55%
GloVe	LSTM	50	95.11%	96.23%	95.76%
GloVe	LSTM	100	95.88%	96.55%	96.23%
GloVe	LSTM	300	95.84%	96.71%	96.35%
GloVe	LSTM	600	95.65%	97.35%	96.52%
GloVe	LSTM	1000	95.41%	96.93%	96.26%
GloVe	CNN+LSTM	50	96.25%	97.13%	96.60%
GloVe	CNN+LSTM	100	95.91%	97.38%	96.60%
GloVe	CNN+LSTM	300	95.45%	97.34%	96.37%
GloVe	CNN+LSTM	600	95.52%	97.32%	96.33%
GloVe	CNN+LSTM	1000	95.58%	97.24%	96.37%
fastText	CNN	50	95.82%	96.85%	96.19%
fastText	CNN	100	95.57%	96.95%	96.19%
fastText	CNN	300	95.60%	97.22%	96.57%
fastText	CNN	600	96.10%	97.54%	96.68%
fastText	CNN	1000	95.99%	97.47%	96.63%
fastText	LSTM	50	94.94%	97.23%	96.43%
fastText	LSTM	100	95.10%	96.86%	96.31%
fastText	LSTM	300	95.75%	97.85%	96.60%
fastText	LSTM	600	95.50%	97.13%	96.50%
fastText	LSTM	1000	95.72%	96.73%	96.22%
fastText	CNN+LSTM	50	96.14%	97.60%	96.75%
fastText	CNN+LSTM	100	96.29%	97.13%	96.64%
fastText	CNN+LSTM	300	95.69%	97.04%	96.37%
fastText	CNN+LSTM	600	95.72%	97.42%	96.51%
fastText	CNN+LSTM	1000	95.91%	97.04%	96.49%

Tabela 5.10: Percentagem de Precisão com PCA

Vetorizador	Classificador	Dimensões	Redução	Mínimo	Máximo	Média
Word2Vec	CNN	600	300	95.80%	96.69%	96.19%
Word2Vec	CNN	1000	300	95.61%	96.75%	96.14%
Word2Vec	LSTM	600	300	95.17%	96.50%	95.84%
Word2Vec	LSTM	1000	300	95.50%	96.85%	96.06%
Word2Vec	CNN+LSTM	600	300	95.12%	97.07%	96.21%
Word2Vec	CNN+LSTM	1000	300	95.90%	97.28%	96.42%
GloVe	CNN	600	300	95.54%	96.60%	96.14%
GloVe	CNN	1000	300	95.72%	96.95%	96.29%
GloVe	LSTM	600	300	95.66%	97.02%	96.34%
GloVe	LSTM	1000	300	95.86%	97.03%	96.46%
GloVe	CNN+LSTM	600	300	95.29%	97.31%	96.45%
GloVe	CNN+LSTM	1000	300	95.70%	97.16%	96.37%
fastText	CNN	600	300	95.89%	96.89%	96.38%
fastText	CNN	1000	300	95.98%	96.70%	96.35%
fastText	LSTM	600	300	95.91%	97.04%	96.33%
fastText	LSTM	1000	300	95.07%	96.91%	96.08%
fastText	CNN+LSTM	600	300	95.68%	97.08%	96.28%
fastText	CNN+LSTM	1000	300	95.13%	97.20%	96.43%

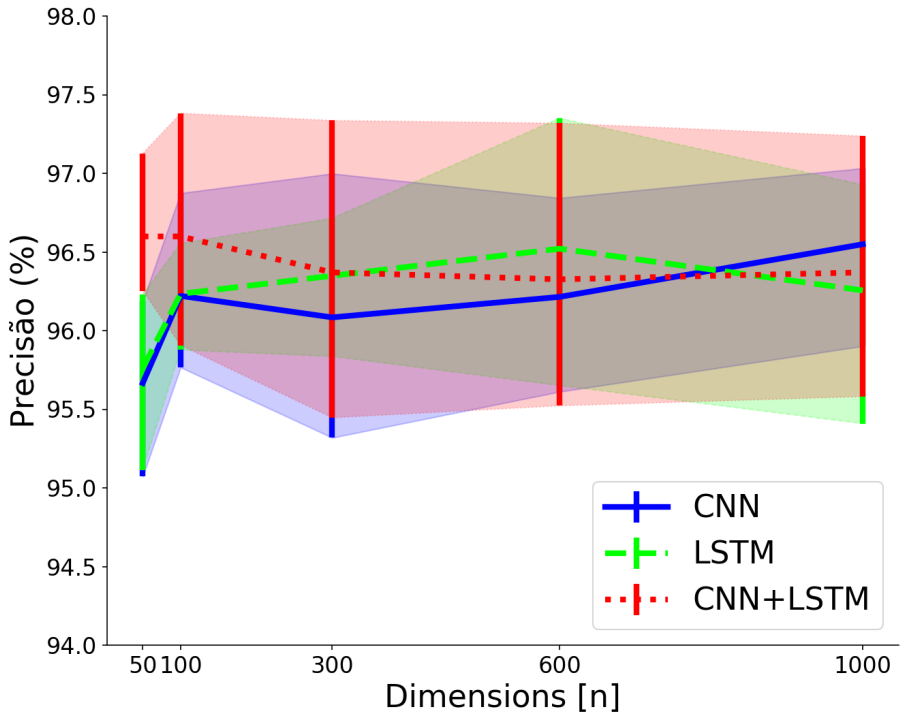

Figura 5.23: Comparação da porcentagem de precisão dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização GloVe.

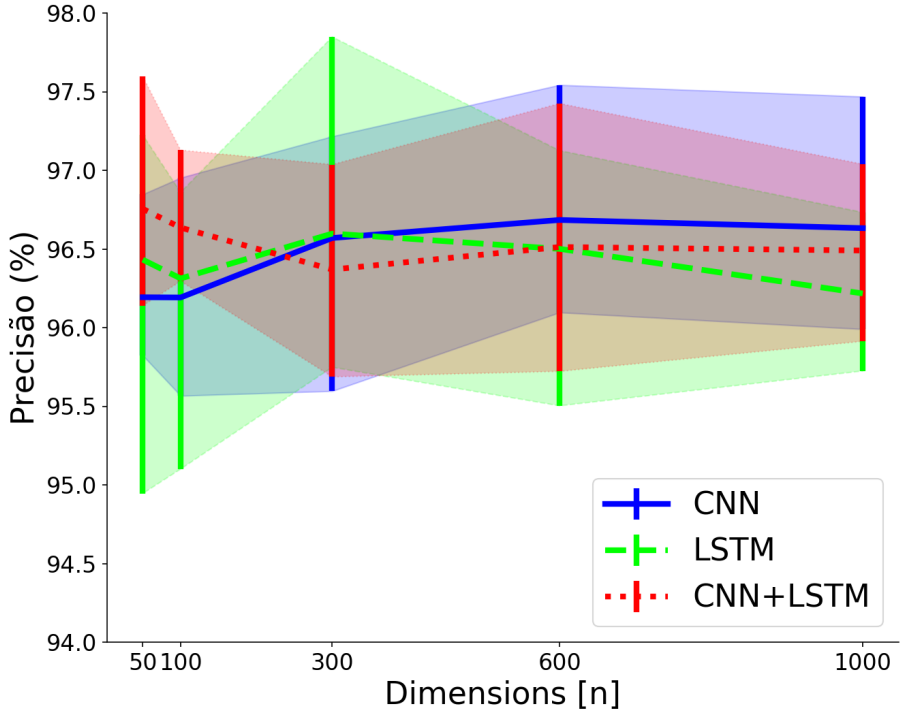

Figura 5.24: Comparação da porcentagem de precisão dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização fastText.

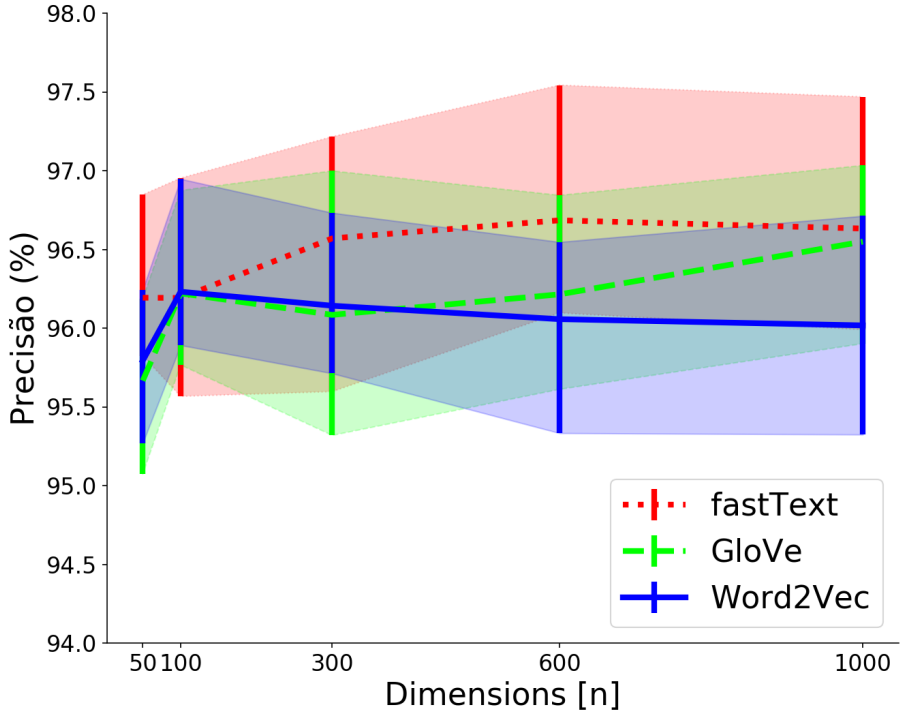

Figura 5.25: Comparação da porcentagem de precisão dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN.

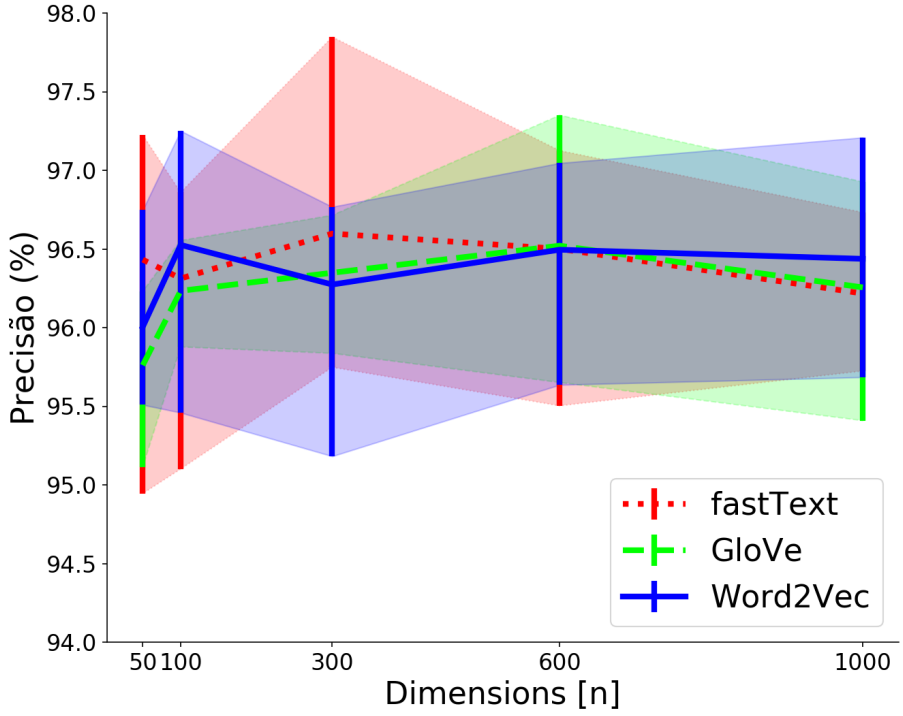

Figura 5.26: Comparação da porcentagem de precisão dos modelos Word2Vec, GloVe e fastText utilizando classificação LSTM

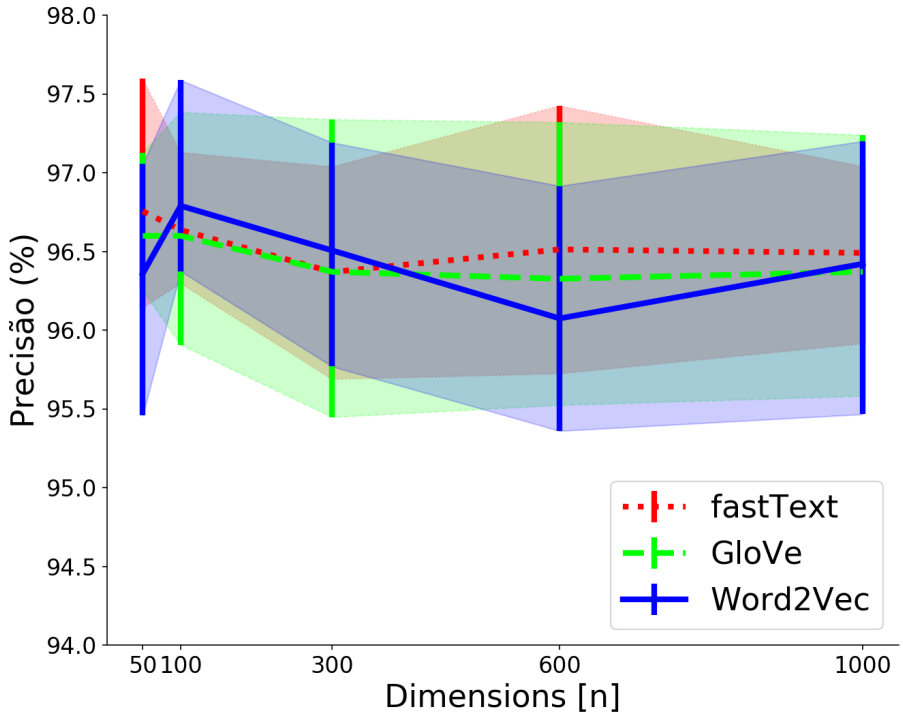

Figura 5.27: Comparação da porcentagem de precisão dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN+LSTM

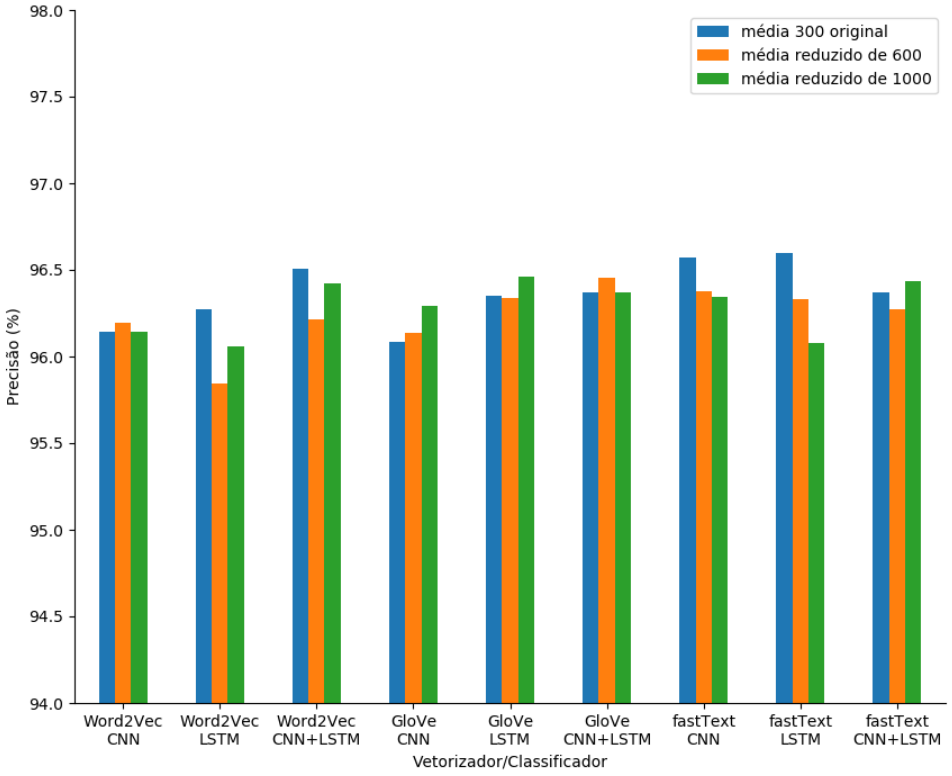

Figura 5.28: Comparação da porcentagem de precisão do modelo em 300 dimensões com os modelos reduzidos via PCA, entre as diversas combinações de vetorização e classificação.

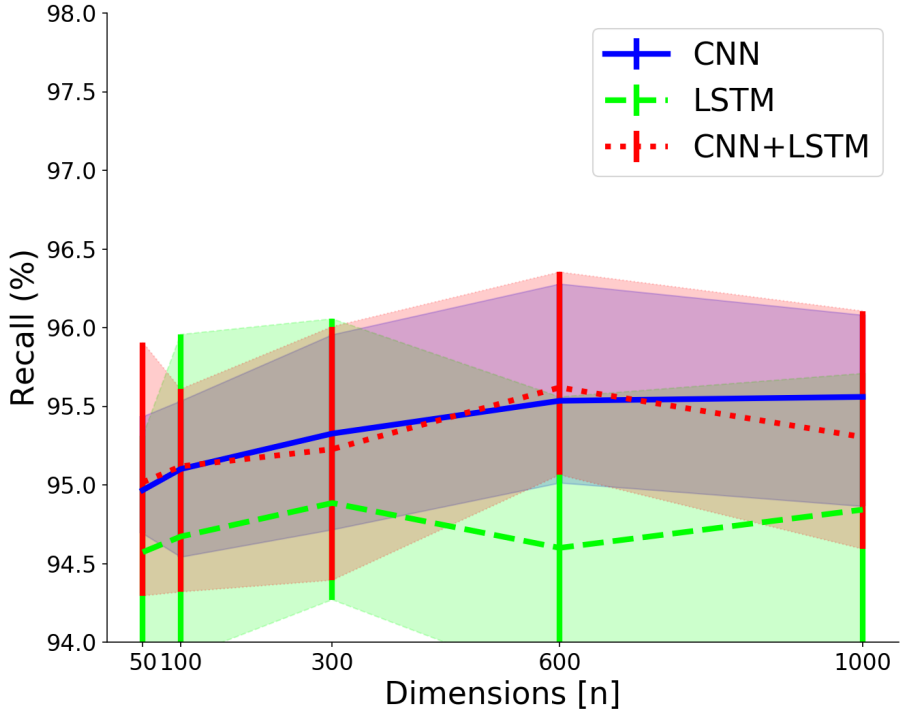

Figura 5.29: Comparação da porcentagem de recall dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização Word2Vec.

5.5.5 Recall

A métrica Recall é definida como a taxa de acertos dentre os resultados cujo gabarito identifica como positivos [26]. A fórmula que a define é $\frac{TP}{TP+FN}$. As tabelas 5.11 e 5.12 mostram os seus resultados, sem e com PCA.

Os gráficos de recall (figuras 5.29 a 5.34) apresentam uma variação entre 93% e 97%, e resultados bastante instáveis, além de geralmente inferiores às outras métricas. Assim como com as outras métricas não há interferência perceptível da variação dimensional, nem da diferença entre o uso ou não de PCA (figura 5.35). Os resultados envolvendo a combinação fastText/CNN+LSTM apresentam uma ligeira superioridade, como com a acurácia.

5.5.6 Medida F1

Como forma de comparar a precisão e recall da aplicação em língua portuguesa com a proposta original de Dabiri e Heaslip, analisamos a medida F1 (*F1 Score*), que calcula a média harmônica entre as duas métricas, dada pela fórmula $F_1 = 2 \times \frac{precisao \times recall}{precisao + recall} =$

Tabela 5.11: Percentagem de Recall sem PCA

Vetorizador	Classificador	Dimensões	Mínimo	Máximo	Média
Word2Vec	CNN	50	94.69%	95.44%	94.97%
Word2Vec	CNN	100	94.54%	95.53%	95.10%
Word2Vec	CNN	300	94.72%	95.96%	95.33%
Word2Vec	CNN	600	95.01%	96.28%	95.53%
Word2Vec	CNN	1000	94.86%	96.08%	95.56%
Word2Vec	LSTM	50	93.72%	95.31%	94.57%
Word2Vec	LSTM	100	93.90%	95.96%	94.67%
Word2Vec	LSTM	300	94.27%	96.06%	94.88%
Word2Vec	LSTM	600	93.72%	95.56%	94.60%
Word2Vec	LSTM	1000	94.00%	95.71%	94.84%
Word2Vec	CNN+LSTM	50	94.29%	95.91%	95.02%
Word2Vec	CNN+LSTM	100	94.32%	95.61%	95.12%
Word2Vec	CNN+LSTM	300	94.39%	96.01%	95.23%
Word2Vec	CNN+LSTM	600	95.06%	96.35%	95.62%
Word2Vec	CNN+LSTM	1000	94.59%	96.11%	95.30%
GloVe	CNN	50	94.49%	95.73%	95.17%
GloVe	CNN	100	94.57%	95.41%	95.00%
GloVe	CNN	300	94.59%	96.48%	95.46%
GloVe	CNN	600	94.37%	95.66%	95.09%
GloVe	CNN	1000	94.10%	95.29%	94.60%
GloVe	LSTM	50	94.37%	95.63%	95.09%
GloVe	LSTM	100	94.47%	95.36%	94.86%
GloVe	LSTM	300	94.57%	95.53%	94.99%
GloVe	LSTM	600	93.87%	95.48%	94.62%
GloVe	LSTM	1000	94.17%	95.88%	94.99%
GloVe	CNN+LSTM	50	93.92%	95.11%	94.72%
GloVe	CNN+LSTM	100	94.12%	95.31%	94.82%
GloVe	CNN+LSTM	300	94.22%	96.18%	95.21%
GloVe	CNN+LSTM	600	94.47%	96.30%	95.34%
GloVe	CNN+LSTM	1000	94.29%	96.03%	95.30%
fastText	CNN	50	94.49%	95.61%	95.23%
fastText	CNN	100	94.72%	96.25%	95.46%
fastText	CNN	300	94.42%	96.40%	95.35%
fastText	CNN	600	94.54%	95.88%	95.31%
fastText	CNN	1000	94.57%	96.20%	95.45%
fastText	LSTM	50	93.75%	95.98%	94.44%
fastText	LSTM	100	94.12%	96.30%	94.97%
fastText	LSTM	300	93.62%	96.08%	95.21%
fastText	LSTM	600	94.72%	96.40%	95.37%
fastText	LSTM	1000	94.74%	96.13%	95.58%
fastText	CNN+LSTM	50	93.70%	95.29%	94.61%
fastText	CNN+LSTM	100	93.97%	95.56%	94.99%
fastText	CNN+LSTM	300	94.91%	96.38%	95.68%
fastText	CNN+LSTM	600	94.69%	96.60%	95.63%
fastText	CNN+LSTM	1000	94.86%	96.25%	95.67%

Tabela 5.12: Percentagem de Recall com PCA

Vetorizador	Classificador	Dimensões	Redução	Mínimo	Máximo	Média
Word2Vec	CNN	600	300	94.69%	95.63%	95.25%
Word2Vec	CNN	1000	300	94.47%	95.56%	95.14%
Word2Vec	LSTM	600	300	94.22%	95.53%	95.01%
Word2Vec	LSTM	1000	300	93.87%	95.56%	94.99%
Word2Vec	CNN+LSTM	600	300	94.19%	96.33%	95.15%
Word2Vec	CNN+LSTM	1000	300	94.17%	95.63%	95.01%
GloVe	CNN	600	300	94.86%	95.76%	95.30%
GloVe	CNN	1000	300	94.47%	95.48%	95.00%
GloVe	LSTM	600	300	94.07%	95.26%	94.89%
GloVe	LSTM	1000	300	94.10%	95.31%	94.79%
GloVe	CNN+LSTM	600	300	94.07%	95.78%	94.98%
GloVe	CNN+LSTM	1000	300	94.29%	96.11%	95.13%
fastText	CNN	600	300	95.19%	96.30%	95.84%
fastText	CNN	1000	300	95.11%	95.96%	95.59%
fastText	LSTM	600	300	94.49%	95.93%	95.21%
fastText	LSTM	1000	300	94.29%	96.13%	95.18%
fastText	CNN+LSTM	600	300	94.82%	96.20%	95.57%
fastText	CNN+LSTM	1000	300	94.82%	96.82%	95.39%

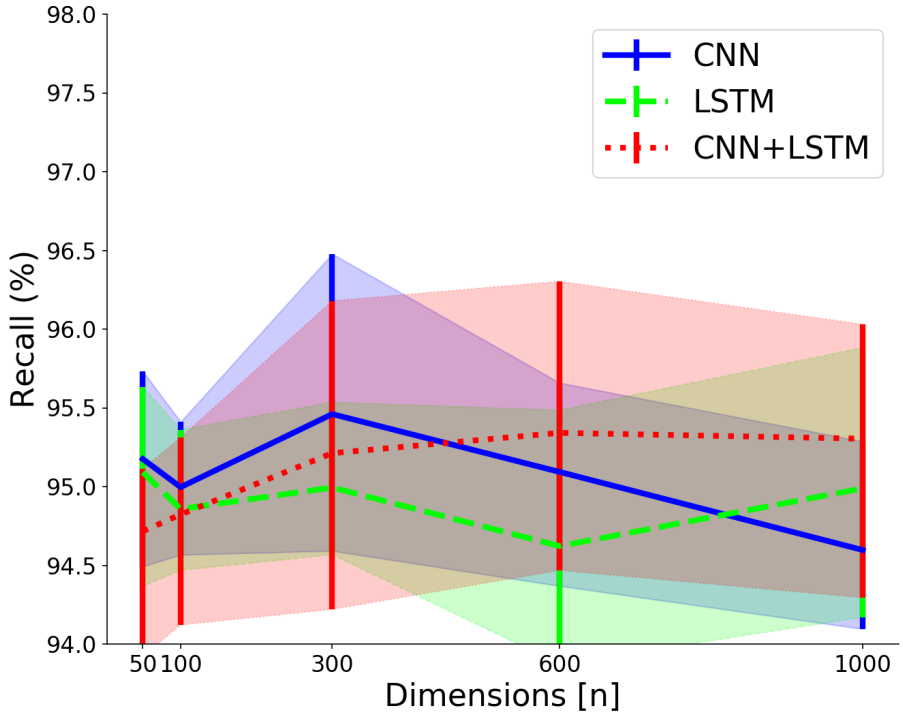

Figura 5.30: Comparação da porcentagem de recall dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização GloVe.

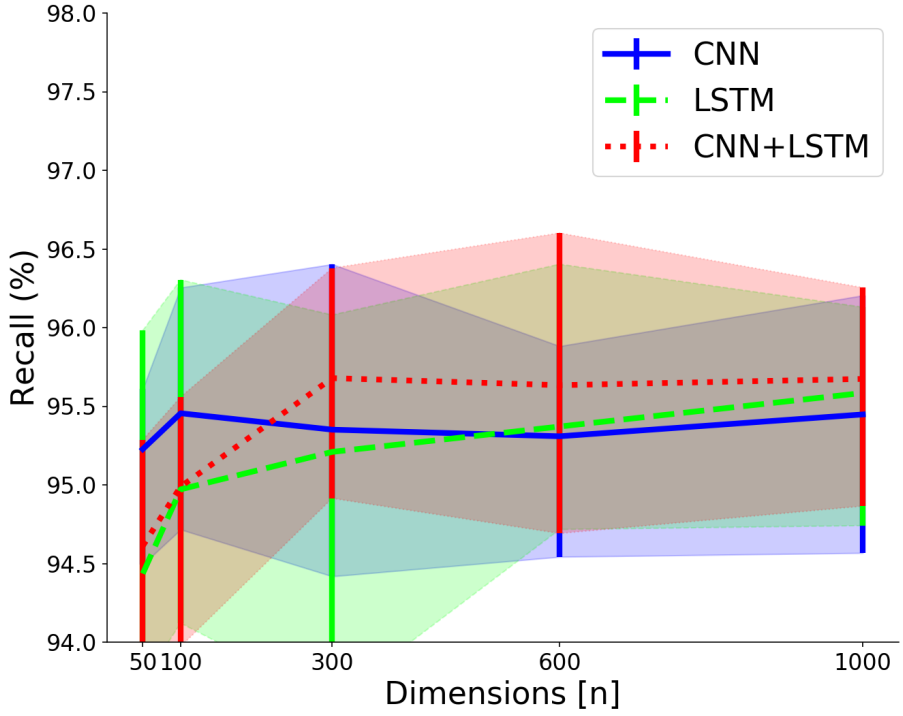

Figura 5.31: Comparação da porcentagem de recall dos modelos CNN, LSTM e CNN+LSTM utilizando vetorização fastText.

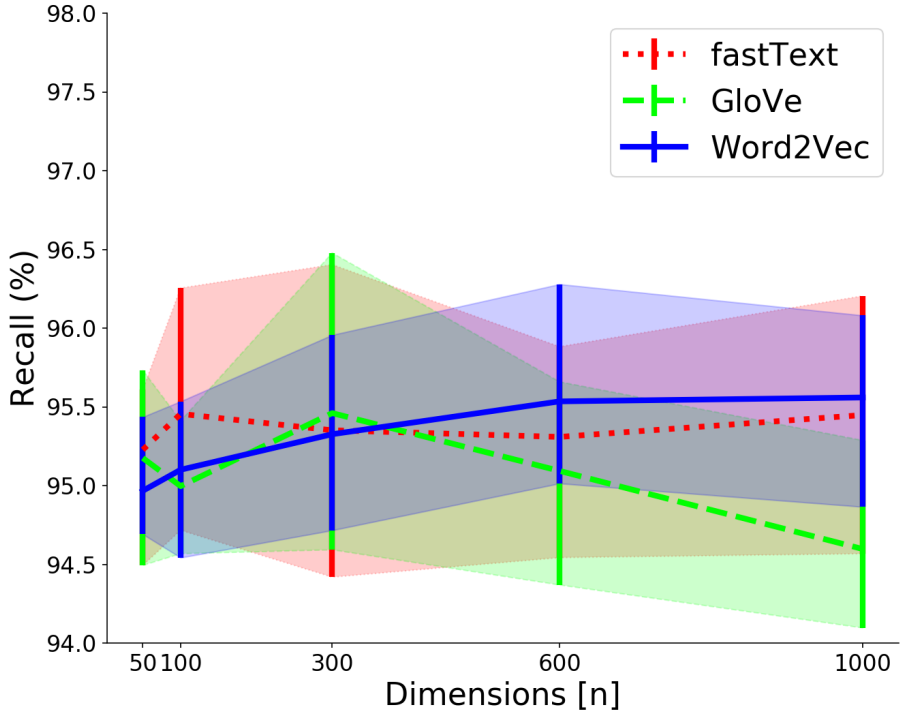

Figura 5.32: Comparação da porcentagem de recall dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN.

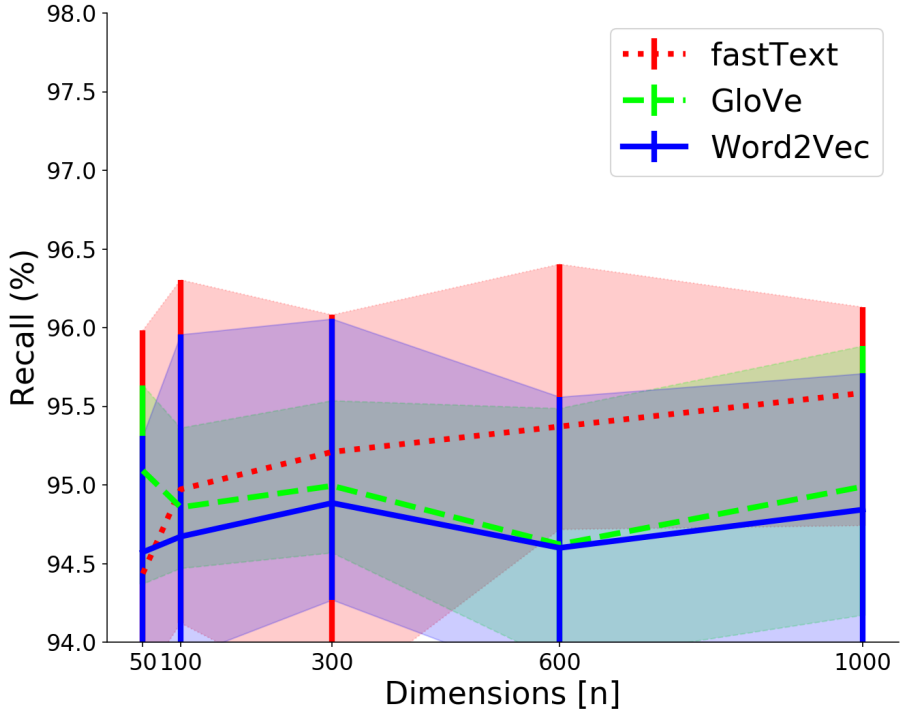

Figura 5.33: Comparação da porcentagem de recall dos modelos Word2Vec, GloVe e fastText utilizando classificação LSTM

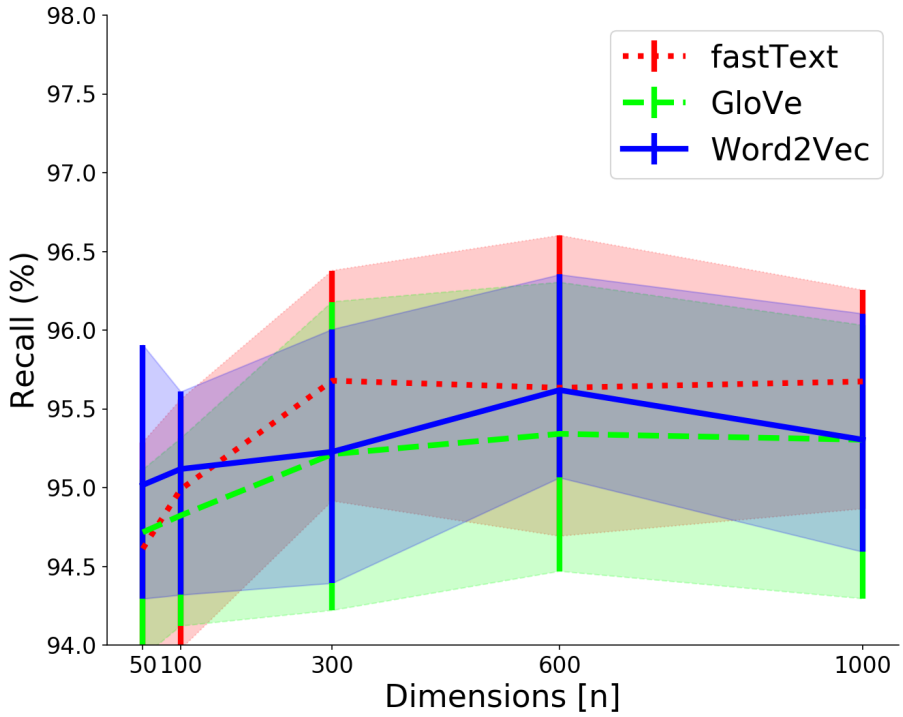

Figura 5.34: Comparação da porcentagem de recall dos modelos Word2Vec, GloVe e fastText utilizando classificação CNN+LSTM

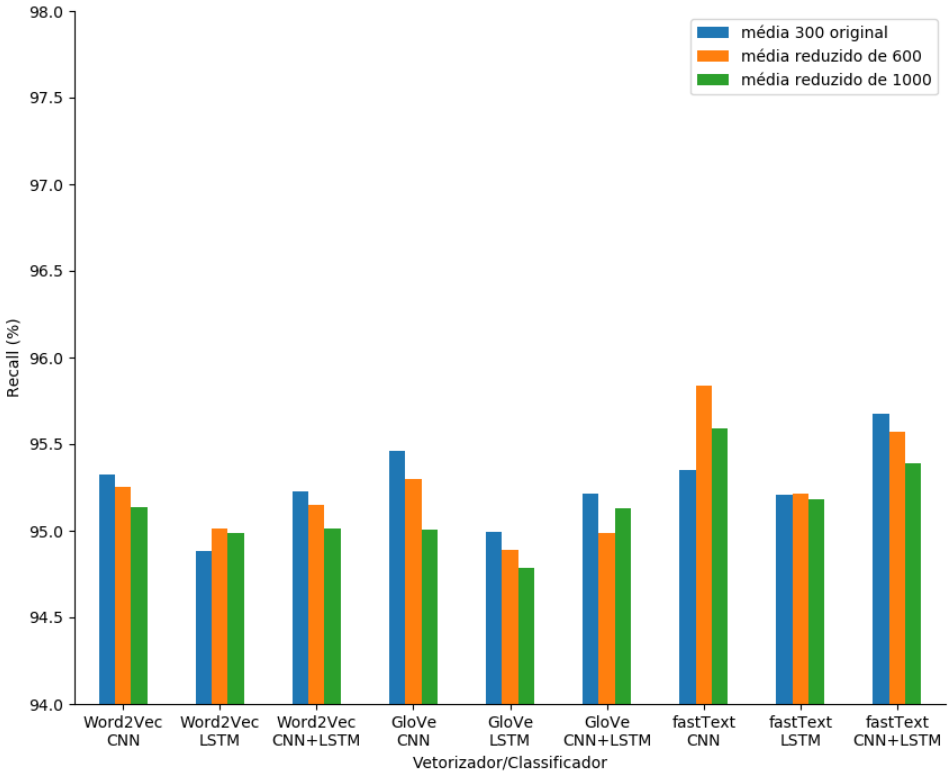

Figura 5.35: Comparação da porcentagem de recall do modelo em 300 dimensões com os modelos reduzidos via PCA, entre as diversas combinações de vetorização e classificação.

Tabela 5.13: Comparação das medidas F1

Vetorizador	Classificador	100 original	300 original	300 reduzido de 1000	200 (Dabiri e Heaslip)
Word2Vec	CNN	95.7%	95.7%	95.6%	98.6%
Word2Vec	LSTM	95.6%	95.6%	95.5%	98.4%
Word2Vec	CNN+LSTM	95.9%	95.9%	95.7%	98.5%
fastText	CNN	95.8%	96.0%	96.0%	98.6%
fastText	LSTM	95.6%	95.9%	95.6%	98.5%
fastText	CNN+LSTM	95.8%	96.0%	95.9%	98.6%

$\frac{tp}{tp+\frac{1}{2}(fp+fn)}$. Dessa forma, pudemos realizar uma comparação indireta, assim como na acurácia, respeitando os mesmos diferenciais. Os resultados foram compilados na tabela 5.13.

Podemos observar com clareza que os resultados da nossa aplicação ficam em média 3% abaixo do modelo original, com ou sem redução dimensional. Assim como na acurácia, isso parece indicar a necessidade de maior afinamento dos parâmetros, de modo a adequar o modelo às características da língua portuguesa.

6. Conclusão

Esta pesquisa estudou o uso de técnicas de deep learning para a representar e classificar microtextos, com respeito à relevância ao domínio de trânsito. Os experimentos realizados objetivaram avaliar modelos bem conhecidos e com aplicações em processamento de linguagem natural, de modo a compreender a aplicabilidade dos modelos à língua portuguesa, e o impacto da variação dimensional nesses modelos.

Com respeito à aplicabilidade em língua portuguesa, verificamos que os modelos apresentam resultados competitivos, com acurácia, precisão e recall em torno de 96%, bastante próximo dos 98% apresentados no trabalho de Dabiri e Heaslip para língua inglesa. Possivelmente, um melhor ajuste paramétrico nas redes classificadoras poderá suprir a diferença apresentada, estudo esse que pretendemos realizar em uma sequência da pesquisa.

Com respeito às combinações representação/classificação, observamos uma ligeira superioridade do par fastText/CNN+LSTM, exatamente como no trabalho de Dabiri e Heaslip. Da mesma forma, o vetorizador GloVe introduzido na pesquisa não apresenta ganhos em relação aos outros. E, finalmente, o uso de CNN puro, junto a quaisquer vetorizadores, apresenta os melhores tempos de treinamento, o que leva a um trade-off entre agilidade no treinamento e resultados. Considerando a pouca diferença em desempenho nas métricas de avaliação, acreditamos que o ganho de tempo seja mais efetivo na otimização do sistema.

Com respeito à sensibilidade à variação dimensional, é efetivamente inócua no tocante às métricas, mas extremamente relevante em relação aos tempos de vetorização e treinamento, com o diferencial que os primeiros se aplicam em qualquer iteração do sistema, mesmo depois de treinadas as redes. Assim, concluímos que o uso das dimensões mais baixas introduzem grande eficiência ao sistema. Também pudemos verificar que a redução artificial das dimensões, via PCA, enquanto que possibilita um aumento da velocidade do sistema, em relação às dimensões originais, também não interfere nos

resultados da classificação.

Com respeito à continuidade da pesquisa, primeiramente planejamos a montagem de um sistema completo de detecção e mapeamento de eventos, incluindo extração de localização dos tweets e posicionamento espaço temporal, com análise da influência dos eventos no trânsito real. O trabalho prévio, desenvolvido na equipe pelos colegas Leonardo Tetéo e Elton Soares, poderá servir de base, ainda que muito de sua codificação precise ser retomada do início.

Além disso, planeja-se estudar a sintonia fina dos classificadores, de modo a aprimorar o desempenho dos modelos existentes em língua portuguesa. Particularmente a variação no número de filtros e janela de convolução, no classificador CNN, deve ter seu impacto mensurado.

Além disso, planejamos estudar o desempenho de outras técnicas, não necessariamente seguindo a arquitetura *deep learning*, tais como aprendizado por reforço, para a classificação de microtexto em português.

Em relação à vetorização, temos como projeto imediato de avaliar o sistema Google BERT[1], e sua aplicação a este projeto.

Outra proposta que temos é levar adiante a redução dimensional, avaliando seu impacto na classificação em espaços vetoriais exíguos, como 10 ou 20 dimensões.

E finalmente, seguindo o trabalho de Bob Sturm [34] na área de inteligência artificial aplicada à música, planejamos utilizar a sua análise das camadas internas da rede no domínio de NLP, estudando quais características dos textos são buscadas para a classificação e o impacto de suas variações.

[1] https://blog.google/products/search/search-language-understanding-bert/

Referências Bibliográficas

[1] Luciana Bencke, Cristian Cechinel, and Roberto Munoz. Automated classification of social network messages into smart cities dimensions. *Future Generation Computer Systems*, 2020.

[2] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media Inc, 2009.

[3] Jason PC Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370, 2016.

[4] Sina Dabiri and Kevin Heaslip. Developing a twitter-based traffic event detection model using deep learning architectures. *Expert systems with applications*, 118:425–439, 2019.

[5] Sina Dabiri, Nikola Marković, Kevin Heaslip, and Chandan K Reddy. A deep convolutional neural network based approach for vehicle classification using large-scale gps trajectory data. *Transportation Research Part C: Emerging Technologies*, 116:102644, 2020.

[6] Sri Krisna Endarnoto, Sonny Pradipta, Anto Satriyo Nugroho, and James Purnama. Traffic condition information extraction & visualization from social media twitter for android mobile application. In *IEEE International Conference on Electrical Engineering and Informatics*, pages 1–4, 2011.

[7] Aldo Gangemi. A comparison of knowledge extraction tools for the semantic web. In *Extended semantic web conference*, pages 351–366. Springer, 2013.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[9] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[10] Nathan S Hartmann, Erick R Fonseca, Christopher D Shulby, Marcos V Treviso, Jéssica S Rodrigues, and Sandra M Aluísio. Portuguese word embeddings: Evaluating on word analogies and natural language tasks. In *Anais do XI Simpósio Brasileiro de Tecnologia da Informação e da Linguagem Humana*, pages 122–131. SBC, 2017.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[12] IBGE. Cidades ibge, 2020 (accessado em 18 de novembro de 2020).

[13] Luis O Jimenez and David A Landgrebe. Supervised classification in high-dimensional space: geometrical, statistical, and asymptotical properties of multivariate data. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 28(1):39–54, 1998.

[14] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition - Second Edition*. Prentice Hall, 2008.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[17] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[18] Linchao Li, Jiasong Zhu, Hailong Zhang, Huachun Tan, Bowen Du, and Bin Ran. Coupled application of generative adversarial networks and conventional neural networks for travel mode detection using gps data. *Transportation Research Part A: Policy and Practice*, 136:282–292, 2020.

[19] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

[20] Rodolfo I. Meneguette, Robson E. De Grande, and Antonio A. F. Loureiro. *Intelligent Transport System in Smart Cities: Aspects and Challenges of Vehicular Networks and Cloud*. Urban Computing. Springer International Publishing, Cham, 2018.

[21] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[22] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. *arXiv preprint arXiv:1712.09405*, 2017.

[23] Jiaqi Mu, Suma Bhat, and Pramod Viswanath. All-but-the-top: Simple and effective postprocessing for word representations. *arXiv preprint arXiv:1702.01417*, 2017.

[24] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[25] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[26] David Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation. *Mach. Learn. Technol.*, 2, 01 2008.

[27] Bayu Yudha Pratama and Riyanarto Sarno. Personality classification based on twitter text using naive bayes, knn and svm. In *2015 International Conference on Data and Software Engineering (ICoDSE)*, pages 170–174. IEEE, 2015.

[28] Vikas Raunak, Vivek Gupta, and Florian Metze. Effective dimensionality reduction for word embeddings. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 235–243, 2019.

[29] Paulo Henrique Lopes Rettore, Igor Araujo, João Guilherme Maia de Menezes, Leandro Villas, and Antonio Alfredo Ferreira Loureiro. Serviço de detecção e enriquecimento de eventos rodoviários baseado em fusão de dados heterogêneos para vanets. In *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 363–376. SBC, 2019.

[30] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, pages 851–860. ACM, 2010.

[31] Neha Sharma, Vibhor Jain, and Anju Mishra. An analysis of convolutional neural networks for image classification. *Procedia Computer Science*, 132:377–384, 2018.

[32] Kaize Shi, Changjin Gong, Hao Lu, Yifan Zhu, and Zhendong Niu. Wide-grained capsule network with sentence-level feature to detect meteorological event in social network. *Future Generation Computer Systems*, 102:323–332, 2020.

[33] Bharath Sriram, Dave Fuhry, Engin Demir, Hakan Ferhatosmanoglu, and Murat Demirbas. Short text classification in twitter to improve information filtering. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 841–842, 2010.

[34] Bob Sturm. What do these 5,599,881 parameters mean?: An analysis of a specific lstm music transcription model, starting with the 70,281 parameters of its softmax layer. In *International Conference on Computational Creativity*, 2018.

[35] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson Education, 2006.

[36] Estevan Teixeira, Pedro Moura, and Carlos Campos. Classificação de tweets sobre trânsito utilizando diferentes técnicas de deep learning. In *Anais Estendidos do X Simpósio Brasileiro de Engenharia de Sistemas Computacionais*, pages 89–96. SBC, 2020.

[37] Fernanda Ferreira Albuquerque Tenorio, Eduarda Chagas, Pedro Barros, and Heitor S Ramos. Detecção de eventos no twitter através de grafos de visibilidade natural. In *Anais do III Workshop de Computação Urbana*, pages 181–193. SBC, 2019.

[38] Leonardo Tetéo. Framework de Extração e Etiquetamento de Informações de Trânsito para Língua Portuguesa, 2017. Monografia (Bacharel em Informática), UNIRIO (Universidade Federal do Estado do Rio de Janeiro), Rio de Janeiro, Brazil.

[39] Leonardo Tetéo, Pedro Moura, Elton F. de S. Soares, and Carlos Alberto V. Campos. Um Framework de Extração e Etiquetamento de Informações de Trânsito. In *18º WPerformance – Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, page 14, 2019.

[40] Twitter. Termos de serviço do twitter. Acessado: 16 de agosto de 2020.

[41] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71):13, 2009.

[42] Senzhang Wang, Xiaoming Zhang, Jianping Cao, Lifang He, Leon Stenneth, Philip S Yu, Zhoujun Li, and Zhiqiu Huang. Computing urban traffic congestions by incorporating sparse gps probe data and social media data. *ACM Transactions on Information Systems (TOIS)*, 35(4):1–30, 2017.

[43] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[44] Zhihao Zheng, Chengcheng Wang, Pu Wang, Yusha Xiong, Fan Zhang, and Yisheng Lv. Framework for fusing traffic information from social and physical transportation data. *PloS one*, 13(8), 2018.