Aplicações de Deep Learning na Classificação de Sentimentos em Tweets

1st João Victor Omena
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brasil
jvrco@cin.ufpe.br

I. Introdução

A análise de sentimentos é uma ferramenta poderosa no campo do processamento de linguagem natural (NLP), amplamente aplicada para extrair opiniões e emoções de grandes volumes de texto. No contexto das redes sociais, como o Twitter, essa análise ganha ainda mais relevância devido ao impacto que essas plataformas têm na formação de opiniões públicas, tendências de mercado e monitoramento de eventos em tempo real. Tweets, por serem curtos, concisos e frequentemente cheios de gírias e abreviações, apresentam desafios únicos para a análise de sentimentos. Contudo, sua análise pode revelar valiosos insights sobre o humor coletivo e as reações a tópicos variados, desde produtos e serviços até eventos políticos.

Este projeto tem como objetivo realizar a análise de sentimentos de um dataset do Twitter Sentiment Analysis utilizando modelos avançados de aprendizado profundo. Explorei várias arquiteturas modernas, como LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit), Bi-LSTM (Bidirectional LSTM), BERT (Bidirectional Encoder Representations from Transformers) e Text-CNN (Convolutional Neural Network). Cada um desses modelos oferece diferentes abordagens para capturar as nuances dos tweets, seja por meio da detecção de padrões espaciais, como nas CNNs, ou da modelagem de dependências temporais e contextuais em sequências de texto, como nas arquiteturas recorrentes e transformadoras.

O uso de múltiplas abordagens tem como objetivo não apenas alcançar uma maior acurácia nas previsões, mas também entender como diferentes arquiteturas performam ao lidar com as particularidades dos tweets, um tipo de dado caracterizado por seu formato curto e informal. Essa comparação detalhada entre modelos oferece uma perspectiva mais ampla sobre os desafios e as melhores práticas para a análise de sentimentos em dados de mídias sociais.

II. Metodologia

A. Conjunto de Dados

O conjunto de dados utilizado neste estudo foi obtido do *Kaggle* e é composto por tweets classificados de acordo com o sentimento expressado. O dataset foi dividido em dois subconjuntos principais: treino e validação, a fim de avaliar o desempenho dos modelos treinados. O conjunto de treino contém aproximadamente 69.000 tweets, enquanto o conjunto de validação contém cerca de 1.000 tweets.

As classes presentes no dataset estão distribuídas em quatro categorias principais:

- **Positivo**: Representa tweets que expressam sentimentos positivos ou de satisfação.
- **Negativo**: Inclui tweets que expressam insatisfação, raiva ou outros sentimentos negativos.
- **Neutro**: Refere-se a tweets sem uma polaridade emocional clara, sendo informativos ou descritivos.
- **Irrelevante**: Tweets que não são relacionados ao contexto de análise de sentimentos, muitas vezes sendo mensagens fora de escopo ou spam.

A distribuição das classes no conjunto de dados é visualizada através de gráficos que ilustram a proporção exata dos dados entre as diferentes classes no conjunto de treino e validação.

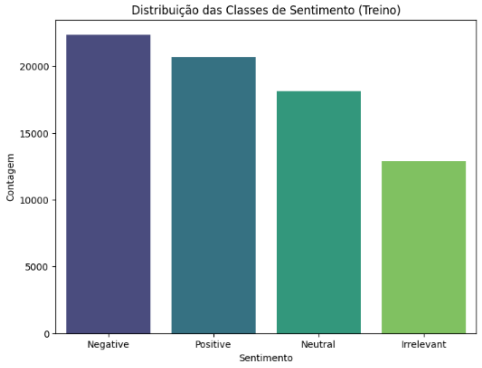

Fig. 1. Dados de Treino

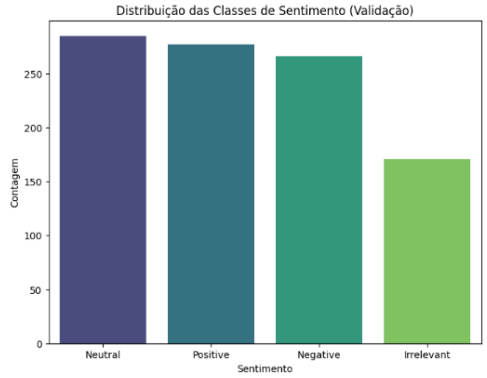

Fig. 2. Dados de Validação

B. Tabela de Exemplo

A tabela a seguir apresenta um exemplo representativo de cada classe do conjunto de dados de tweets:

Classe	Exemplo de Tweet
Positivo	"I just got my new phone, and I absolutely love it! Best purchase ever."
Negativo	"Terrible service at the restaurant today. Not coming back!"
Neutro	"The meeting is scheduled for 2 PM tomorrow. Please be on time."
Irrelevante	"Get your free iPhone now by clicking this link!"

TABLE I
EXEMPLOS CARACTERÍSTICOS DE TWEETS POR CLASSE

C. Limpeza dos Dados

Para garantir a qualidade e consistência dos dados, diversas etapas de pré-processamento foram aplicadas tanto ao conjunto de treino quanto ao de validação.

1) Normalização dos Tweets: Primeiramente, todos os tweets foram convertidos para letras minúsculas, garantindo uniformidade no tratamento das palavras. URLs e menções a outros usuários foram removidas, uma vez que não contribuem para a análise de sentimento. Além disso, caracteres especiais, números e espaços em excesso foram eliminados, mantendo apenas o texto relevante.

2) Remoção de Duplicatas e Dados Faltantes: Após o pré-processamento inicial, foram removidos os tweets duplicados para evitar vieses nos modelos. Em seguida, linhas com tweets ausentes foram excluídas, assegurando que todas as entradas contivessem dados válidos.

3) Estrutura Padronizada do Dataset: Os datasets foram padronizados com as seguintes colunas:

- `ID`: Identificação única do tweet.
- `Category`: Categoria à qual o tweet pertence.
- `Sentiment`: Classificação do sentimento associado ao tweet.
- `Tweet`: O conteúdo do tweet após a limpeza.

A padronização e a limpeza dos dados asseguraram que apenas tweets relevantes e devidamente formatados fossem utilizados nas etapas subsequentes de modelagem.

D. Pré-processamento dos Dados

Após a limpeza dos dados, o próximo passo foi o pré-processamento para preparar os tweets e as classes associadas para os modelos de aprendizado profundo. Esse processo foi dividido em várias etapas:

1) Tratamento de Valores Nulos e Conversão: Primeiramente, os tweets no conjunto de treino e validação foram tratados para substituir valores nulos por strings vazias, garantindo que todos os exemplos tivessem algum conteúdo textual. As classes de sentimentos foram separadas e preparadas para serem transformadas.

2) Codificação dos Sentimentos: Utilizei um *label encoder* para transformar as classes de sentimento em valores numéricos, facilitando o treinamento dos modelos. O *label encoder* foi ajustado ao conjunto de treino, e os valores transformados foram aplicados tanto ao conjunto de treino quanto ao conjunto de validação.

3) Tokenização dos Tweets: Em seguida, os tweets foram convertidos para sequências numéricas utilizando um tokenizador. Este tokenizador foi configurado para considerar um vocabulário de até 10.000 palavras únicas, com uma tokenização especial para palavras fora do vocabulário (*OOV*). As sequências foram geradas para ambos os conjuntos, de treino e validação, representando cada tweet como uma sequência de números inteiros, onde cada número corresponde a um token (palavra).

4) Padronização do Comprimento das Sequências: Para que todas as sequências tivessem o mesmo comprimento, foi definido um limite máximo de 50 tokens por tweet. As sequências menores foram preenchidas com *padding*, garantindo que todas tivessem o mesmo tamanho, facilitando a entrada dos dados nos modelos. O preenchimento foi feito no final da sequência (*post-padding*).

5) Conversão para Tensores: Após a tokenização e padronização das sequências, os dados foram convertidos para tensores utilizando a biblioteca *PyTorch*. Isso incluiu tanto os tweets padronizados quanto as classes de sentimento correspondentes, para os conjuntos de treino e validação.

6) Criação dos DataLoaders: Por fim, criei os *DataLoaders*, responsáveis por fornecer lotes de dados durante o treinamento e validação dos modelos. Defini um tamanho de lote de 64 para o conjunto de treino, com os dados sendo embaralhados a cada época, enquanto o conjunto de validação foi carregado sem embaralhamento, permitindo uma avaliação consistente.

E. Pré-processamento para BERT

No caso do BERT, o pré-processamento dos dados envolveu algumas etapas específicas que diferem dos outros modelos de aprendizado profundo, devido à natureza do modelo transformador e sua necessidade de entradas padronizadas.

1) Tokenização com BERT: Em vez de utilizar um tokenizador padrão, o BERT requer um tokenizador pré-treinado específico para o modelo. Para este projeto, utilizei o `BertTokenizer` da variante `bert-base-uncased`, que

já contém o vocabulário necessário para realizar a tokenização dos textos.

Durante esse processo, os tweets foram convertidos em IDs de tokens baseados no vocabulário do BERT, e foi gerada uma máscara de atenção (*attention mask*) para indicar quais tokens fazem parte da sequência e quais são apenas preenchimentos (*padding*).

2) Truncamento e Padding: O comprimento máximo das sequências foi definido como 128 tokens, e as sequências que excediam esse valor foram truncadas. As sequências menores foram preenchidas com *padding*, garantindo uniformidade no comprimento de todas as entradas. Esse processo é necessário para que o BERT processe as entradas de maneira eficiente.

3) Codificação e Conversão para Tensores: Após a tokenização, os dados de treino e validação foram codificados, produzindo três componentes principais:

- `input_ids`: Os IDs dos tokens que representam cada palavra do tweet.
- `attention_mask`: Máscara binária que indica quais tokens são válidos (1) e quais são preenchimentos (0).
- `labels`: As classes dos sentimentos, codificadas anteriormente.

Esses componentes foram então convertidos para tensores do PyTorch para serem usados no treinamento do modelo BERT.

4) Criação dos DataLoaders: Assim como para os outros modelos, os dados tokenizados foram organizados em *DataLoaders* com lotes de 16 exemplos, para serem processados durante o treinamento e a validação.

F. Introdução aos Modelos Utilizados

Neste projeto, diferentes arquiteturas de redes neurais profundas foram exploradas para a tarefa de análise de sentimentos em tweets, cada uma com características únicas para capturar e interpretar o contexto dos dados de entrada.

Modelos baseados em arquiteturas recorrentes, como LSTM, GRU e Bi-LSTM, são adequados para capturar dependências temporais e sequenciais, sendo eficientes em tarefas de processamento de linguagem natural que envolvem sequências de texto. O LSTM (Long Short-Term Memory) e o GRU (Gated Recurrent Unit) são populares por conseguirem modelar essas dependências em textos mais longos, enquanto o Bi-LSTM adiciona a capacidade de considerar tanto o passado quanto o futuro de uma palavra em uma sequência.

Além disso, abordagens baseadas em convoluções, como a Text-CNN, foram utilizadas para identificar padrões locais nos textos, como n-gramas, oferecendo uma forma alternativa de processamento que se foca mais em padrões curtos e locais.

Por fim, o modelo BERT (Bidirectional Encoder Representations from Transformers), uma das arquiteturas mais modernas, foi utilizado para capturar representações bidirecionais do contexto das palavras em um tweet. O BERT é altamente eficaz para entender relações entre palavras em diferentes partes de uma sequência e, por isso, é amplamente utilizado em tarefas complexas de NLP.

Esses modelos foram escolhidos para oferecer uma visão abrangente de como diferentes arquiteturas de aprendizado profundo performam na tarefa de análise de sentimentos em tweets, cada um com sua capacidade particular de capturar padrões linguísticos e semânticos.

III. Metodologia de Treinamento

Nesta seção, detalho a metodologia de treinamento utilizada para cada modelo de aprendizado profundo, incluindo a configuração dos hiperparâmetros e as técnicas aplicadas para otimizar o desempenho.

A. Modelos LSTM, GRU e Bi-LSTM

Os três modelos recorrentes — LSTM, GRU e Bi-LSTM — compartilham uma arquitetura geral semelhante, com algumas diferenças nas camadas internas e na maneira como capturam dependências temporais nas sequências de palavras. A seguir, destaco os principais pontos compartilhados entre eles:

a) Arquitetura Comum:

- **Camada de Embedding**: Todos os modelos usam uma camada de embedding para transformar os tokens dos tweets em vetores densos de dimensão 64, permitindo ao modelo capturar semelhanças semânticas entre palavras.
- **Camada Recurrente**:
  - O **LSTM** utiliza uma camada LSTM com 128 unidades ocultas, que captura dependências temporais ao longo da sequência.
  - O **GRU** emprega uma camada GRU com 128 unidades ocultas, similar ao LSTM, mas com uma estrutura mais simples e menos intensiva computacionalmente.
  - O **Bi-LSTM** é uma versão bidirecional do LSTM, processando as sequências em duas direções (do início ao fim e do fim ao início), o que dobra a dimensão da saída.
- **Camada Linear (Fully Connected)**: A saída final dos modelos é passada por uma camada linear com 4 unidades, correspondendo às 4 classes de sentimento (Positivo, Negativo, Neutro e Irrelevante).

b) Função de Custo e Otimização: Para todos os modelos, foi utilizada a função de perda `CrossEntropyLoss`, adequada para classificação multiclass, e o otimizador `Adam` com uma taxa de aprendizado (*learning rate*) de 0.001.

c) Critérios de Parada: A técnica de *early stopping* foi aplicada em todos os modelos para evitar o sobreajuste, com um critério de melhora mínima (*min_delta*) de 0.001 na perda de validação e paciência de 3 épocas consecutivas sem melhora.

d) Métricas de Avaliação: Os modelos foram avaliados com base nas seguintes métricas: Acurácia, F1-Score, Precisão e Recall, tanto no conjunto de treino quanto no de validação. As métricas foram monitoradas em todas as épocas para garantir que os modelos estivessem generalizando bem.

B. Text-CNN (Convolutional Neural Network para Texto)

O modelo Text-CNN segue uma abordagem diferente, utilizando convoluções para capturar padrões locais nos textos:

a) Arquitetura do Modelo:

- **Camada de Embedding**: Assim como os modelos recorrentes, a Text-CNN usa uma camada de embedding para transformar os tokens em vetores densos.
- **Camadas Convolucionais**: A Text-CNN aplica convoluções com diferentes tamanhos de filtros (*kernel sizes* de 3, 4 e 5), cada um com 100 filtros, permitindo que o modelo capture padrões de n-gramas de diferentes comprimentos.
- **Max-Pooling**: Após as convoluções, a técnica de max-pooling é usada para reduzir a dimensionalidade e manter as informações mais relevantes.
- **Camada Linear (Fully Connected)**: As saídas das camadas convolucionais são concatenadas e passadas por uma camada totalmente conectada com 4 unidades de saída, correspondendo às classes de sentimento.

b) Função de Custo e Otimização: A função de perda foi a mesma (`CrossEntropyLoss`), assim como o otimizador `Adam` com taxa de aprendizado de 0.001.

c) Critérios de Parada e Avaliação: Assim como nos modelos recorrentes, utilizei *early stopping* com os mesmos critérios de parada. As métricas avaliadas também foram Acurácia, F1-Score, Precisão e Recall, e o treinamento foi interrompido na 8ª época.

C. BERT (Bidirectional Encoder Representations from Transformers)

O modelo BERT foi utilizado para capturar o contexto de palavras nos tweets de maneira bidirecional, analisando tanto o contexto anterior quanto o posterior das palavras. A seguir estão os detalhes da arquitetura e metodologia de treinamento deste modelo.

a) Arquitetura do Modelo: O modelo utilizado foi o `BertForSequenceClassification` da *Hugging Face*, pré-treinado com o `bert-base-uncased`. Ele foi ajustado para a classificação de quatro classes de sentimento (Positivo, Negativo, Neutro e Irrelevante) utilizando o parâmetro `num_labels=4`. A arquitetura BERT é composta por múltiplas camadas de atenção que permitem ao modelo aprender representações ricas e contextualizadas de cada palavra.

A saída final de BERT, com 768 dimensões, é passada por uma camada linear que gera a classificação para as quatro classes. O *dropout* foi configurado em 0.1 para evitar o overfitting.

b) Função de Custo e Otimização: A função de perda escolhida foi a `CrossEntropyLoss`, apropriada para tarefas de classificação. O otimizador utilizado foi o `AdamW`, com uma taxa de aprendizado (*learning rate*) de 5e-5. Também foi utilizado um *scheduler* de aprendizado linear com aquecimento de 0 (*warm-up steps*) para melhor ajuste do gradiente durante o treinamento.

c) Critérios de Parada: Implementei o *early stopping* com uma paciência de 2 épocas consecutivas sem melhora na perda de validação e uma melhoriana perda de validação e uma melhoria mínima (*min_delta*) de 0.001. O treinamento foi interrompido na 4ª época com base nesses critérios.

d) Métricas de Avaliação: As métricas monitoradas durante o treinamento incluíram Acurácia, F1-Score, Precisão e Recall, tanto no conjunto de treino quanto no de validação. Essas métricas foram calculadas em cada época para garantir que o modelo estivesse aprendendo adequadamente e generalizando bem.

IV. Resultados dos Modelos

Nesta seção, são apresentados os resultados de cada modelo. Todos os modelos foram treinados utilizando uma GPU T4, o que acelerou significativamente o processo de treinamento. Embora o modelo **BERT** tenha obtido o melhor desempenho em termos de acurácia e outras métricas, ele levou cerca de 1 hora para completar 4 épocas de treinamento. Em contrapartida, os demais modelos (LSTM, GRU, Bi-LSTM e Text-CNN) levaram entre 5 e 10 minutos para completar todo o treinamento, com o *early stopping* sendo ativado na 8ª época para o Text-CNN.

A. Resultados do LSTM

O modelo LSTM apresentou boa generalização e foi interrompido na 10ª época devido ao critério de *early stopping*. A acurácia de validação começou em 0.4910 e subiu para 0.9337. A perda de validação caiu de 1.1946 para 0.2021. Os gráficos a seguir mostram a evolução da acurácia, F1-Score, precisão, recall e perda durante o treinamento.

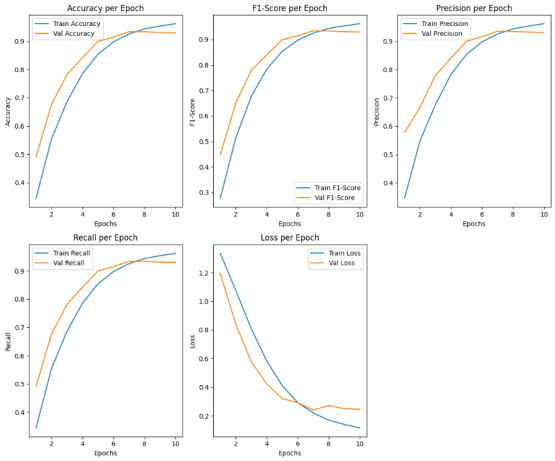

Fig. 3. Evolução das métricas do modelo LSTM

B. Resultados do GRU

O modelo GRU foi interrompido na 10ª época, após começar com uma acurácia de validação de 0.6376 e alcançar 0.9441. O F1-Score de validação foi de 0.9418 e a perda de validação caiu de 0.9238 para 0.2629.

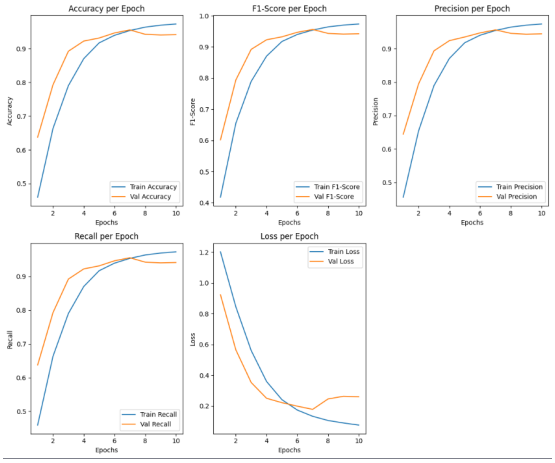

Fig. 4. Evolução das métricas do modelo GRU

C. Resultados do Bi-LSTM

O modelo Bi-LSTM alcançou uma acurácia de validação de 0.9378 na 10ª época. O F1-Score de validação também foi de 0.9378 e a perda de validação caiu de 1.1031 para 0.2487.

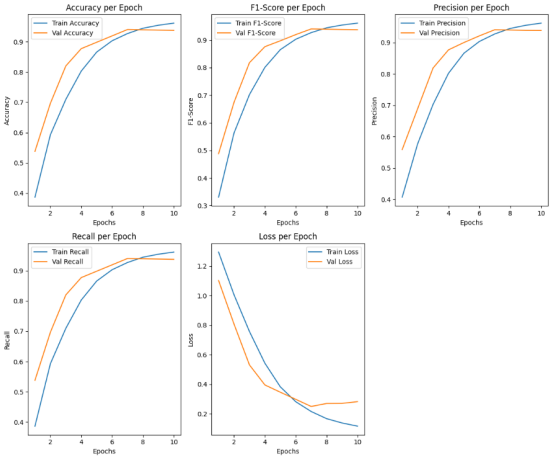

Fig. 5. Evolução das métricas do modelo Bi-LSTM

D. Resultados do Text-CNN

O modelo Text-CNN apresentou uma acurácia inicial de validação de 0.8082 e atingiu 0.9528 na 8ª época. O F1-Score de validação foi de 0.9528 e a perda de validação caiu de 0.6111 para 0.1800.

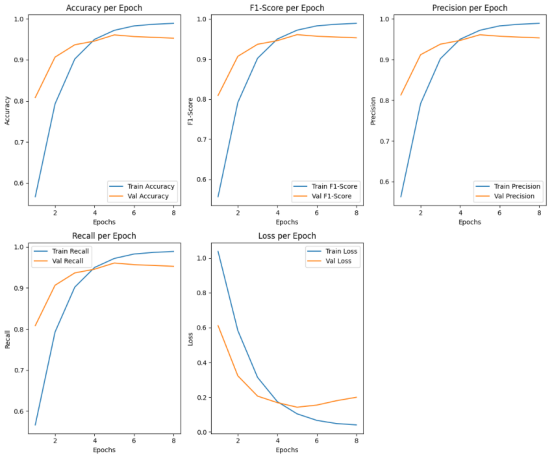

Fig. 6. Evolução das métricas do modelo Text-CNN

E. Resultados do BERT

O modelo BERT apresentou os melhores resultados, alcançando uma acurácia de validação de 0.9790 e um F1-Score de 0.9791 na 4ª época. A perda de validação caiu de 0.1678 para 0.1097. No entanto, o tempo de treinamento foi significativamente maior, levando cerca de 1 hora para completar 4 épocas.

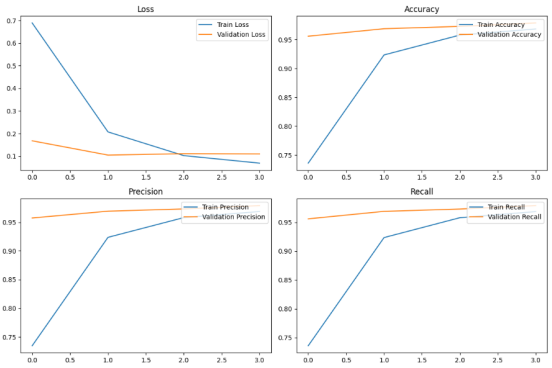

Fig. 7. Evolução das métricas do modelo BERT

V. Comparação dos Resultados

A tabela a seguir resume os melhores resultados obtidos para cada modelo, considerando acurácia, F1-Score, precisão, recall e perda de validação.

TABLE II
Comparação dos Resultados dos Modelos na Melhor Época

Modelo	Época	Acurácia	F1-Score	Precisão	Recall	Perda de Valid
LSTM	10	0.9337	0.9297	0.9337	0.9297	0.2021
GRU	10	0.9441	0.9418	0.9441	0.9418	0.2629
Bi-LSTM	10	0.9378	0.9378	0.9378	0.9378	0.2487
Text-CNN	8	0.9528	0.9528	0.9528	0.9528	0.1800
BERT	4	0.9790	0.9791	0.9791	0.9790	0.1097

A. Análise Comparativa

Os resultados mostram distinções claras entre o desempenho e o custo computacional dos diferentes modelos empregados

para a tarefa de análise de sentimentos em tweets. Cada modelo, com sua arquitetura única, oferece vantagens e desvantagens.

O **BERT** apresentou a maior acurácia (97.90%) e o melhor F1-Score (97.91%). Esse desempenho pode ser atribuído à sua capacidade de capturar dependências bidirecionais no texto, permitindo uma compreensão mais profunda do contexto de cada palavra em relação às outras. O mecanismo de atenção utilizado pelo **BERT** é altamente eficaz para tarefas como a análise de sentimentos, onde a relação entre palavras em diferentes partes do texto é crucial. No entanto, essa superioridade veio acompanhada de um custo computacional elevado, uma vez que o modelo levou cerca de 1 hora para completar apenas 4 épocas de treinamento. Esse tempo é significativamente maior comparado aos 5-10 minutos necessários para os outros modelos. Esse tempo prolongado deve-se à complexidade do mecanismo de atenção e ao grande número de parâmetros que o **BERT** utiliza, tornando-o menos viável em cenários com recursos computacionais ou tempo limitados.

Em contrapartida, o **Text-CNN** alcançou uma acurácia de 95.28% e um F1-Score de 95.28%, com um custo computacional muito mais baixo. O modelo se beneficia de sua capacidade de detectar padrões locais no texto, como n-gramas, utilizando diferentes tamanhos de filtros nas camadas convolucionais. Essa abordagem o torna eficiente para lidar com textos curtos, como tweets, onde padrões locais são suficientes para capturar o significado. Além disso, o **Text-CNN** conseguiu completar seu treinamento em apenas 8 épocas, com o critério de **early stopping** sendo ativado rapidamente, tornando-o uma escolha ideal para situações em que eficiência e rapidez são necessárias, embora não capture o contexto mais profundo que o **BERT** oferece.

Os modelos recorrentes — **LSTM**, **GRU** e **Bi-LSTM** — também apresentaram bom desempenho, com acurácias variando entre 93.37% e 94.41%. Dentre eles, o **GRU** teve o melhor resultado, alcançando 94.41% de acurácia e 94.18% de F1-Score. Sua arquitetura simplificada, em comparação com o **LSTM**, reduz a complexidade computacional, o que pode ter contribuído para esse desempenho. Apesar de possuir menos parâmetros, o **GRU** consegue capturar bem as dependências temporais no texto, tornando-o eficiente para sequências curtas, como tweets. O **Bi-LSTM** também apresentou bons resultados, alcançando uma acurácia de 93.78%, beneficiando-se da capacidade de processar a sequência em ambas as direções (frente e verso). No entanto, essa bidirecionalidade adiciona um nível extra de complexidade, resultando em um treinamento mais demorado em comparação com o **GRU**.

Já o **LSTM**, embora eficaz, apresentou o menor desempenho entre os modelos recorrentes, com uma acurácia de 93.37%. Sua arquitetura mais complexa, com múltiplos mecanismos de controle para lidar com dependências de longo prazo, pode ter contribuído para essa leve queda de desempenho. Além disso, o **LSTM** é mais propenso ao **overfitting** em textos curtos, como os tweets, onde as dependências de longo prazo não são tão significativas quanto em textos mais longos.

B. Trade-offs

As trocas entre esses modelos são claras:

- O **BERT** oferece a melhor acurácia e compreensão contextual, graças ao seu mecanismo de atenção bidirecional. No entanto, seu custo computacional elevado torna-o uma escolha menos viável para aplicações em tempo real ou em ambientes com recursos limitados.

- O **Text-CNN**, apesar de não ser tão robusto na captura de contexto quanto o **BERT**, oferece um excelente equilíbrio entre desempenho e eficiência computacional. É mais rápido e econômico, tornando-se uma escolha sólida para aplicações práticas que exigem respostas rápidas.

- Os modelos recorrentes **LSTM**, **GRU** e **Bi-LSTM** se encontram no meio-termo entre essas duas abordagens. Enquanto capturam dependências temporais de forma eficiente, eles não superam o **BERT** em termos de acurácia, nem competem com o **Text-CNN** em termos de rapidez no treinamento. Entre eles, o **GRU** se destaca como a melhor opção, oferecendo bom desempenho com menor custo computacional.

C. Conclusão

Concluindo, o **BERT** é o modelo com o melhor desempenho em termos de acurácia e compreensão do contexto, mas seu custo computacional elevado é um fator limitante. O **Text-CNN** apresenta um equilíbrio excelente entre precisão e eficiência, sendo a escolha mais prática para a maioria das aplicações. Já os modelos recorrentes, embora eficazes, ficam atrás do **Text-CNN** em velocidade e do **BERT** em capacidade de compreensão contextual. Em ambientes com recursos limitados, o **GRU** se destaca como uma boa escolha, oferecendo um desempenho sólido com menor custo computacional.

ACKNOWLEDGMENT

REFERENCES