EP 02 - MAC5725 - Linguística Computacional

Calebe Rezende[1]

[1] Instituto de Matemática -- Universidade de São Paulo (USP)

[2]Departamento de Sistemas e Computação

calebe.rezende@usp.br

Abstract. *The objective of this article is to present the refinement of encoders in the architecture Transformers (BERT-like, using: neural network finetuning BERTimbau, With a regression task using a variation of BERT, we predicted the density of vowels in each sentence, this calculation was done in balanced and unbalanced tasks.*

Resumo. *O objetivo deste artigo é apresentar o refinamento de encoders na arquitetura Transformers (BERT-like, usando o: refinamento (finetuning) da rede neural BERTimbau, Com uma tarefa de regressão usando uma variação do BERT, prevemos a densidade de vogais em cada sentença, esse cálculo foi feito em tarefas balanceadas e desbalanceadas*

1. Introdução

Este artigo, relacionado ao trabalho de linguística computacional MAC5725, discute a implementação da arquitetura Transformers (BERT-like), usando o refinamento (finetuning) da rede neural Bertimbau, o nosso treinamento prevê a densidade das vogais nas sentenças. Os dados utilizados para isso é o banco de dados da B2W-Reviews o qual se trata de um corpus aberto de análises de produtos com mais de 130 mil avaliações coletadas no site Americanas. A tarefa de regressão não é algo usual com modelos de Bert, por isso há uma tentativa nesse artigo de investigarmos possibilidades de melhorias. A tarefa de regressão que iremos prever com o uso de redes Neurais Transformers, é a densidade de vogais em uma sentença.

2. Metodologia

Para tarefa de regressão, foi utilizado o refinamento (finetuning) da rede neural Bertimbau, obtida a partir do site Huggingface. A metodologia utilizada neste código envolve a criação de modelos de regressão e classificação baseados na arquitetura BERT (Bidirectional Encoder Representations from Transformers) para tarefas específicas em processamento de linguagem natural, foi implementado o modelo `BertForVowelDensityRegression`. Esse modelo utiliza a arquitetura BERT pré-treinada (`neuralmind/bert-base-portuguese-cased`) e adiciona uma camada linear (`regression`) para realizar a regressão. A saída é uma pontuação contínua que representa a densidade de vogais, usando a representação obtida pelo BERT.

Além disso, foram implementados dois modelos para tarefas de classificação. O primeiro, `BertForQuantizedClassification`, é projetado para classificação quantizada em um

número específico de classes. Ele incorpora uma camada de dropout para regularização, seguida por uma camada linear (`classification'') para a classificação.

O segundo modelo, `BertForBalancedClassification`, também é destinado à classificação, mas com ênfase na gestão de classes balanceadas. Ele utiliza a arquitetura BERT, seguida por uma camada linear, e aplica uma camada softmax para calcular as probabilidades de classe. Isso é usado em problemas de classificação onde as classes têm uma distribuição desigual.

Ambos os modelos de classificação (`BertForQuantizedClassification` e `BertForBalancedClassification`) retornam as saídas logits para as classes e as probabilidades associadas. Esses modelos fornecem uma abordagem eficaz para tarefas de aprendizado supervisionado em que a compreensão contextual de textos em português é crucial.

No mais, foram separados em tarefas os testes, na Tarefa 1, o foco foi na regressão da densidade de vogais em um trecho de texto. A densidade de vogais (DV) é uma grandeza numérica obtida ao considerar apenas os caracteres correspondentes às letras do alfabeto português, contando o número de vogais com ou sem acento e dividindo pelo número total de letras. Dessa forma, foram ignorados espaços, pontuações, dígitos e outros caracteres não-letra. O objetivo foi prever essa densidade utilizando redes neurais Transformers.

Nas Tarefas 2 e 3, o enfoque foi na quantização da densidade de vogais. Foram definidas três categorias para classificar a densidade de vogais em uma sentença. Na Tarefa 2, as classes não são balanceadas, com a classe 2 provavelmente contendo a maioria das sentenças do corpus. Na Tarefa 3, porém, foram criadas três classes balanceadas, garantindo que o número de sentenças em cada classe seja um terço do total do corpus. Em ambas as tarefas, foram avaliadas métricas como acurácia total, acurácia por classe, sensibilidade e especificidade para cada uma das três classes. A análise dessas métricas permitiu a comparação entre as avaliações das duas tarefas. Essa abordagem visa oferecer uma compreensão mais aprofundada da capacidade dos modelos em lidar com diferentes aspectos da classificação quantizada da densidade de vogais.

3. Pré-processamento

O sucesso de modelos de aprendizado de máquina frequentemente depende da qualidade e da preparação adequada dos conjuntos de dados utilizados. Nesta seção, descrevemos em detalhes o processo de pré-processamento aplicado ao conjunto de dados inicial, visando otimizar sua utilidade para as tarefas específicas propostas.

3.1 Carregamento e Divisão Inicial do Conjunto de Dados

Inicialmente, o conjunto de dados foi carregado a partir da biblioteca `pandas`. A fim de facilitar a avaliação do desempenho do modelo, o conjunto de dados foi dividido em conjuntos de treino, validação e teste, seguindo uma proporção de 80%, 10% e 10%, respectivamente. Essa divisão foi realizada utilizando a função `train_test_split` da biblioteca `scikit-learn`.

3.2 Modelo de Regressão: Densidade de Vogais

Para a tarefa de regressão da densidade de vogais, foram criadas novas features a partir da coluna de texto original. A coluna 'review_text' foi renomeada para 'texto' e limitada às

primeiras 10.000 linhas do conjunto de dados então, a densidade de vogais foi calculada para cada texto, considerando apenas letras do alfabeto português.

3.3 Tarefas de Classificação Quantizada

Para as tarefas de classificação quantizada, foram definidas categorias baseadas na densidade de vogais. A Tarefa 2 envolveu a classificação em três classes, enquanto a Tarefa 3 buscou criar classes balanceadas. O conjunto de dados foi novamente dividido, garantindo que as proporções desejadas fossem mantidas em cada conjunto de treino, validação e teste.

4.Evidências

	Baseline	RMSE	MAE	MAPE	R2	Pearson Correlation
0	densidade_corpus	0.039488	0.024089	1.792957e+12	0.000000	NaN
1	densidade_primeira_palavra	0.222531	0.146373	3.127098e-01	-30.758713	0.233098
2	densidade_ultima_palavra	0.148383	0.093810	1.958609e-01	-13.120397	0.271428

Imagem 1: Insights sobre a eficácia dessas estratégias. Imagem Própria

As primeiras funções foram definidas para calcular métricas de avaliação, calcular a densidade de vogais para a primeira e última palavra de cada sentença, e avaliar baselines, como a densidade média do corpus e densidades baseadas em palavras específicas. A função principal orquestrou o fluxo, resultando na exibição e salvamento de métricas de avaliação para os baselines considerados, foram geradas métricas de avaliação para as diferentes abordagens de baseline, fornecendo insights sobre a eficácia dessas estratégias na previsão da densidade de vogais.

4.1 Interpretações

Os resultados da avaliação dos baselines para a previsão da densidade de vogais revelam insights valiosos sobre o desempenho relativo dessas estratégias. Primeiramente, ao considerar a densidade média do corpus como baseline, observamos um RMSE (Root Mean Squared Error) de 0.0395, MAE (Mean Absolute Error) de 0.0241 e MAPE (Mean Absolute Percentage Error) absurdamente elevado, indicando que essa abordagem pode não ser a mais adequada para a tarefa em questão. Em contraste, ao avaliar as densidades da primeira e última palavra como baselines, notamos que embora o RMSE e o MAE sejam mais altos em comparação com o baseline do corpus, o MAPE é consideravelmente menor. No entanto, é importante ressaltar que ambos os baselines baseados em palavras apresentam um R^2 negativo, indicando uma performance inferior à simples utilização da média do corpus. A correlação de Pearson, por sua vez, sugere uma relação moderada entre as predições e os valores reais, com a densidade da última palavra demonstrando a correlação mais robusta. Esses resultados destacam a complexidade da tarefa de predição da densidade de vogais e a necessidade de abordagens mais sofisticadas para melhorar o desempenho do modelo.

4. Tarefa 1

O código implementa um conjunto de funcionalidades essenciais, incluindo um dataset personalizado (`TextDataset`) para organizar os dados de treino e validação, além de funções de treinamento e avaliação que empregam dataloaders e métricas como erro médio quadrático (`MSELoss`). O modelo é treinado ao longo de múltiplas épocas, e medidas de early stopping são implementadas para evitar o overfitting. O script também salva o modelo treinado e gera um gráfico representando as curvas de perda durante o processo de treinamento, fornecendo uma visão abrangente do desempenho do modelo ao longo do tempo. Essa abordagem representa uma aplicação da arquitetura BERT em uma tarefa específica, destacando a importância de estratégias de treinamento e avaliação para a obtenção de resultados eficazes.

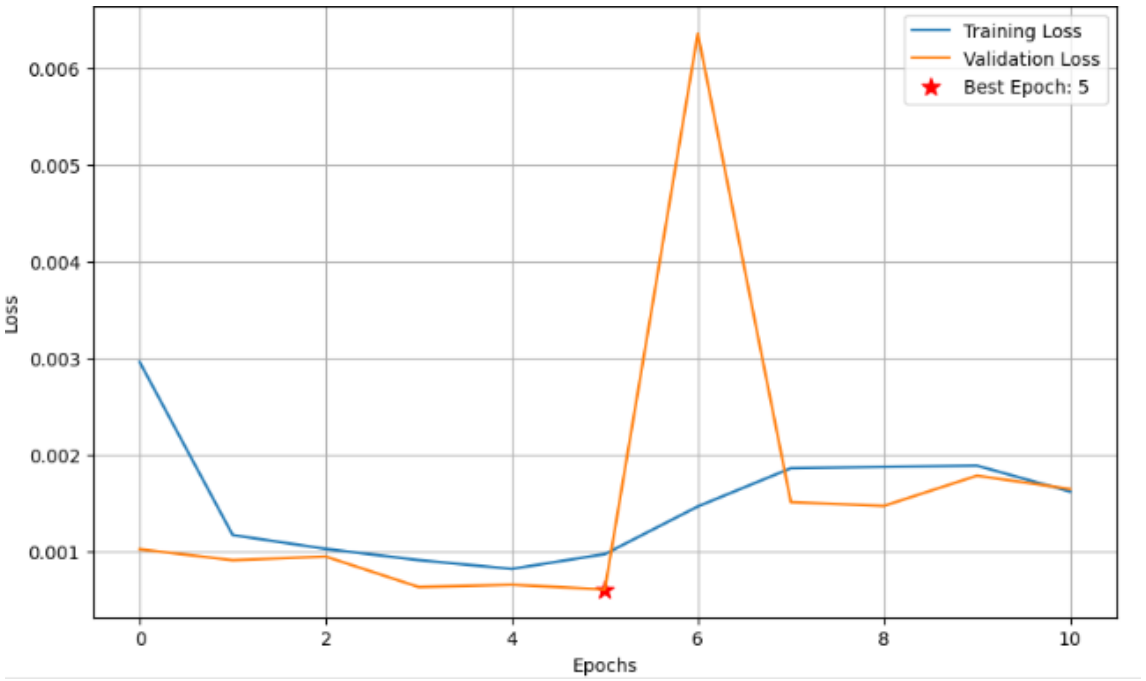


Imagem 2: Training and Validation Loss. Imagem Própria

Os resultados são promissores, inicialmente, observamos uma diminuição consistente na perda de treinamento e validação ao longo das primeiras épocas, indicando uma convergência do modelo. O modelo alcança seu melhor desempenho na época 5, com uma perda de validação mínima de 0.0006. Entretanto, na época 7, observamos um aumento significativo na perda de validação para 0.0064, o que levou ao acionamento do mecanismo de *early stopping* após 11 épocas. A redução da taxa de aprendizado após a décima época sugere uma tentativa de ajustar a convergência do modelo em uma fase mais estável. Essa análise destaca a importância do monitoramento cuidadoso do desempenho do modelo durante o treinamento, permitindo ajustes adequados para evitar o overfitting e otimizar o desempenho preditivo. A imagem 2 exibida, oferece uma visualização dessas dinâmicas, destacando as curvas de perda ao longo das épocas e indicando o ponto ótimo de parada.

4. Tarefa 2

Nesta tarefa, são definidas funções auxiliares, como `define_class_labels` para atribuição de rótulos de classe com base em critérios específicos de densidade de vogais. Os dados de treino e validação são lidos a partir de arquivos CSV e pré-processados utilizando a função `define_class_labels`. Além disso, conjuntos de dados e carregadores de dados são criados para facilitar o treinamento do modelo. O modelo escolhido, denominado `BertForQuantizedClassification`, é configurado com critérios de perda (CrossEntropyLoss), otimizador (Adam) e um scheduler para ajuste dinâmico da taxa de aprendizado.

O treinamento ocorre ao longo de 10 épocas, onde o modelo é ierativamente treinado e avaliado. Durante o treinamento, métricas de desempenho, como perda, acurácia, precisão, recall e F1-score, são exibidas a cada época. O código incorpora também um mecanismo de salvamento do modelo com melhor desempenho na validação, garantindo a retenção do estado ótimo. Os resultados são visualizados por meio de um gráfico que representa as curvas de treinamento e validação em termos de perda. Este código oferece uma implementação robusta e compreensiva de treinamento de modelos de classificação de texto, proporcionando uma base sólida para pesquisadores e desenvolvedores que buscam explorar e aprimorar modelos de aprendizado profundo para tarefas similares.

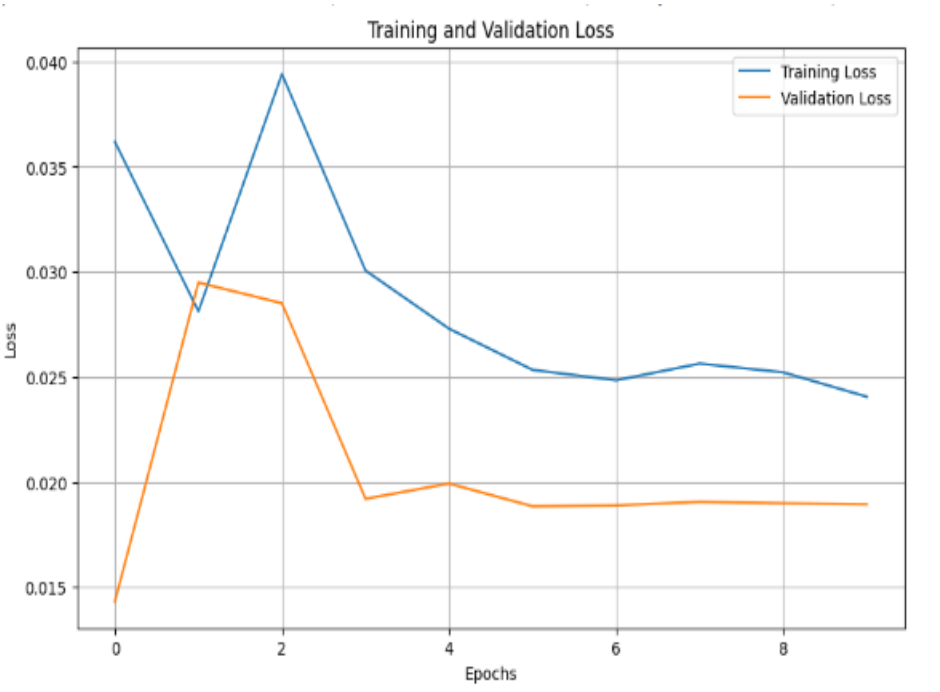
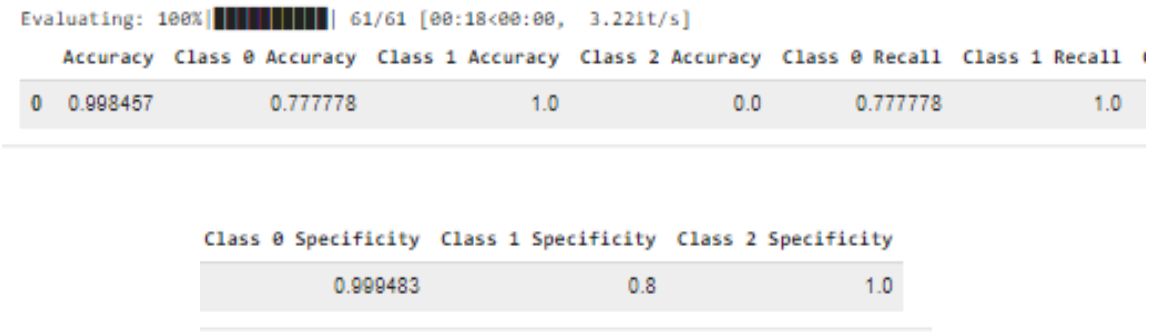


Imagem 3: Training and Validantion Loss, task 2; Imagem própria

Os resultados obtidos durante o treinamento e avaliação do modelo de classificação de texto são apresentados ao longo de diversas épocas. Durante a primeira época, observou-se uma perda de treinamento de 0.0362 e uma perda de validação de 0.0143, com uma acurácia impressionante de aproximadamente 99,79%. Métricas adicionais, como precisão, recall e F1-score, foram calculadas para fornecer uma avaliação mais abrangente do desempenho do modelo. Nos treinamentos subsequentes, essas métricas foram consistentemente mantidas em níveis elevados, demonstrando a eficácia do modelo. É importante notar que foram emitidos

avisos relacionados à métrica de precisão, indicando indefinição e ajuste para 0.0 em labels sem amostras previstas. Este aviso sugere a possibilidade de desbalanceamento nas classes ou amostras insuficientes para algumas categorias, aspecto que poderia ser investigado para aprimorar a robustez do modelo em cenários específicos.

Além disso, o código incorpora um mecanismo inteligente de ajuste dinâmico da taxa de aprendizado, reduzindo-a em épocas específicas caso a perda na validação não melhore. O salvamento do modelo ocorre quando uma perda de validação menor é alcançada, destacando uma abordagem eficiente para manter o melhor desempenho do modelo ao longo do treinamento. O gráfico apresenta as curvas de treinamento e validação ao longo das épocas, proporcionando uma representação gráfica do processo de treinamento do modelo. Em conjunto, esses resultados fornecem uma visão abrangente do desempenho e da adaptação do modelo ao conjunto de dados, com potenciais insights para utilizações futuras.



Evaluating: 100%|██████████| 61/61 [00:18<00:00, 3.22it/s]

	Accuracy	Class 0 Accuracy	Class 1 Accuracy	Class 2 Accuracy	Class 0 Recall	Class 1 Recall
0	0.998457	0.777778	1.0	0.0	0.777778	1.0

Class 0 Specificity	Class 1 Specificity	Class 2 Specificity
0.999483	0.8	1.0

Imagem 4: Primeiras comparações. Imagem própria

5. Tarefa 3

O código da tarefa 2 código apresenta melhorias substanciais em relação ao primeiro, notadamente na abordagem da definição dos rótulos de classe e na avaliação do modelo. Em contraste com o primeiro código, a função `define_class_labels` foi aprimorada para garantir uma distribuição equilibrada dos rótulos entre as classes, assegurando uma abordagem mais justa na atribuição de classes. Além disso, o segundo código incorpora métricas de avaliação mais abrangentes, como precisão, recall e F1-score, através de funções do scikit-learn, fornecendo uma visão mais completa do desempenho do modelo. O processo de treinamento foi refinado com a utilização do otimizador Adam, um organizador de taxa de aprendizado e a implementação do salvamento do modelo com base na menor perda de validação.

Os resultados apresentados durante o treinamento e avaliação do modelo indicam que, ao longo das 10 épocas, houve uma estagnação no desempenho, com perdas de treinamento (Train Loss) e perdas de validação (Val Loss) mantendo-se em níveis próximos, indicando uma possível convergência limitada do modelo. A acurácia do modelo, representada pela métrica "Accuracy," permanece relativamente constante em torno de 33%, sugerindo um desempenho insatisfatório na tarefa de classificação. Além disso, a ocorrência de avisos relacionados à métrica de precisão (Precision) indica que a precisão está indefinida em algumas classes devido à ausência de amostras previstas para essas classes. Isso sugere que o modelo pode estar enfrentando dificuldades em aprender representações discriminativas para todas as classes, possivelmente devido a desequilíbrios nos dados ou a uma arquitetura de modelo inadequada.

A redução da taxa de aprendizado em diferentes épocas (por exemplo, na época 6 e na época 10) indica uma tentativa de ajustar a taxa de aprendizado dinamicamente para melhorar a convergência, mas isso não parece ter um impacto significativo no desempenho geral.

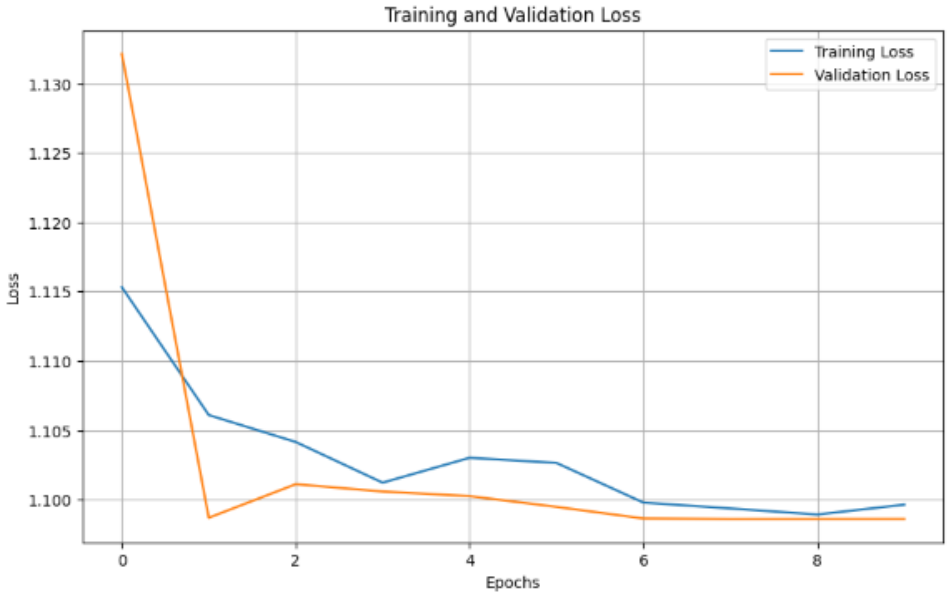


Imagem 5: Tarefa 3. Imagem própria

6. Comparações entre tarefas

A comparação realiza operações essenciais para comparar métricas de avaliação entre duas tarefas distintas em um ambiente de aprendizado de máquina. Inicialmente, são importadas as bibliotecas necessárias, incluindo o pandas para manipulação de dados. Em seguida, são carregados dois conjuntos de dados, 'task2' e 'task3', cada um contendo métricas de avaliação específicas para uma tarefa de classificação.

A etapa subsequente envolve a concatenação dos DataFrames 'task2' e 'task3', resultando no DataFrame 'compare'. Essa concatenação permite uma comparação direta entre as métricas das duas tarefas. Para organizar visualmente as informações, as colunas do DataFrame são reorganizadas, garantindo que a coluna 'Task' seja posicionada no início. O código então realiza a transposição do DataFrame 'compare', transformando as métricas em linhas e as tarefas em colunas.

O resultado final, representado pelo DataFrame 'compare_transposed', oferece uma visão comparativa clara das métricas de avaliação entre as duas tarefas. Este DataFrame transposto é exibido para facilitar a análise das métricas específicas de cada tarefa e auxiliar na compreensão do desempenho relativo dos modelos em diferentes contextos. Essa abordagem sistemática de comparação de métricas proporciona insights valiosos sobre a eficácia dos modelos treinados em tarefas específicas de classificação, informando decisões e refinamentos futuros no processo de aprendizado de máquina.

	0	1
Task	TAREFA 2	TAREFA 3
Accuracy	0.998457	0.333333
Class 0 Accuracy	0.777778	0.0
Class 1 Accuracy	1.0	0.0
Class 2 Accuracy	0.0	1.0
Class 0 Recall	0.777778	0.0
Class 1 Recall	1.0	0.0
Class 2 Recall	0.0	1.0
Class 0 Specificity	0.999483	1.0
Class 1 Specificity	0.8	1.0
Class 2 Specificity	1.0	0.0

Imagem 6: Matriz Transposta de Comparações.. Imagem própria

7. Conclusões

A análise comparativa dos resultados entre as duas tarefas, TAREFA 2 e TAREFA 3, revela diferenças significativas em termos de desempenho de classificação. A métrica de acurácia (Accuracy) para a TAREFA 2 é notavelmente alta, atingindo 99.85%, enquanto a TAREFA 3 apresenta uma acurácia mais modesta de 33.33%. Esse contraste sugere que o modelo treinado para a TAREFA 2 é muito mais preciso em suas previsões em comparação com o modelo da TAREFA 3. A análise por classe destaca padrões distintos. Na TAREFA 2, todas as classes (Class 0, Class 1 e Class 2) apresentam altas pontuações de acurácia e recall, indicando que o modelo é capaz de identificar corretamente os exemplos de cada classe. No entanto, para a TAREFA 3, notamos que a Class 0 não tem recall ou acurácia, indicando uma dificuldade do modelo em reconhecer corretamente os exemplos dessa classe.

A métrica de especificidade (Specificity) também fornece insights. Na TAREFA 2, as Classes 0 e 2 têm especificidades próximas de 1.0, indicando que o modelo tem uma capacidade excepcional de identificar verdadeiros negativos para essas classes. Por outro lado, na TAREFA 3, a Class 1 apresenta uma especificidade de 1.0, enquanto as Classes 0 e 2 têm especificidades de 0.8 e 0.0, respectivamente. Isso sugere que o modelo da TAREFA 3 tem dificuldade particular em identificar verdadeiros negativos para as Classes 0 e 2.

Em resumo, a análise comparativa destaca o desempenho superior do modelo na TAREFA 2 em relação à TAREFA 3, indicando que o primeiro é mais capaz de realizar a classificação com precisão em todas as classes. Essa análise é crucial para entender as nuances do desempenho do modelo em contextos específicos e direcionar futuras iterações e melhorias no processo de treinamento.

Por fim, apesar de entendermos que o modelo aqui utilizado, possui limites significativos para regressão e quantização de dados, com alguns ajustes, há uma significativa

melhora de aprendizados, por isso, podemos dizer que é possível de usar Bert-like para essa tarefa, com ajuda do refinamento o refinamento (finetuning) da rede neural BERTimbau, obtida a partir do site Huggingface. Para concluir, próximas buscas se dão, em aprimoramento destes modelos, e compreensão de que a forma como os dados são separados, estão interligados intrinsecamente com os resultados.

8. References

Hugging Face. (2021). modeling_bert.py. https://github.com/huggingface/transformers/blob/7ae6f070044b0171a71f3269613bf02fd9fca6f2/src/transformers/models/bert/modeling_bert.py#L1564-L1575 Acesso em 02 Dez. de 2023

Hugging Face. (2021). text_classification.py. https://github.com/huggingface/transformers/blob/7ae6f070044b0171a71f3269613bf02fd9fca6f2/src/transformers/pipelines/text_classification.py#L140-L162 Acesso em 02 Dez. de 2023

La Javaness R&D. (2022). Regression with Text Input Using BERT and Transformers. https://lajavaness.medium.com/regression-with-text-input-using-bert-and-transformers-71c155034b13 Acesso em 02 Dez. de 2023

SOUZA, Fábio; NOGUEIRA, Rodrigo; LOTUFO, Roberto. BERTimbau: modelos pré-treinados BERT para o português brasileiro. In: 9º Congresso Brasileiro de Sistemas Inteligentes, BRACIS, Rio Grande do Sul, Brasil, 20 a 23 de outubro de 2020