UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Raphael Ramos da Silva

Análise da aderência semântica de redações do ENEM ao tema: uma abordagem baseada no BERTimbau

Florianópolis
2023

Raphael Ramos da Silva

Análise da aderência semântica de redações do ENEM ao tema: uma abordagem baseada no BERTimbau

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em Sistemas de informação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para obtenção do título de Bacharel em Sistemas de informação.
Orientador: Prof. Renato Fileto, Dr.
Coorientador: Prof. Osmar de Oliveira Braz Junior, M.e.

Florianópolis
2023

Dedico esse trabalho à meus pais, Pedro e Cida e às minhas irmãs Ruth e Nathalia. Em mim há muito de cada um de vocês.

AGRADECIMENTOS

Agradeço primeiramente a Jesus, o autor e consumador da minha fé "Pois todas as coisas vêm dele, existem por meio dele e são para ele. A ele seja toda a glória para sempre. Amém!"(Romanos 16:36). Agradeço também a minha família, sem vocês eu certamente não teria chegado até aqui. Obrigado por todo amor, apoio e ensinamentos recebidos ao longo de minha vida! Agradeço aos meus orientadores, professores Osmar e Renato pelo conhecimento compartilhado. Faço menção especial ao professor Leandro Komosinski *(in memorian)* que com o seu amor pelo ensinar me estimulou o aprender. Por fim, agradeço aos meus companheiros de graduação com quem tanto aprendi durante essa jornada, Luiz Felipe, Rossini Leite, Peter Krause, Gabriel Dall'Agnol, José França, Matheus Gomes, e Samuel Landre, meu muito obrigado.

"E o Verbo se fez carne, e habitou entre nós, cheio de graça e de verdade; e vimos a sua glória, como a glória do unigênito do Pai."João 1:14

RESUMO

Todos os anos, milhares de estudantes brasileiros se submetem à maior avaliação de ensino do país, o Exame Nacional do Ensino Médio (ENEM). O exame avalia não só a qualidade da educação básica nacional, mas também é utilizado para o ingresso em instituições de ensino superior. Além de questões de múltipla escolha abrangendo as grandes áreas do conhecimento, a prova também é composta por uma redação que deve ser redigida obedecendo o estilo dissertativo-argumentativo. A redação é avaliada em 5 competências, sendo a segunda competência a responsável por avaliar se o texto produzido se adequa ao tema proposto. O processo manual de avaliação das redações é dispendioso. O custo estimado em 2015 para cada correção de redação era de R$*15,88*. Nesse mesmo ano, *6,4 milhões* de redações foram corrigidas. Levando isso em consideração, o presente trabalho propõe o uso de técnicas de Processamento de Linguagem Natural (Processamento de Linguagem Natural (PLN)), incluindo modelos de linguagem baseados em aprendizado profundo, para predizer automaticamente a pontuação de cada redação na segunda competência avaliativa. Tal proposta não apenas se alinha ao que há de mais recente nas práticas de PLN voltadas ao âmbito educacional, como também busca preencher a lacuna de aplicações correlatas especificamente adaptadas para a língua portuguesa. O uso do modelo de linguagem BERT é central para nossa investigação, especificamente a variação *BERTimbau*, pré-treinada para a língua portuguesa do Brasil. Para atingir os objetivos, foi primeiramente feita uma análise exploratória dos dados. Posteriormente, experimentos utilizaram o *BERTimbau* para extrair *embeddings* contextualizados dos textos das redações e dos textos motivadores.A partir desses *embeddings* foram calculadas medidas de similaridade das redações com textos motivadores, para primeiro investigar possíveis correlações dessas medidas com as notas na competência 2. Como não foram observadas correlações significativas, posteriormente foram criados um modelo de regressão e outro de classificação, mediante *fine-tuning* do *BERTimbau*, para predizer as notas a partir dos textos das redações. Ambos os modelos foram treinados com três diferentes taxas de aprendizado e testados usando validação cruzada (*k-fold cross validation)*. O modelo produzido para a tarefa de classificação apresenta boa capacidade de generalização a novos dados, atingindo a acurácia 81,73 e F1-*score* de 0,80, para a taxa de aprendizado 5e-5 com o *dataset* de validação. Já os resultados para o modelo de regressão sugerem baixa adaptabilidade para resolver o problema proposto tendo seu MAE superior a 120 para treinamento e validação.

Palavras-chave: Aderência de ensaios/redações a temas, redações do ENEM, similaridade semântica, modelos contextualizados de linguagem, BERT, regressão, classificação.

ABSTRACT

Every year, thousands of Brazilian students undergo the largest teaching assessment in the country, the Exame Nacional do Ensino Médio (ENEM), in a free translation, National Secondary Education Exam. The exam not only assesses the quality of national basic education but is also used for admission to higher education institutions. In addition to multiple-choice questions covering major areas of knowledge, the test also consists of an essay that must be written in accordance with the dissertation-argumentative style. The writing is analyzed according to 5 skills, the second skill being related to the semantic adherence of the text produced to the proposed theme. The manual process of evaluating essays is expensive. The estimated cost in 2015 for each writing correction was R$*15.88*. In that same year, 6.4 million of essays were corrected. Taking this into consideration, the present work proposes the use of Natural Language Processing (NLP) techniques, including deep learning-based language models, to automatically predict the score of each essay in the second assessment competency. This proposal not only aligns with the latest practices in NLP aimed at the educational sphere but also seeks to fill the gap in related applications specifically adapted for the Portuguese language. The use of the BERT language model is central to our investigation, specifically the BERTimbau variation, pre-trained for Brazilian Portuguese. To achieve the objectives, an exploratory data analysis was first carried out. Subsequently, experiments used BERTimbau to extract contextualized embeddings from the essay texts, themes, and texts motivating the themes. Using these embeddings, measures of similarity of essays with themes and motivating texts were calculated, to first investigate possible correlations of these measures with scores in competency 2. As no significant correlations were observed, a regression model and a classification model were subsequently created, using fine-tuning of BERTimbau, to predict grades based on the essay texts. Both models were trained with three different learning rates and tested using k-fold cross-validation. The model produced for the classification task presents good generalization capacity to new data, reaching an accuracy of 90.45 and an F1-score of 0.90, for one of the learning rates. The results of the regression model suggest low adaptability to solve the proposed problem, with its MAE greater than 120 for training and validation.

Keywords: Adherence of essays no themes, ENEM essays, semantic similatiry, contextualizaed language models, BERT.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

BERT	Bidirectional Encoder Representationsfrom Transformers
EDA	Análise Exploratória de Dados
ENEM	*Exame Nacional do Ensino Médio*
LSA	Latent Semantic Analysis
MAE	*Mean Absolute Error*
MSE	*Mean Squared Error*
NLP	Natural Language Processing
PCA	*Principal Component Analysis*
PLN	Processamento de Linguagem Natural
RD	*Redução de Dimensionalidade*
RMSE	*Root Mean Squared Error*
RN	Redes Neurais
SBERT	Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks
SVM	*Support Vector Machine*

SUMÁRIO

1 INTRODUÇÃO

O advento das *redes sociais*, *mídias digitais* e o *big data* propiciou nas últimas décadas um aumento considerável no volume de dados gerados por usuários na Internet. A maior parte desses dados estão em formato texto e seu volume estimado é na ordem de petabytes (BAEZA-YATES; RIBEIRO-NETO, 2013). Estima-se que até 2025, o volume de dados no mundo atingirá a marca de 175 *zetabytes* (REINSEL JOHN GANTZ, 2022), sendo 1 *zetabyte* o equivalente a 1 trilhão de *gigabytes.* (BAEZA-YATES; RIBEIRO-NETO, 2013). O famoso jargão *"Dados, o novo petróleo."* (tradução do autor) foi cunhado em 2017 pelo jornal inglês *The Economist* no artigo *"O recurso natural mais valioso do mundo"* [1] (tradução do autor). Nesse artigo, o jornal britânico aponta para um novo tipo de *commoditie* que substituiria o petróleo em poucos anos, se tornando o centro das atenções na dinâmica da economia mundial. O artigo se referia aos dados oriundos da interação de usuários com dispositivos conectados à internet.

Nesse cenário, ferramentas computacionais capazes de processar a linguagem humana têm ganhado destaque, e a demanda pela solução de problemas reais a partir de técnicas de Processamento de Linguagem Natural (PLN) emergem aos montes nos mais diferentes setores da sociedade. Entende-se por Processamento de Linguagem Natural, grosso modo, o emprego de técnicas computacionais para processar, compreender, ou ainda, gerar linguagem humana (HAGIWARA, 2021). Atualmente, técnicas de PLN são utilizadas para resolverem diversos tipos de problemas em diferentes contextos, como por exemplo, na automatização de correção de questões discursivas (OLIVEIRA; POZZEBON; SANTOS, 2020), na análise de *feedbacks* em ambientes de aprendizagem virtual (CAVALCANTI; MELLO; MIRANDA PERICLES BARBOSA CUNHA DE AND, 2020), na análise de propriedades linguísticas em redações do ENEM (BERTUCCI, 2021), na análise de coerência na postagens de fóruns de dúvidas (BRAZ; FILETO, 2021) ou ainda na mineração de padrões linguísticos em discursos proferidos por políticos durante a corrida eleitoral americana de 2016 (SORATO; GOULARTE; FILETO, 2020).

A popularização de plataformas de ensino à distância (INEP, 2020) também tem demandado ferramentas capazes de dar suporte ao processamento e análise de textos (CAVALCANTI et al., 2020) produzidos por estudantes, devido ao seu grande potencial de amenizar o esforço humano empregado na realização das tarefas de avaliação de textos. Um dos maiores expoentes dessa demanda é o Exame Nacional do Ensino Médio (ENEM), visto que em 2015 o custo para a correção manual de cada redação era de *15,88 R$*. Naquele mesmo ano, foram registradas *6,4 milhões* de redações corrigidas, com um custo total de pelo menos *101 milhões* aos cofres públicos. Havendo discrepância entre as avaliações dos dois primeiros avaliadores, um terceiro avaliador é acionado e automaticamente os custos por redação aumentam.

[1] https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data

1.1 DESCRIÇÃO DO PROBLEMA

A nota da redação do ENEM tem peso significativo na composição da nota final do candidato, e por isso é uma das etapas mais críticas durante a etapa de correção. Além disso, o exame estabelece uma série de regras que descrevem as situações que podem levar uma redação a nota zero. Entre todas as situações descritas e explicitadas no documento "Situações nota zero"[2] a que é mais pertinente para este trabalho é a seguinte:

- Não atender à proposta solicitada ou possua outra estrutura textual que não seja a estrutura dissertativo-argumentativa, o que configurará “Fuga ao tema ou não atendimento à estrutura dissertativo-argumentativa

Essas redações são avaliadas segundo 5 critérios descritos na cartilha do participante de 2022[3] sendo eles:

1. Demonstrar domínio da modalidade escrita formal da língua portuguesa
2. Compreender a proposta de redação e aplicar conceitos das várias áreas de conhecimento para desenvolver o tema, dentro dos limites estruturais do texto dissertativo-argumentativo em prosa.
3. Selecionar, relacionar, organizar e interpretar informações, fatos, opiniões e argumentos em defesa de um ponto de vista.
4. Demonstrar conhecimento dos mecanismos linguísticos necessários para a construção da argumentação
5. Elaborar proposta de intervenção para o problema abordado, respeitando os direitos humanos.

Nosso trabalho se debruçará especialmente sobre a segunda competência e terá como objetivo satisfazer este critério avaliativo a partir da análise da compatibilidade entre os universos semânticos das redações produzidas e seus respectivos textos motivadores. Está fora de escopo avaliar se o texto obedece ao tipo textual dissertativo-argumentativo. Nos ateremos apenas a avaliar se o texto respeita a proposta de redação.

1.1.1 Redação nota zero

Na edição do ENEM de 2015, pelo menos *529,373* textos foram "zerados", ou ainda, anulados, sendo a fuga total ao tema o maior causador de notas zero.[4] A título de exemplo, observemos o seguinte cenário, na edição do ENEM de 2018, o tema proposto para a redação foi **“Manipulação do comportamento do usuário pelo controle de dados na internet”**. Consideremos as seguintes sentenças retiradas redações oficiais:

[2] https://download.inep.gov.br/educacao_basica/enem/downloads/2020/Situacoes_nota_zero.pdf
[3] https://download.inep.gov.br/download/enem/cartilha_do_participante_enem_2022.pdf
[4] https://memoria.ebc.com.br/educacao/2015/01/enem-2014-fuga-do-tema-foi-o-principal-erro-que-estudantes-cometeram-na-redacao

1. *"Em sua canção "Pela Internet", o cantor brasileiro Gilberto Gil louva a quantidade de informações disponibilizadas pelas plataformas digitais para seus usuários".*

2. *"Me desculpa, mas não vou fazer essa redação, minha cabeça tá prestes a estourar. Eu só queria estar em casa assistindo o jogo do meu mengão"*

Na primeira sentença, as palavras sublinhadas*"Internet"*, *"informações"*, *"plataformas digitais"* e *"usuários"* estão relacionadas semanticamente ao tema na edição da prova em 2018 e indica que o candidato compreendeu e discorreu sobre o tema proposto. A segunda sentença entretanto, retirada de uma redação disponível no documento "Situações nota zero"[5] (Exemplo 26) disponibilizado pelo INEP, desconsidera completamente o tema proposto, além de não compor um texto dissertativo-argumentativo. Nesse caso, toda a redação foi anulada por "Fuga ao Tema".

Para resolver este problema pretendemos avaliar algoritmos que identifiquem e analisem os padrões semânticos nas redações, utilizando mineração de dados e *embeddings* contextualizados para aproximação de sentenças à um universo semântico conhecido, isto é, o tema e textos motivadores.

1.2 OBJETIVOS

O objetivo geral deste trabalho é desenvolver modelos treinados em aprendizado profundo capazes de, dada redação e seu respectivo texto motivador, aferir a potencial nota atingida na competência 2 do ENEM ou ainda a qual classe de notas. Será crucial para os experimentos, bem como para as etapas de treinamento, a representação do conteúdo semântico desses documentos através de *embeddings* contextualizados de palavras, viabilizado pelo modelo de linguagem BERT. Por fim, o presente trabalho almeja contribuir para os avanços no desenvolvimento de ferramentas de PLN adaptadas para a língua portuguesa a partir de tecnologias em estado da arte.

1. Encontrar um conjuntos de dados que possua redações escritas no formato dissertativo-argumentativo e escolher o que melhor se adéque aos objetivos.
2. Aplicar técnicas de PLN e de análise exploratória de dados visando entendê-los, garantir seu processamento eficaz e contribuir para o pre-processamento dos mesmos e a qualidade dos dados produzidos.
3. Desenvolver e avaliar abordagens alternativas para aferir aderência semântica de textos a temas, tais como correlação de medidas de similaridade, classificação e regressão.
4. Realizar experimentos para comparar os resultados das abordagens alternativas consideradas na aferição da aderência de redações.
5. Disseminar os resultados obtidos na pesquisa em notebooks, manuais, monografia e artigos a serem submetidos para publicação.

[5] https://download.inep.gov.br/educacao_basica/enem/downloads/2020/Situacoes_nota_zero.pdf

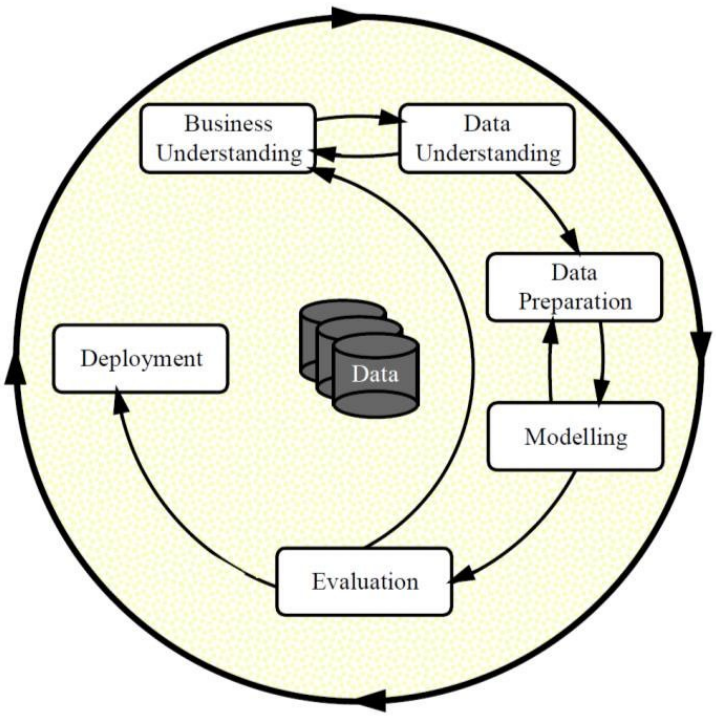
1.3 METODOLOGIA

A princípio, será realizada uma pesquisa bibliográfica sobre o estado da arte acerca dos seguintes temas: PLN, classificação morfossintática, similaridade semântica e aprendizado não-supervisionado. Faremos também a coleta e pré-processamento dos textos utilizados, etapa que compreende o agrupamento de redações por tema, ano da edição, nota, e textos motivadores em sua versão na íntegra. Embora nosso enfoque seja nos textos que são reconhecidos pelo INEP por terem atingido nota máxima, analisaremos também textos não oficias, majoritariamente do conjunto de dados UOL Redações[6].

1.3.1 Processo de referência do CRISP-DM

O presente trabalho será orientado pela pelo processo de referência *Cross-Industry Standard Process for Data Mining* - CRISP-DM (WIRTH; HIPP, 2000) para mineração de dados e aprendizado de máquina. Ele consiste de uma sucessão de etapas iterativas que buscam sistematizar a solução de problemas dirigidos a dados. As etapas compreendem: 1) entendimento do negócio; 2) entendimento dos dados; 3) preparação dos dados; 4) modelagem; 5) avaliação e 6) implantação;

Figura 1 – Fases do CRISP-DM



Fonte: (WANG; YAN; WU, 2018)

[6] https://educacao.uol.com.br/bancoderedacoes/

1.4 ESTRUTURA DO TRABALHO

O presente trabalho segue a seguinte estrutura: no capítulo Capítulo 2 - Fundamentos, será abordado a fundamentação teórica utilizada como embasamento para o desenvolvimento da pesquisa. Além disso, o capítulo também abordará os principais conceitos relacionados a área de Processamento de Linguagem Natural; o capítulo Capítulo 3 - Trabalhos Relacionados, analisa os trabalhos correlatos em que os autores também se propuseram a resolver o problema da avaliação automática de redações a partir de técnicas computacionais; o capítulo Capítulo 4 - Desenvolvimento, detalha os métodos, técnicas e critérios que foram utilizados para estabelecer o plano de fundo em que os experimentos aconteceram; o capítulo Capítulo 5 - Análise Exploratória dos Dados exibirá as principais características do conjunto de dados utilizado, tais como: medidas de estatística descritiva, distribuição de notas finais e na competência 2, distribuição de redações por tema, entre outras; o capítulo Capítulo 6 - Experimentos e resultados, detalha os principais experimentos e apresenta seus resultados, seguido de uma discussão que aborda o desempenho dos modelos e os compara; por fim, o capítulo 7 - Conclusão, enuncia como o presente trabalho atingiu aos objetivos definidos e ainda aborda fatores limitantes e trabalhos futuros.

2 FUNDAMENTOS

A presente sessão apresenta o objeto de estudo desse trabalho, textos dissertativo-argumentativos escritos em conformidade com as regras atuais do ENEM. A seção também discute brevemente a concepção e as principais aplicações de técnicas de PLN, e contextualiza o leitor sobre umas de suas vertentes conhecida como *Similaridade Semântica Textual*, tópico especialmente relevante para este trabalho. Por fim, abordaremos *embeddings* de palavras e suas propriedades, modelos contextualizados e treinados para a língua portuguesa e finalmente técnicas de redução de dimensionalidade.

2.1 A REDAÇÃO DO ENEM

A redação do ENEM é um componente obrigatório da prova de língua portuguesa e literatura e exige do candidato a elaboração de um texto em prosa, do tipo dissertativo-argumentativo, coerente e bem estruturado sobre um determinado tema. O tema da redação geralmente está relacionado a uma questão social, cultural ou política relevante para a sociedade brasileira.

A Tabela 1 descreve os critérios para avaliar essas redações que são divididos em 5 competências. Cada uma das competências pode receber de 0 a 200 pontos, e soma dos pontos em todas as competências comporá a notal final.

Tabela 1 – Competências gerais avaliadas em redações do ENEM (INEP, 2022)

Competência 1	Demonstrar domínio da modalidade escrita formal da língua portuguesa.
Competência 2	Compreender a proposta de redação e aplicar conceitos das várias áreas de conhecimento para desenvolver o tema, dentro dos limites estruturais do texto dissertativo-argumentativo em prosa.
Competência 3	Selecionar, relacionar, organizar e interpretar informações, fatos, opiniões e argumentos em defesa de um ponto de vista.
Competência 4	Demonstrar conhecimento dos mecanismos linguísticos necessários para a construção da argumentação
Competência 5	5 Elaborar proposta de intervenção para o problema abordado, respeitando os direitos humanos.

Fonte: elaborada pelo autor

A redação é uma parte importante do exame porque avalia não apenas o conhecimento do aluno em língua portuguesa, mas também sua capacidade de expressar suas ideias de forma clara e persuasiva. Também é visto como uma forma de promover o pensamento crítico e a consciência social entre os estudantes brasileiros. Além da estrutura obrigatória apresentada, um texto deverá respeitar uma série de outros critérios para não ser atribuído nota 0 (zero), sendo alguns deles: fuga total ao tema, não obediência ao tipo textual dissertativo-argumentativo, textos com até 7 (sete) linhas de extensão, etc. (INEP, 2022)

2.1.1 Competência 2

A segunda competência avaliativa é a responsável por identificar se o candidato compreendeu a proposta da redação, composta por um tema específico, e se desenvolveu sua argumentação a partir disso. Conjuntamente, é verificado se o texto está no formato dissertativo-argumentativo. O que faz dessa competência uma das mais importantes é o fato de que ela é a única capaz de anular uma redação por inteiro, isto é, atribuir nota 0 (zero) a todas as competências avaliativas. Isso ocorrerá caso o conteúdo do texto redigido não tenha compatibilidade com a proposta de redação, caracterizando fuga total ao tema, ou o candidato não respeite o tipo textual exigido. Neste trabalho focamos no primeiro caso.

2.1.1.1 Tema e textos motivadores

O tema constitui o cerne das ideias que comporão o texto, e via de regra consiste de um recorte de um assunto mais amplo. O tema vem acompanhado de textos motivadores, que tem como objetivo explorar o assunto do tema bem como estimular a criatividade do participante. Estes textos também buscam suscitar o uso do **repertório sócio-cultural** do estudante, ou seja, sua experiência de vida de modo a contribuir para o enriquecimento argumentação. Contudo, essa exploração do assunto deve ser cautelosa e não distanciar-se do tema o que pode implicar em penalidades na pontuação da segunda competência. Outro fator importante é que trechos dos textos motivadores não devem ser copiados literalmente no texto produzido sob risco de penalidade e até mesmo sua completa anulação.

2.1.1.2 Níveis de pontuação

Cada competência apresentada na Tabela 1, é composta por 6 níveis de pontuação, que variam de 0 a 200 em intervalos de 40 pontos. Cada pontuação representa um nível de proficiência e conformidade com as características esperadas para a competência. A pontuação máxima, indica que todos os requisitos avaliativos foram atingidos enquanto que 0 (zero) aponta completa inadequação à competência. A Figura 2 apresenta os níveis de desempenho para a competência 2:

Figura 2 – Níveis de desempenho para avaliação da competência 2

200 pontos	Desenvolve o tema por meio de argumentação consistente, a partir de um repertório sociocultural produtivo, e apresenta excelente domínio do texto dissertativo-argumentativo.
160 pontos	Desenvolve o tema por meio de argumentação consistente e apresenta bom domínio do texto dissertativo-argumentativo, com proposição, argumentação e conclusão.
120 pontos	Desenvolve o tema por meio de argumentação previsível e apresenta domínio mediano do texto dissertativo-argumentativo, com proposição, argumentação e conclusão.
80 pontos	Desenvolve o tema recorrendo à cópia de trechos dos textos motivadores ou apresenta domínio insuficiente do texto dissertativo-argumentativo, não atendendo à estrutura com proposição, argumentação e conclusão.
40 pontos	Apresenta o assunto, tangenciando o tema, ou demonstra domínio precário do texto dissertativo-argumentativo, com traços constantes de outros tipos textuais.
0 ponto	Fuga ao tema/não atendimento à estrutura dissertativo-argumentativa. Nestes casos a redação recebe nota zero e é anulada.

Fonte: Cartilha do participante (INEP, 2022)

2.2 CONJUNTOS DE REDAÇÕES

Através de pesquisa bibliográfica foi percebido que as bases de dados mais utilizadas para análise e processamento de textos argumentativos e/ou dissertativos em língua portuguesa eram três, à saber, *Essay-BR* (MARINHO; ANCHIêTA; MOURA, 2021), Banco de Redações UOL [1], e o Projeto Redação nota 1000 [2]. Cada um será detalhado a seguir.

2.2.1 Essay-BR

O Essay-BR é um *corpus* de ensaios dissertativos escritos em língua portuguesa cujos textos seguem o padrão mais atual do ENEM. A iniciativa se deu quando um grupo de pesquisa da Universidade Federal do Piauí (UFPI), em contribuição para os avanços nas pesquisas em Avaliação Automática de Redações (do inglês, *Automatic Essays Scoring - AES*), decidiu criar um *corpus* atualizado com textos escritos em português do Brasil. O *corpus* contém 4,570 documentos, divididos em 86 tópico como direitos humanos, política e cultura.

As redações foram coletadas durante o período de Dezembro de 2015 a Abril de 2020, e em sua maioria vieram do já citado site, Banco de redações UOL e também do Vestibular UOL [3]. Os textos coletados foram então pré-processados, onde se removeu *tags* HTML e ano-

[1] https://educacao.uol.com.br/bancoderedacoes/
[2] https://projetoredacaonota1000.com.br/
[3] https://vestibular.brasilescola.uol.com.br/banco-de-redacoes

tações dos avaliadores e as pontuações foram normalizadas, já que o site utilizava uma estratégia diferente da empregada pelo exame oficial.

No momento em que este trabalho é realizado, o *dataset* do foi atualizado[4] e o número de textos passa e temas passam a ser 6.577 e 150, respectivamente. A Tabela 2 traz detalhes descritivos sobre o *corpus* atualizado e a Tabela 3 e Tabela 4 trazem amostras de redações e seus respectivos temas e textos motivadores.

Tabela 2 – Descrição do *copus* Essay-BR

Detalhes	**Essay-BR *corpus***
Tipo textual	Dissertativo-argumentativo
Nível de linguagem dos autores	Ensino Médio
Pontuação	Holística
Número de textos	6.577
Número de tópicos	150
Número de competências	5
Intervalo de pontuação por competência	[0 - 200]
Pontuações	[0,40,80,120,160,200]
Intervalo da pontuação geral	[0 - 1000]

Fonte: elaborada pelo autor

Tabela 3 – Amostras de redações

Tema ID	**Título da Redação**	**Redação**	**c1**	**c2**	**c3**	**c4**	**c5**	**Nota**
47	Não tenho internet em casa?	[Questiona-se muito, como está sendo o desenvo...]	120	120	80	80	40	440
83	Como exercitar?	[A prática de esportes é de fundamental import...]	80	160	120	160	200	720
72	NaN	[Com o surgimento de ferramentas online como o...]	120	200	200	200	200	920

Fonte: elaborado pelo autor

[4] https://github.com/lplnufpi/essay-br

Tabela 4 – Amostras de temas e textos motivadores

Tema ID	Tema	Texto Motivador	Categoria
47	Desafios na Educação a Distância no Brasil	[Educação a distância é a modalidade educaci-ona...]	Educação
83	O que podemos aprender com a prática esportiva?	[O homem moderno vem deixando de lado as práti...]	Saúde
72	Informação e sociedade: o combate às fake news em ano de eleições presi-denciais	[As fake news são mais do que simplesmente not...]	Sociedade e cultura

Fonte: elaborado pelo autor

2.2.2 Banco de Redações UOL

O Banco de redações UOL é um dos maiores sites especializados em correção de redações online no Brasil. Há mais de uma década, o site Educação UOL, estimula internautas ao aprimoramento da escrita através da prática, sobretudo visando o supracitado exame. Para tal, disponibiliza especialistas que corrigem gratuitamente redações submetidas por usuários registrados[5]. Contudo, o site não divulga os autores das redações. A nota da redação varia de 0 à 200 em intervalos de 40 pontos. São 5 níveis avaliados e são apresentados em detalhe na Figura 2.

2.2.3 Projeto Redação nota 1000

O projeto redação nota 1000 é uma plataforma em formato de fórum que permite que usuários cadastrados submetam e também corrijam redações. As métricas de avaliação seguem o mesmo formato do ENEM e todas as correções são disponibilizadas em domínio público. Um dos problemas da plataforma é que os textos são corrigidos pelos próprios estudantes e não por especialistas de domínio ou profissionais experientes.

2.2.4 Comparação das coleções de redações

Das opções listadas, o conjunto de dados que melhor se adequou a proposta do presente trabalho, foi o *corpus* Essay-BR. Um dos fatores que contribuíram para essa escolha é que conjunto de dados disponibiliza redações no formato almejado, seus respectivos textos motivadores, e as notas para cada competência avaliativa. Além disso, possui uma quantidade significativa de registros e seu *dataset* está disponível no formato CSV no Github[6]. A tabela

[5] https://educacao.uol.com.br/bancoderedacoes/redacoes/a-apropriacao-cultural-e-o-conhecimento-historico.htm

[6] https://github.com/rafaelanchieta/essay

Tabela 5 compara as características dos conjuntos de dados considerados.

Tabela 5 – Tabela comparativa das bases de dados

Fonte	**Tipo de texto**	**Tema**	**Notas por competência**	**Texto motivador**
Essay-BR (MARINHO; ANCHIêTA; MOURA, 2021)	ENEM	Sim	Sim	Sim
Banco de redações UOL [7] [1]	ENEM	Sim	Sim	Não
Projeto redação nota 1000 [8] [2]	ENEM	Sim	Sim	Sim

Fonte: elaborado pelo autor

2.3 PROCESSAMENTO DE LINGUAGEM NATURAL

Processamento de Linguagem Natural (PLN), segundo (HAGIWARA, 2021) é uma abordagem computacional baseada em princípios científicos utilizada para o processamento da linguagem humana. Segundo o mesmo autor, o campo é um sub-campo da Inteligência Artificial pelo fato de que boa parte da inteligência humana está codificada em palavras. Os estudos na área de PLN datam desde a década de *1930*, ondes foram registrados os primeiros experimentos envolvendo uma máquina de tradução, isto é, um mecanismo capaz de traduzir uma sequência de texto em uma dada língua para uma outra. (HUTCHINS, 2005)

Desde seus primeiros passos até os dias atuais, a área de PLN têm sido amplamente utilizada em diversas aplicações, como por exemplo na correção automática de erro sintáticos e gramaticais, vide famosos *softwares* como *Grammarly, Wordtune, Sapling* etc., na análise de sentimento em textos opinativos, na otimização de métodos avançados de busca em grandes bases de dados não estruturados (BAEZA-YATES; RIBEIRO-NETO, 2013), e etc. Em suma, a PLN já é uma parte integral de nossas vidas, a despeito de sua baixa popularidade para o senso comum.

2.4 SIMILARIDADE SEMÂNTICA TEXTUAL

Uma das sub-áreas do PLN e que sobretudo interessa ao presente trabalho é conhecida como *Similaridade Semântica Textual*. O principal desafio dessa área é corretamente aferir a similaridade (HARTMANN et al., 2017) entre dois termos ou textos curtos, sobretudo levando em consideração seus contextos semânticos. Um dos principais desafios está em como os métodos tradicionais lidam com acrônimos e palavras da moda, ou seja palavras altamente atreladas a um contexto cultural específico (PRAKOSO; ABDI; AMRIT, 2021); siglas, pronomes, e palavras ambíguas também figuram na lista dos desafios da área.

A título de exemplificação considere o seguinte par de sentenças :

1. *"A organização*[1] *criminosa*[2] *é formada por diversos* ***empresários*** *e por um* ***deputado*** *estadual"*

2. *"Segundo a investigação[2] diversos* ***empresários*** *e um* ***deputado*** *estadual integram o grupo[1]."*

Fica evidente que as frases acima compartilham um mesmo universo semântico. Isso se dá especialmente por duas razões, as palavras destacadas dentro de retângulos são comuns as duas sentenças e a similaridade de cosseno entre as palavras *organização*[1] e *grupo*[1], e ainda entre, *criminosa*[2] e *investigação*[2] são consideradas significativas como mostra a Tabela 6. Desse modo podemos, podemos garantir a semelhança entre esse dois textos curtos com o auxílio da similaridade por cosseno.

Tabela 6 – Similaridade de cosseno entre palavras nas sentenças exemplificadas

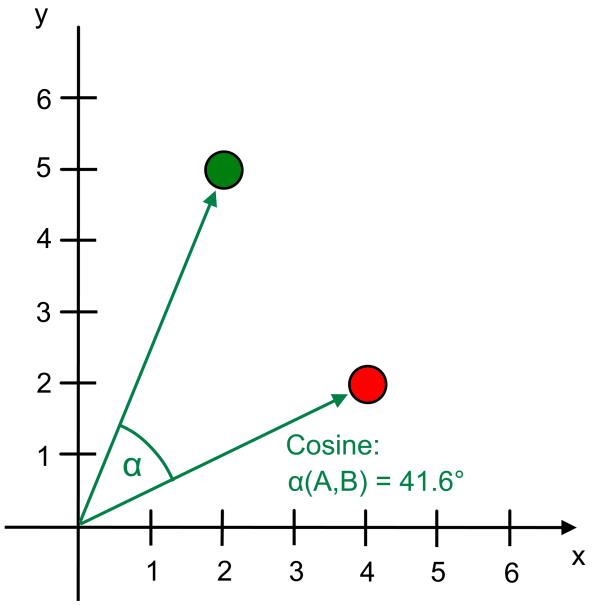
Palavra - Sentença 1	**Palavra - Sentença 1**	**Similaridade**
criminosa	investigação	0,87
organização	grupo	0,69
empresário	empresário	1,00
deputado	deputado	1,00

Fonte: elaborado pelo autor

2.4.1 Similaridade por cosseno

Uma das formas mais utilizadas para determinar a semelhança entre dois documentos textuais é conhecida como similaridade por cosseno. A eficácia dessa métrica é devida ao fato de que ela leva em consideração o sentido e contexto dos conteúdos dos documentos em análise. Em termos simples, cada palavra possui uma representação vetorial, e um documento é representado em um espaço vetorial n-dimensional. Dessa forma, a similaridade por cosseno considera o ângulo entre dois vetores, isto é, ela revela o quão similar ou dissimilar esses vetores são entre si. Por exemplo, se dois vetores são idênticos, o ângulo entre eles é *zero*, a similaridade por cosseno será 1, o maior valor possível. Se os vetores estiverem em lados completamente opostos, o cosseno será -1, o menor valor possível.

Figura 3 – Similaridade de cosseno



Fonte: (EVERT, 2016)

2.5 *WORD EMBEDDINGS*

Embeddings de palavras são representações vetoriais que discretizam variáveis nominais em contínuas viabilizando complexas computações por modelos de Redes Neurais (RN) que essencialmente lidam com números (HAGIWARA, 2021). Cada vetor representando uma palavra ou uma sentença é formado por uma sequência de valores reais e possuem um tamanho fixo. Dessa forma se torna possível a representação *n*-dimensional de palavras, sentenças ou textos, bem como a realização de operações matemáticas com estes vetores tais quais, subtração, média ou adição (HAGIWARA, 2021). Em PLN existem duas categorias de *embeddings* de palavras, *embeddings* estáticos e contextualizados, que serão melhor detalhados a seguir.

2.5.1 *Embeddings* estáticos

Embeddings estáticos podem ser abstraídos como uma coleção estática de parâmetros treinados a partir de grandes datasets de texto-plano, como o *Wikipedia PT-BR*, por exemplo e são nomeados *corpus*. Isso implica dizer que uma dada palavra sempre resultará o mesmo e único vetor de números reais . Considere as seguintes representações a título de exemplo:

$$vec(\text{"}cachorro\text{"}) = [0,8,0,3,0,1]$$
$$vec(\text{"}gato\text{"}) = [0,7,0,5,0,1]$$
$$vec(\text{"}manga\text{"}) = [0,5,0,2,0,8]$$

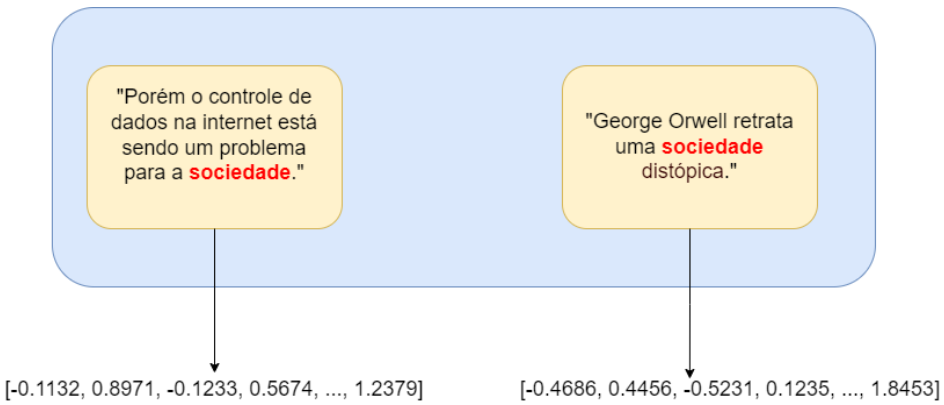
Não importa a situação as mesmas palavras resultarão nos mesmos vetores exemplificados acima. Um dos principais problemas desse tipo de mapeamento direto é que o sentido

das palavras é completamente ignorado, não sendo possível identificar apropriadamente palavras polissêmicas, ou seja, palavras com mais de um sentido dependendo do contexto *e.g.: "Sujei a **manga** da blusa."* e *"A **manga** estava muito doce."*. Em ambos os casos não é possível distinguir a palavra **manga**. Alguns dos algoritmos mais importantes para o treinamento de vetores estáticos são Word2vec (MIKOLOV et al., 2013), FastText (BOJANOWSKI et al., 2017), Wang2vec (LING et al., 2015) e Glove (PENNINGTON; SOCHER; MANNING, 2014).

2.5.2 *Embeddings* contextualizados

Embeddings contextualizados possuem as mesmas propriedades que os *embeddings* estáticos, porém resolvem o problema do contexto semântico em que a palavra está inserida. Neste exemplo abordaremos como o BERT resolve o problema da polissemia, isto é, quando uma palavra possui mais de um significado, levando em consideração as palavras em volta um dada palavra-central para a geração de sua representação vetorial contextualizada (HAGIWARA, 2021). A Figura 4 apresenta a palavra em destaque *sociedade*, presente em ambas as sentenças retiradas de nosso *corpus* de redações, que possuem funções distintas em cada frase. Na primeira sentença, ela assume o papel de substantivo enquanto na segunda seu papel é mais próximo de um adjetivo. Tal distinção não passa despercebidos a modelos contextualizados de linguagem como o BERT.

Figura 4 – A mesma palavra, diferentes sentenças e seus *embeddings* contextualizados



Fonte: elaborado pelo autor

2.6 BERT

BERT, do inglês, *Bidirectional Encoder Representations from Transformers*, é um modelo pré-treinado de representação linguística lançado pela *Google AI* em 2018. O modelo traz avanços consideráveis para tarefas de PLN, sendo uma das mais relevantes a utilização de *embeddings* contextualizados em que o sentido de uma palavra é influenciado pelas palavras ao seu redor. Ao contrário de modelos anteriores, como o Word2Vec[9] e o Glove (PENNINGTON;

[9] https://code.google.com/archive/p/word2vec/

SOCHER; MANNING, 2014) que geravam apenas *embeddings* estáticos e fixos, a preservação de contexto entre palavras possibilita a captação de nuances e desambiguação de sentido.

A arquitetura que viabiliza a eficiência do modelo BERT é chamada de *Tranformers* (VASWANI et al., 2017), que utiliza mecanismos de auto-atenção ou bidirecionalidade para aplicar pesos às palavras de uma sentença. Esse processo ocorre durante a etapa de *encoding*.

O BERT é treinado em cima de grandes bases de dados textuais e é especialmente eficaz na predição de palavras faltantes i.e. *Masked Language Model (MLM)* e predição de próxima sentença, do inglês *Next Sentence Prediction (NSP)* (DEVLIN et al., 2018) através da apreensão do contexto de palavras. Após a etapa de pré-treinamento, o algoritmo pode ser ajustado, isto é, *fine-tuned*, para uma série de tarefas em PLN como: Análise de sentimento, máquinas de tradução, Reconhecimento de Entidade Nomeadas (NER), Q&A, classificação de textos, entre outras.

2.6.1 Símbolos especiais

Os *tokens [CLS]* e *[SEP]* são símbolos especiais utilizados para que o modelo identifique o início e o fim de cada entrada (DEVLIN et al., 2018). O também símbolo especial *[UNK]* representa palavras desconhecidas ao vocabulário. Por fim, *tokens* iniciados por *##* representam palavras que foram subdivididas em sub-palavras ou ainda em um único carácter. Essa é uma das formas que o modelo utiliza para lidar com palavras desconhecidas, do inglês *Out-of-Vocabulary - OOV words*. Desse modo, se uma palavra não é encontrada no vocabulário, a chance de alguma de suas sub-palavras ser encontrada aumenta.

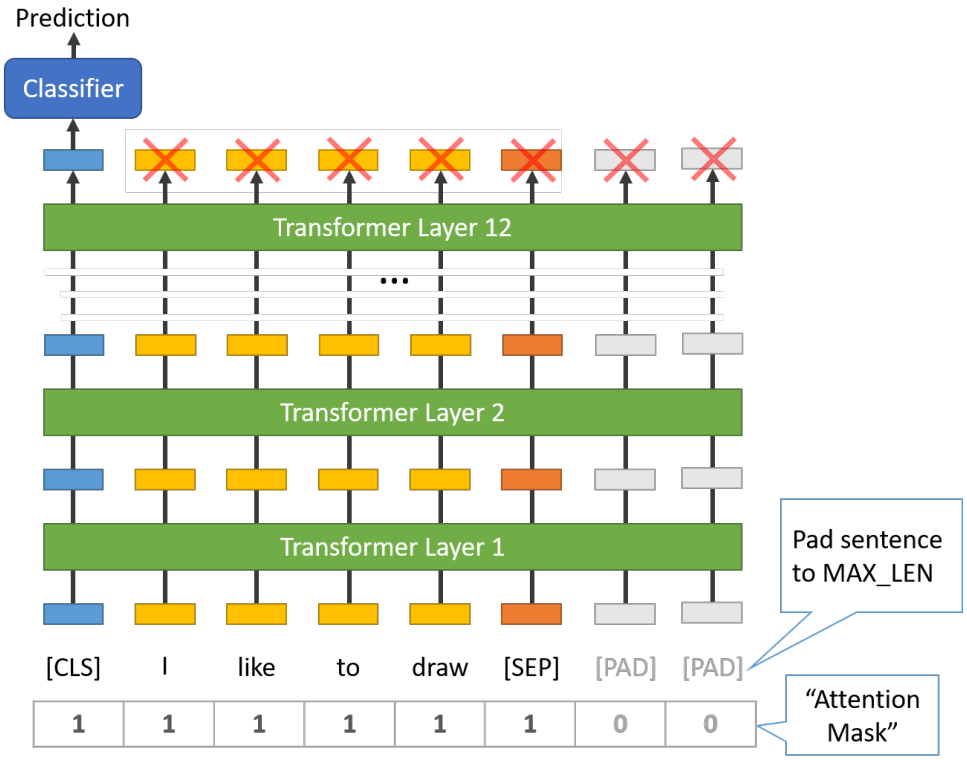
Durante a etapa de treinamento o *token* especial [CLS] é também utilizado pelo classificador para armazenar o agregamento da representação da sequência de texto processada ao longo das 12 camadas do modelo. Os *embeddings* utilizados no presente trabalho são extraídos da última camada do BERT, como recomendam os autores (DEVLIN et al., 2018).

Outro destaque que precisa ser dado é para a forma como o BERT lida com textos de tamanhos diferentes, o que inevitavelmente é o caso no em nossos experimentos. Textos com mais *tokens* que o limite máximo, 512, são truncados, e textos com menos *tokens* que o limite utilizam a técnica de *padding*. Essa técnica consiste em adicionar o *token* especial [PAD] à representação do texto. Dessa forma, *tokens* desse tipo são ignorados pelo mecanismo de auto-atenção do BERT. A Figura 5 detalha como o modelo codifica toda informação que será usada pelo classificador em um único vetor de *embeddings* no *token* especial de classificação da última camada.

2.7 BERTIMBAU

BERTimbau (SOUZA; NOGUEIRA; LOTUFO, 2020) é um modelo treinado em cima do BERT que utilizou técnicas de transferência de conhecimento, do inglês *Transfer Learning*, para ser adaptado para a língua portuguesa do Brasil. O modelo alcançou performances

Figura 5 – Saída do *token* CLS na última camadas do modelo BERT



Fonte: (MCCORMICK, 2023)

em estado-da-arte em três tarefas de PLN: Reconhecimento de Entidades Nomeadas (NER), Similaridade Textual entre Sentenças, e Inferência de Linguagem Natural (ILN) (SOUZA; NOGUEIRA; LOTUFO, 2020). O modelo é disponibilizado em dois tamanhos: *Base* (L = 12 camadas, H = 768, 12 *attention heads*, e 110M parâmetros) e o Large co(L = 12 camadas, H = 1024, 24 *attention heads*, e 330M parâmetros)(SOUZA; NOGUEIRA; LOTUFO, 2020). Nosso objetivo é experimentar diferentes formas de obter *embeddings* contextualizados dos textos processados. Durante o processo de *tokenização*, melhor descrito na seção 6.4.1, o *token* **CLS** é colocado no início de cada sequência de texto. Este é um *token* especial que o modelo foi condicionado a utilizar para codificar a representação das sequências de entrada para tarefas de classificação (DEVLIN et al., 2018). A saída da última camada do modelo consiste na agregação de todos essas representações e por isso é a forma mais comum de se obter *embeddings* contextualizados.

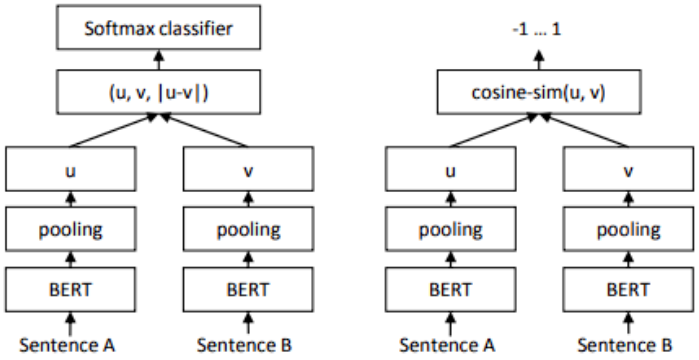
2.8 SBERT

Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks (SBERT) (REIMERS; GUREVYCH, 2019) é uma implementação que modifica o já mencionado BERT (DEVLIN et al., 2018) visando o aprimoramento de tarefas como Pesquisa por Similaridade Semântica, do inglês, *Semantic Similarity Search* e de clusterização não supervisionada, do inglês *clustering*. Tal implementação torna possível a identificação de sentenças semanticamente simi-

lares, e consequentemente sua representação em espaço vetorial através de vetores esparsos de *embeddings* contextualizados. Umas das principais contribuições do *SBERT* é habilitar o BERT a realizar tarefas antes inviáveis devido aos valores proibitivos de processamento requerido.

Buscando mitigar os gargalos de performance identificados no uso do BERT para resolver o problema da similaridade entre sentenças em grandes bases de dados, o SBERT foi implementado através da arquitetura de redes siamesas, o que possibilita a derivação de vetores de tamanho fixo para sentenças usadas como entrada. Tal representação torna possível e computacionalmente eficiente, realizar operações aritméticas entre vetores, tais como agregação, soma e média, bem como a geração de métricas de similaridade, como a de cosseno, distância euclidiana, e *manhattan*. A quantidade máxima de *tokens* processados por entrada é de 128.

Figura 6 – Arquitetura SBERT- à esquerda *fine-tunning* e à direita de inferência SBERT



Fonte: (REIMERS; GUREVYCH, 2019)

2.9 MÉTRICAS DE AVALIAÇÃO E DESEMPENHO

Parte importante do processo do desenvolver um sistema de classificação, ou ainda, de regressão, é validar os resultados obtidos. Sem métricas de avaliação robustas e estabelecidas, não há como avaliar o quão bom o modelo é (BAEZA-YATES; RIBEIRO-NETO, 2013).

2.9.1 Acurácia

A acurácia, dada pela Equação 2.1, representa a proporção dos resultados que o classificador foi capaz de acertar. Ela é calculada pela divisão do número de previsões corretas pelo número total de previsões. A equação é formada pelas variáveis: Verdadeiros Positivos (**VP**) representam corretamente as instâncias positivas, Verdadeiros Negativos (**VN**) são as instâncias negativas corretas, Falsos Positivos (**FP**) são instâncias negativas incorretamente classificadas como positivas, e Falsos Negativos (**FP**) são instâncias positivas incorretamente classificadas como negativas. Entretanto, essa métrica não pode ser avaliada isoladamente, pois nem sem-

pre um modelo com alta acurácia é um modelo confiável. Por exemplo, um modelo capaz de prever 95% de transações bancárias fraudulentas é um modelo ineficiente, já que uma parcela considerável de clientes serão prejudicados.

$$\text{Acurácia} = \frac{VP+VN}{VP+VN+FP+FN} \tag{2.1}$$

2.9.2 Precisão e Revocação

Precisão e Revocação são métricas utilizadas para medir a qualidade do classificador de textos, e quando combinadas na Equação F1*score*, pode minimizar os problemas mencionados sobre a acurácia (BAEZA-YATES; RIBEIRO-NETO, 2013).

A precisão representa o número de resultados verdadeiro positivos dividido pelo número total de positivos preditos, incluindo os falsos positivos. Revocação, ou do inglês, *Recall* representa o número de resultados positivos divido por todas as amostras. A Equação 2.2 define a precisão e a revocação é dada pela Equação 2.3.

$$\text{Precisão} = \frac{VP}{VP+VP} \tag{2.2}$$

$$\text{Revocação} = \frac{VP}{VP+FN} \tag{2.3}$$

2.9.3 F1 score

O *F1 score*, dado pela Equação 2.4, é uma métrica mais confiável pois estabelece um equilíbrio na aferição da capacidade do modelo classificar classes corretamente. Isso acontece porque o *F1 score* é definido pela média harmônica entre a Precisão e a Revocação. O primeiro, demonstra a habilidade do modelo em classificar valores positivos corretamente, e o segundo revela a capacidade do modelo em encontrar os valores positivos. O valor do F1 *score* pode variar de 0 a 1, e quanto mais próximo de 1 melhor o modelo (LEG/UFPR. . . , 2023).

$$F1score = 2 \times \frac{\text{Precisão} \cdot \text{Revocação}}{\text{Precisão} + \text{Revocação}} \tag{2.4}$$

2.9.4 *Loss*

A *Loss* é uma penalidade que o modelo recebe para uma predição errada. Quanto mais próximo de zero for esse valor, indicará a eficiência do modelo em não errar. Se as predições do modelo forem perfeitas, sua *loss* será exatamente zero. Existem diferentes estratégias de penalização do modelo, e cada contexto demandará uma função de perda diferente. (HAGIWARA, 2021) O presente experimento utilizou a função *Cross-Entropy Loss* para a tarefa de classifica-

ção e a função *MSELoss* para a tarefa de regressão. Ambas as funções pertencem a biblioteca *PyTorch*[10].

2.9.5 Validação cruzada

A validação cruzada é um método comumente utilizado para garantir a validação estatística dos resultados de classificação. Essa técnica consiste em criar k classificadores diferentes, e para cada classificador, dividir o conjunto de dados em k partições ou *folds* disjuntos, de modo que o *i-ésimo fold* é utilizado para teste, e restante dos dados para treinamento (BAEZA-YATES; RIBEIRO-NETO, 2013).

A validação *K-fold* é especialmente útil, pois sua dinâmica de validação impede que o modelo seja enviesado por uma determinada partição dos dados. Outro fator importante, é que esse método é amplamente recomendado para treinar conjunto de dados pequenos e desbalanceados. A validação cruzada finalmente ocorre quando é realizada a computação da média das k medidas. (BAEZA-YATES; RIBEIRO-NETO, 2013) Naturalmente, esse tipo de validação é mais custoso em tempo e processamento computacional.

2.9.6 MAE e MSE

Amplamente utilizado em tarefas de regressão, o MAE, do inglês *Mean Absolute Error*, é dado pela Equação 2.5. A medida representa a média das diferenças entre os valores preditos pelo modelo e os valores reais. (JUNIOR; SPALENZA; OLIVEIRA, 2017).

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - y_i| \tag{2.5}$$

O Erro Quadrático Médio - EQM, do inglês *Mean Squared Error*, é comumente utilizado para verificar a acurácia de um modelo de regressão. Isso acontece porque a métrica penaliza os maiores erros do modelo elevando-os ao quadrado. Ao fim calculá-se a média dos erros quadráticos. O MSE é dado pela equação 2.6.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2 \tag{2.6}$$

2.10 VISUALIZAÇÃO DE *EMBEDDINGS* E REDUÇÃO DE DIMENSIONALIDADE

O principal objetivo das técnicas de redução de dimensionalidade é viabilizar a representação mais compacta, sobretudo de dados com um número significativo de atributos em que a visualização de todo o *dataset* se torna inviável. Existem diferentes estratégias para a redução de dimensões em um conjunto de dados, entretanto há de se considerar que inevitavelmente qualquer uma delas implicará em perda de informação (VERLEYSEN; LEE, 2013). Técnicas

[10] https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

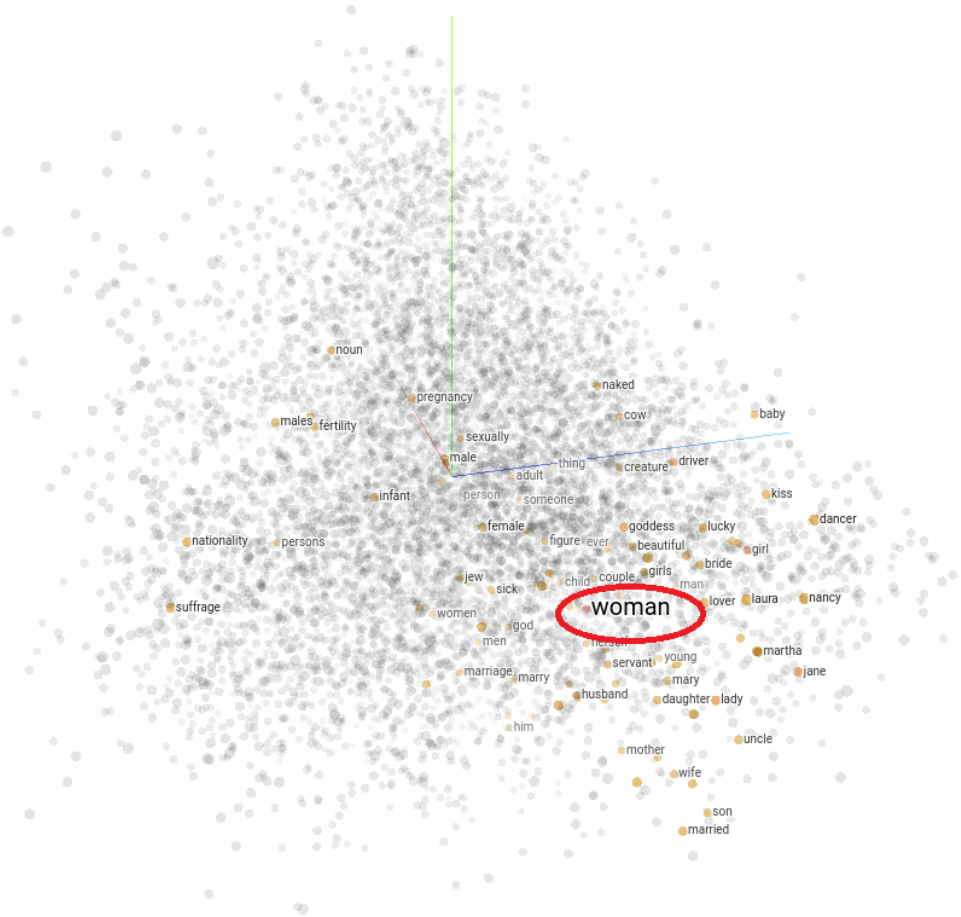
de redução de dimensionalidade - *RD* são especialmente importantes no contexto de PLN devido a natureza desse tipo de dado, grandes quantidades de textos ricos em palavras, e ainda as representações vetoriais de *embeddings* para cada palavra. A visualização de dados se torna importante sobretudo na etapa de análise e exploração em que a visualização destes torna possível a extração de conhecimento *à priori*, identificação de *outliers* e ainda a clusterização por atributos. (VERLEYSEN; LEE, 2013)

Uma das técnicas mais utilizadas e especialmente importante para o presente trabalho é o *PCA - Principal Component Analysis*, em que novos atributos são gerados a partir de combinações lineares dos atributos originais(DEMERS; COTTRELL, 1992), gerando assim os chamados componentes principais que ajudam a explicar a máxima variância dos dados com perda mínima de informação. A representação *2D ou 3D* de *embeddings* de palavras, que são essencialmente vetores *n*-Dimensionais tipicamente com centenas de pontos(SMILKOV et al., 2016), é uma aplicação prática do uso do *PCA* para resolver o problema de visualização em tarefas de PLN.

A Figura 7 exibe a visualização através de *Redução de Dimensionalidade (RD)* utilizando o algoritmo *PCA* e o software *Embedding Projector*[11] (SMILKOV et al., 2016) das palavras mais próximas à palavra *woman*, do inglês, mulher em um *dataset* com 10,000 pontos:

[11] https://projector.tensorflow.org/

Figura 7 – Representação 3D dos *embeddings* de 10.000 pontos/palavras reduzidos a 10 dimensões usando PCA.



Fonte: elaborado pelo autor

3 TRABALHOS RELACIONADOS

Esta seção discute alguns trabalhos relacionados que se propuseram resolver o problema de aferir a consistência semântica de textos longos redigidos por estudantes dado um universo semântico conhecido. Os trabalhos citados utilizaram técnicas similares às apresentadas nos objetivos do presente trabalho e serão detalhados a seguir.

Junior, Spalenza e Oliveira (2017) propõe um sistema de avaliação automática de redações do ENEM. Tal ferramenta faz uso de técnicas de PLN e busca solucionar o problema do alto custo envolvido no processo de correção manual dessas redações. As redações analisadas foram submetidas a pré-processamento e limpeza dos dados. Nessa etapa, caracteres indesejados foram removidos e *tags* morfológicas atribuídas às palavras restantes. Essa última tarefa foi realizada com o framework Apache OpenNLP. Após o pré-processamento, foi realizada a extração de características dos textos a serem analisados, e posteriormente a classificação onde o algoritmo *Support Vector Machine (SVM)* foi utilizado. Os resultados do modelo de classificação foi aferido por funções de precisão, que busca identificar as classificações corretas (True Positive) e as incorretas (False Negative). Mais adiante uma função de qualidade foi implementada para melhorar os resultados obtidos através de ajustes parametrizáveis. Os resultados obtidos indicam que grande parte das redações apresentaram graves erros de gramática e uso vicioso da linguagem. Tal ferramenta, pode ser útil não apenas na redução do custo envolvido na correção de redações, mas também, como ferramenta de apoio a professores e profissionais que avaliam alunos desde a educação básica.

Filho et al. (2018) propõe uma abordagem baseada em aprendizado de máquina para correção automática de redação, levando em conta aderência ao tema e estrutura argumentativa dos ensaios. O trabalho foi feito de acordo com os critérios avaliativos do ENEM capturando aspectos gerais do texto para treinamento de modelos de classificação e regressão baseados em máquinas de vetores de suporte (SVM). O nível de precisão obtido mostra a efetividade da proposta, deixando para trabalhos futuros diversas melhorias, incluindo a normalização de balanceamento dos dados de treinamento. Um modelo de coesão textual adaptado ao português também se mostrou promissor para avaliar a aderência ao tema e a estrutura argumentativa dos ensaios.

Bertucci (2021) analisa propriedades linguísticas de redações nota 1000 no ENEM. Seu enfoque é dado na recorrência de elementos linguísticos que caracterizam o gênero dissertativo-argumentativo, como também a consistência entre a nota obtida pelo estudante e a satisfação do texto nas 5 competências consideradas na avaliação do exame. Em quatro das cinco competências, o enfoque dado é nas características lexicais das redações, como, uso de conectivos, modalizadores, verbos, e etc. Na intenção de analisar o "Universo de referência", ou seja, o repertório dos estudantes refletido nas redações, o autor realiza algo próximo de uma análise semântica em que os principais tópicos são identificados e é verificado a correspondência de palavras encontradas nos textos relacionadas com o tema daquele ano. Como no seguinte exemplo em que o tópico *"comunicação_e_midia"* é correlacionado com as palavras "linguagem, internet

e comunicação"uma vez que o tema exigido na edição do exame daquele ano foi *"Manipulação do comportamento do usuário pelo controle de dados na internet"*. As análises foram feitas em 95 redações prototípicas, realizadas entre 2014 e 2019. Para tal foi utilizado o software *TROPES* [1], uma ferramenta especializada em classificação semântica e análise textual. Em geral nos textos analisados, foram identificados o uso abundante de conectivos e modalizadores, amplitude no universo de referência, e predomínio da estrutura impessoal.

Finalmente Oliveira et al. (2022) utilizam modelos de regressão linear para estimar a coesão textual de redações realizadas em edições anteriores do *ENEM*. O principal enfoque da pesquisa foi analisar o quarto critério [2] considerado na correção das redações, a saber, *(iv) - "Demonstrar conhecimento dos mecanismos linguísticos necessários para a construção da argumentação"*. Para tal, foram identificadas 151 características segundo a literatura especializada sobre o que caracteriza um texto harmoniosamente conexo que foram posteriormente estruturadas em seis grupos: **Uso de conectivos**; **Diversidade Léxica**; **Legibilidade**; **Sobreposição de Frases**; **Coh-Metrix** (GRAESSER; MCNAMARA; KULIKOWICH, 2011) e **Outros** que abarcou aspectos gerais como o número total de frases e palavras, *stop words* e média de palavras por frase. Os autores utilizaram o *corpus Essay-BR*(MARINHO; ANCHIÊTA; MOURA, 2021) que possui 4,570 textos dissertativos em português escritos por estudantes e avaliado por especialistas. Para a seleção do modelo de regressão foi realizado um *benchmark* entre os seguintes algoritmos de regressão: *Extremely Randomized Trees*, *Gradient Boost*, *Perceptron* de múltiplas camadas, Regressão Linear e Regressão de vetores de suporte (SVM). As implementações dos algoritmos utilizados está disponível na biblioteca *scikit-learn*[3]. Para escolha do melhor algoritmo foram levadas em consideração as seguintes métricas: Correlação de Pearson, Erro Quadrático Médio, do inglês *Root Mean Squared Error* (RMSE), e o Erro Absoluto Médio, do inglês, *Mean Absolute Error* (MAE). Sendo assim o algoritmo *Extremely Randomized Trees* foi o que melhor performou apresentando os menores valores para o RMSE 35,94 e MAE: 26,97, bem como a maior correlação de Pearson: 53,08. Os resultados obtidos revelam que as características mais relevantes a partir da correlação de pearson foram: Diversidade léxica, a alta incidência de substantivos e adjetivos, total de palavras com e sem remoção de *stop words*, total de *stop words*, uso de conectivos, sobreposição de palavras entre parágrafos, Índice de legibilidade (Flesch), e por fim a média de sílabas usadas nas palavras.

A Tabela 7 apresenta a relação dos trabalhos citados, seus os objetivos e métodos e características, e ao fim destaca a proposta do presente trabalho.

[1] http://www.semantic-knowledge.com/tropes.htm.
[2] No total são cinco critérios, já apresentados na Seção 2.1
[3] https://scikit-learn.org/stable/

Tabela 7 – Tabela comparativo dos trabalhos relacionados

Trabalho	**Textos**	**Método**	**Ferramentas**	**Saída**
(JUNIOR; SPALENZA; OLIVEIRA, 2017)	Redações do ENEM	Classificação supervisionada (SVM, Gradient Boosting)	OpenNLP, ReGra, CoGrOO	Predição de notas, comparação com avaliações humanas
(FILHO et al., 2018)	Redações do ENEM (Ensaios argumentativos)	Latent Semantic Analysis (LSA), Classificação e regressão (SVM)	TAACO, CoGrOO	Classificação automática de redações
(BERTUCCI, 2021)	Redações do ENEM	Mapeamento de universo de referência	Tropes	Detalhamento gramatical das propriedades linguísticas
(OLIVEIRA et al., 2022)	Redações do ENEM (Essay-BR)	Regressão Linear (Extremely Randomized Trees)	Scikit-Learn	Nota de 0-200 para competência 4 do ENEM
Nossa proposta	Redações ENEM	Treinamento de modelo de regressão e classificação	BERT*imbau*	Predição de notas e categorias de notas na competência 2 do ENEM

Fonte: elaborado pelo autor

4 DESENVOLVIMENTO DO TRABALHO

O presente capítulo reporta as principais etapas realizadas no desenvolvimento do trabalho, descrevendo processos, atividades, métodos, técnicas e critérios relevantes utilizados em cada etapa. Primeiramente, a Seção 4.1 apresenta como a hipótese inicial de pesquisa sobre a possível correlação entre as notas atribuídas às redações e medidas consolidadas das similaridades dos seus textos com os dos seus respectivos temas e textos motivadores. Tal similaridade consolidada é aferida a partir das similaridades dos *embeddigns* dos respectivos documentos ou de suas sentenças. A hipótese foi rigorosamente testada usando diferentes abordagens para calcular similaridades. Os resultados dessas investigações preliminares serviram como indicadores críticos, ajudando a refinar e definir a estratégia adotada para resolver o problema proposto. Posteriormente, a Seção 4.2 descreve os processos e critérios adotados na construção dos modelos de regressão e de classificação para predição de notas das redações.

4.1 ANÁLISE DA HIPÓTESE INICIAL: CORRELAÇÃO NOTAS – SIMILARIDADES

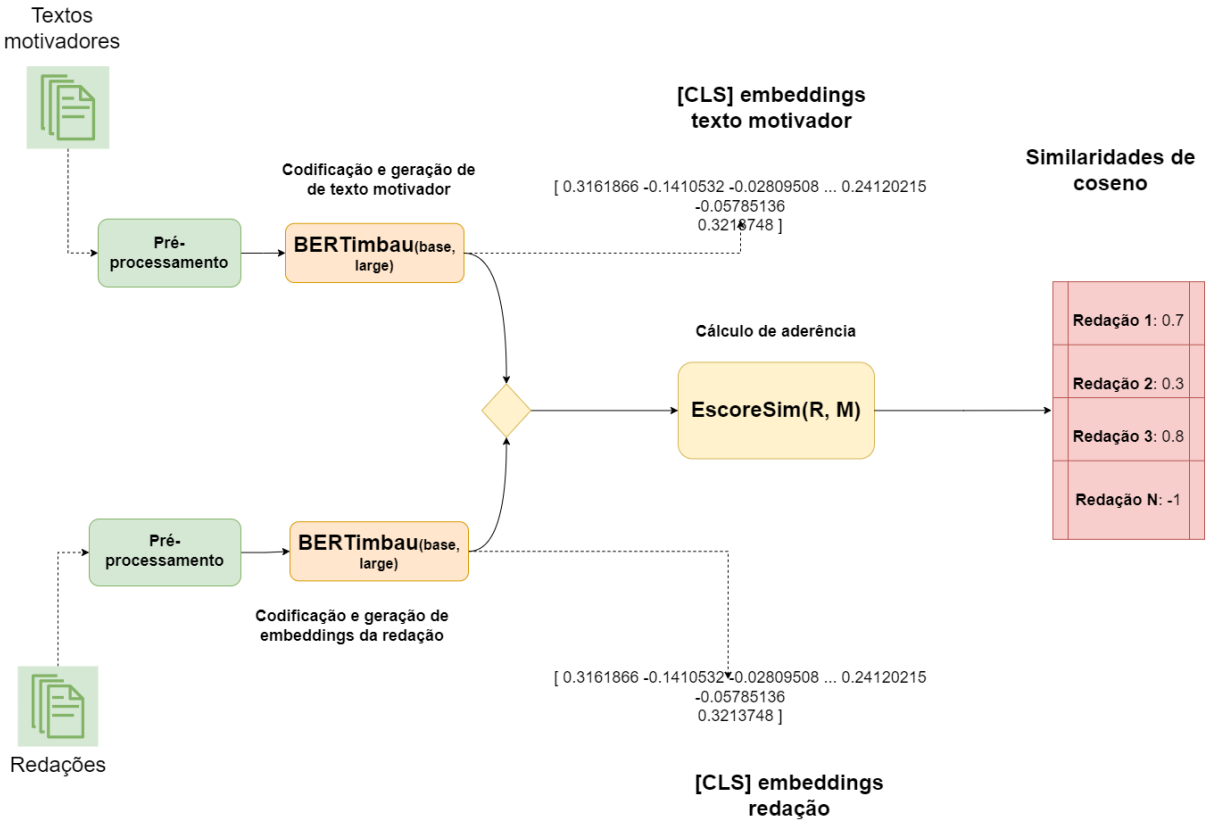
Inicialmente estabelecemos a hipótese que quanto maior a nota na categoria 2 de uma dada redação, maior será a similaridade do cosseno entre *embeddings* de documentos ou sentenças da redação e os dos respectivos temas e textos motivadores. Esta hipótese foi testada calculando similaridades baseadas em *embeddings* de documentos e de sentenças, como detalhado nas subseções subsequentes. A partir desses experimentos, foi possível definir a melhor abordagem para a extração dos *embeddings* a serem utilizados nos experimentos. Os resultados dessa primeira etapa são detalhados no capítulo a seguir. A Seção 6.2 detalha os resultados.

4.1.1 Similaridades com *embeddings* BERT de documentos

Foram inicialmente gerados BERT *embeddings* para documentos inteiros com duas versões do modelo BERT, *base* e *large*. Em seguida foi calculada a similaridade de cosseno entre cada redação e seu respectivo texto motivador. Logo após, uma regressão linear foi traçada entre as similaridades obtidas e a nota da respectiva redação na categoria 2.

A estratégia de utilizar documentos inteiros permite que nuances de sentido e contexto sejam melhor preservadas. Porém, a limitação na quantidade de *tokens* que o BERT é capaz de processar, a saber 512, era um fator que poderia comprometer o experimento já que documentos maiores que o limite máximo são truncados (DEVLIN et al., 2018) implicando em perda de informação. Contudo, durante a etapa de exploração dos dados, descrita no Capítulo 5, observamos que dos 6454 registros em nosso *dataset*, apenas 2653 redações e textos motivadores possuíam menos de 500 *tokens*. Desse modo, os experimentos foram realizados somente com esse subconjunto de dados. A Figura 8 ilustra o fluxo de processamento dos textos e a computação das similaridades de cosseno para documentos com menos de 500 *tokens*.

Figura 8 – Processamento de redações e textos motivadores com menos de 500 *tokens* usando o BERT



Fonte: elaborado pelo autor

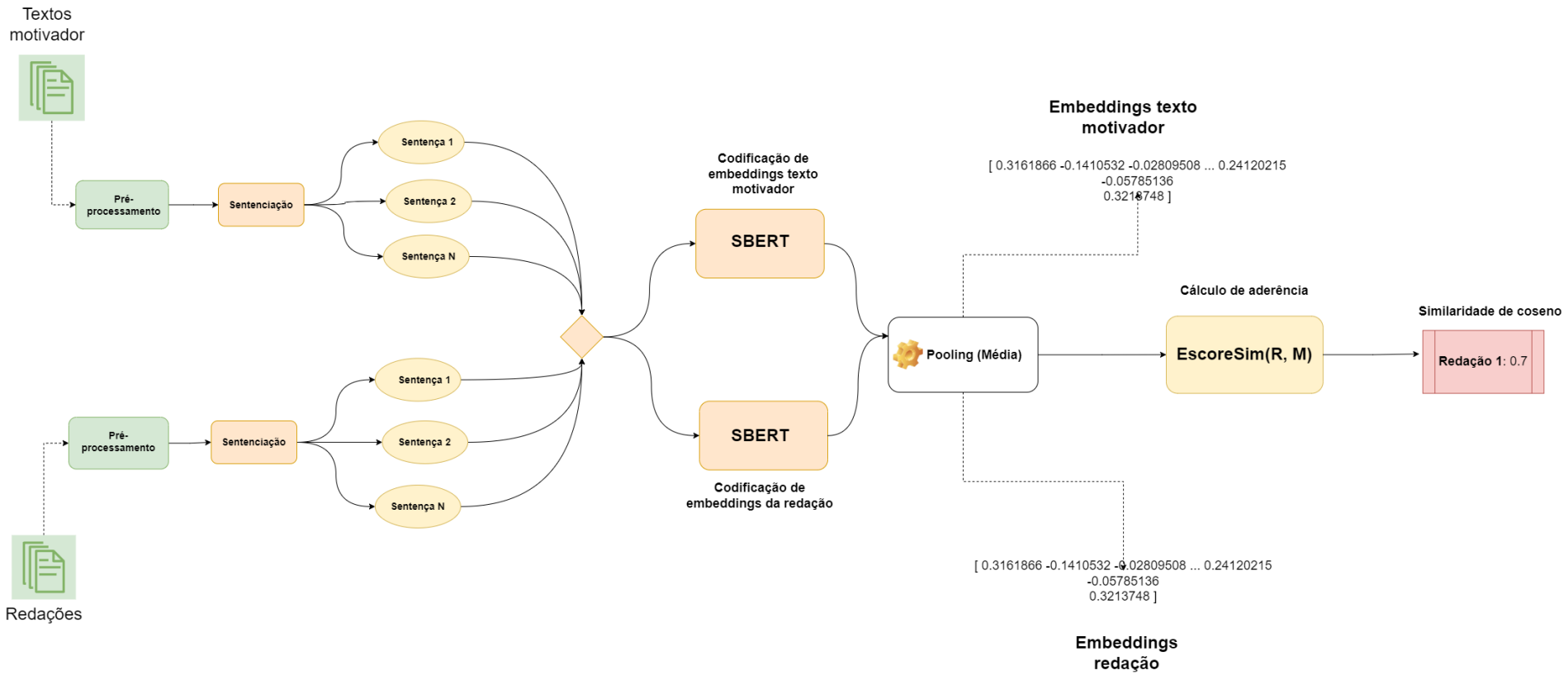
4.1.2 Similaridades com *embeddings* SBERT de sentenças

Uma alternativa ao uso de documentos inteiros foi segmentar sentenças das redações e textos motivadores utilizando o SBERT, uma modificação do BERT, que utiliza redes siamesas para também derivar *embeddings* contextualizados, porém a nível de sentença. Uma vez que os *embeddings* são gerados para cada sentença, tanto para redações quanto para textos motivadores, realiza-se a operação de média dos vetores de sentenças de cada documento produzindo-se assim duas representações vetoriais unidimensionais. Por fim calcula-se a similaridade de cosseno entre os dois vetores. O mesmo subconjunto de dados anterior foi submetido fluxo de processamento detalhado na Figura 9. A geração dos *embeddings* de sentenças com o SBERT e a computação das similaridades de 2653 redações e seus textos motivadores levaram cerca de 20 minutos. Esse mesmo processo se submetido ao BERT levaria cerca de 165 horas para ser completado, segundo nossas estimativas. A Figura 23 apresenta o resultado da correlação entre as similaridades obtidas e as notas da segunda competência.

Cálculo das medidas de aderência usando embeddings de sentenças

As Equações 4.1, 4.2, 4.3 detalham o cálculo do escore de similaridade $EscoreSim(R,t,M)$ usando os conjuntos de *embeddings* R e M das sentenças de cada redação e respectivo texto mo-

Figura 9 – SBERT - pipeline de processamento de redações e textos motivadores via *pooling* de sentenças



Fonte: elaborado pelo autor

tivador, respectivamente, além do *embedding t* da única sentença do título do tema. Os números $\#Sentencas(R)$ e $\#Sentencas(S)$ representam o número de sentenças da redação e do respectivo texto motivador, respectivamente. O parâmetro $0 \leq \alpha \leq 1$ permite ajustar os pesos das similaridade médias se sentenças da redação com a do tema e as o texto motivador, no cálculo do escore geral.

$$SimTema(R,t) = \frac{\sum_{s \in R,t} cos(s,t)}{\#Sentencas(R)} \tag{4.1}$$

$$SimTexMotiv(R,M) = \frac{\sum_{s \in R, s' \in M} cos(s,s')}{\#Sentencas(R) * \#Sentencas(M)} \tag{4.2}$$

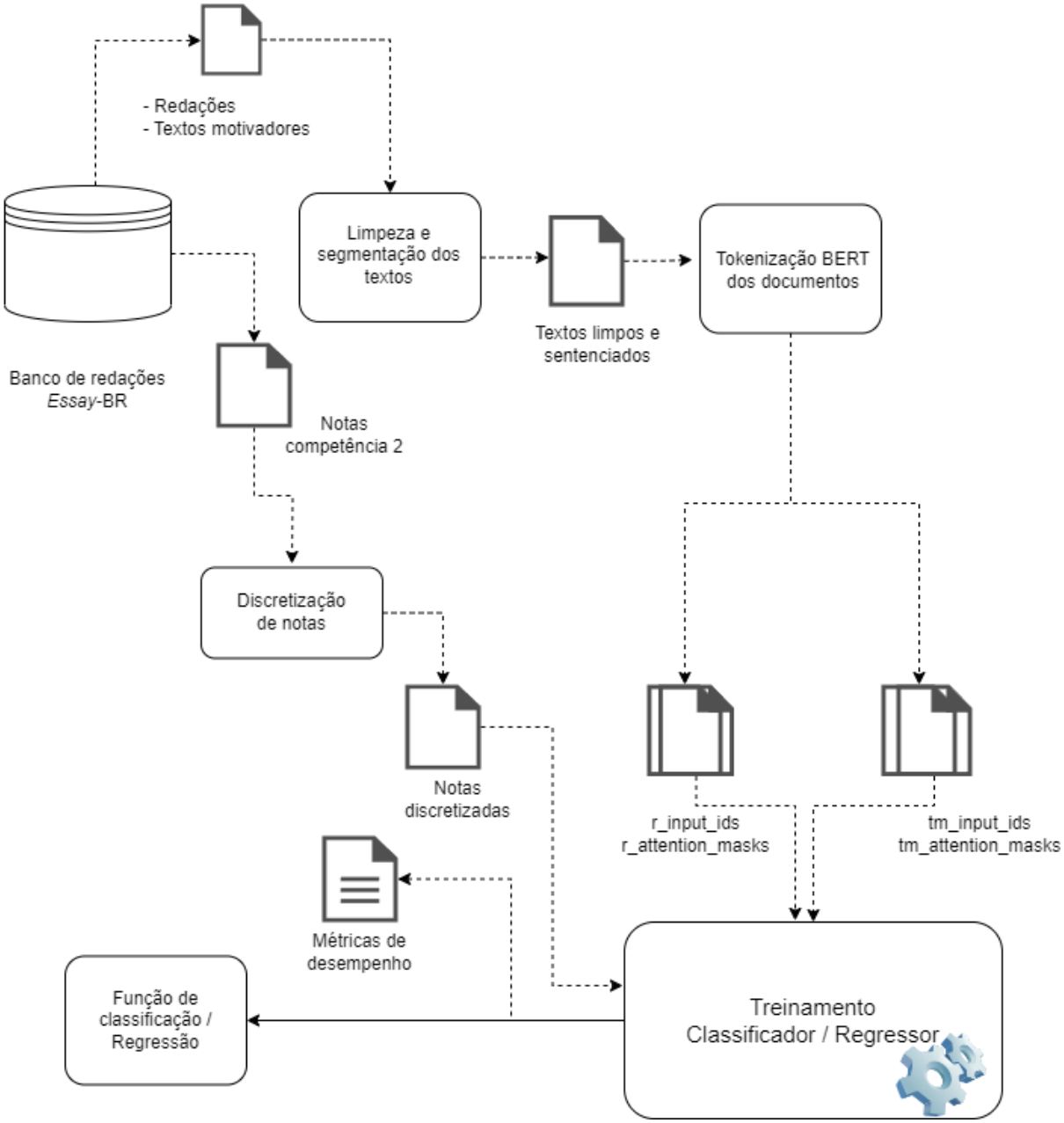
$$EscoreSim(R,t,M) = \alpha * SimTema(R,t) + (1-\alpha) * SimTexMotiv(R,M) \tag{4.3}$$

4.2 CONSTRUÇÃO DE REGRESSORES E CLASSIFICADORES

A Figura 10 ilustra o fluxo para processar redações e textos motivadores no formato do ENEM e predizer a nota na competência 2. Os experimentos são divididos em duas tarefas, uma de classificação e a outra de regressão. A tarefa de classificação tem por objetivo traduzir a correspondência semântica entre redação e texto motivador, em uma das possíveis classes de nota que vai de 0 a 200 em intervalos de 40 *i.e.*(0-39, 40-79, 80-119, 120-159, 160-199, 200). A tarefa de regressão busca estimar a nota com o menor erro médio possível entre o valor predito e o valor real.

As redações provenientes da coleção *Essay-BR* (MARINHO; ANCHIêTA; MOURA, 2021) são primeiramente submetidas ao processo de segmentação e limpeza. Após essa etapa, os textos limpos e segmentados em sentenças são disponibilizados para a etapa de codificação das entradas. Essa etapa é importante porque o BERT não trabalha diretamente com textos, mas com representações numéricas. Esse processo é detalhado na sub-seção 6.4.1. Uma vez que as entradas estão formatadas, Ambos os modelos, de regressão e classificação, usam os textos codificados para treinamento. Especialmente o modelo de classificação, depende que a nota na categoria 2 seja discretizada para uma categoria de notas, que varia de 0 a 200. Essa etapa é apresentada em detalhe na seção 6.6.

Figura 10 – Fluxo de pré-processamento de redações e textos motivadores e treinamento de modelos



Fonte: elaborado pelo autor

5 ANÁLISE EXPLORATÓRIA DE DADOS

Etapa crucial em qualquer tarefa de PLN, a Análise Exploratória de Dados (EDA) permite a visualização e a sumarização das principais características de um *dataset*. Além de validar se a hipótese inicial do problema identificado faz sentido, essa técnica também possibilita uma melhor compreensão dos dados através da identificação de padrões, correlações, anomalias e entre outros. Os resultados aqui descritos refletem apenas o conjunto de dados efetivamente utilizado ao longo de todo o experimento. Isso porque, devido a limitação do BERT quanto ao número máximo de *tokens* processados por entrada, realizamos um filtro onde apenas redações e textos motivadores com menos de 500 *tokens* foram levadas em consideração (Subseção 4.1.1).

5.1 ESTATÍSTICAS DESCRITIVAS

A estatística descritiva busca descrever um conjunto de dados a partir de várias técnicas de sumarização. Algumas das métricas produzidas são medidas de tendência central, como a média, mediana, e moda, medidas de variabilidade como desvio padrão e variância, e os valores máximos e mínimos. A Tabela 8 apresenta as principais dessas métricas para a nota na competência 2 e a nota final.

Tabela 8 – Estatísticas descritivas para as notas

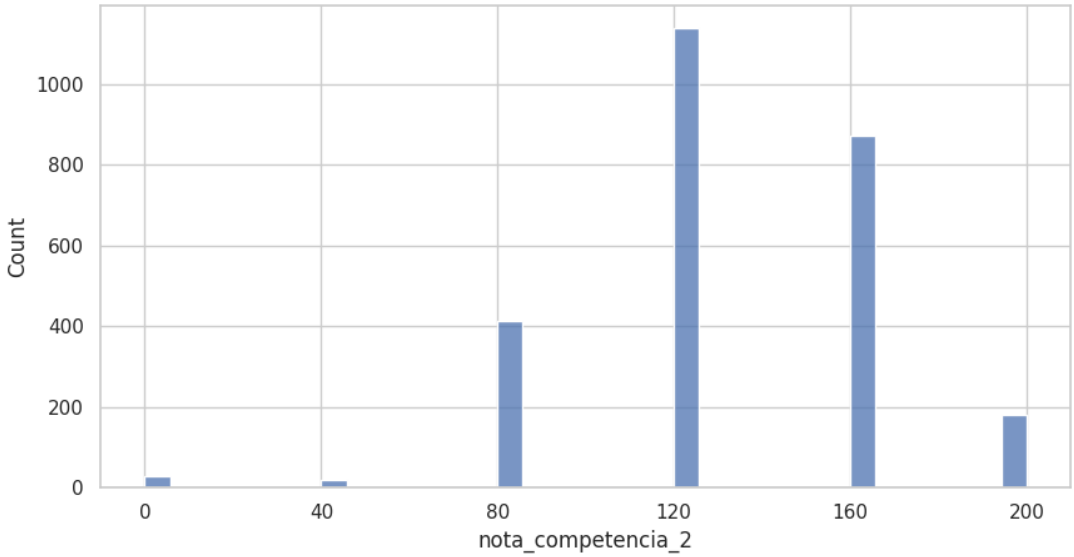
Nota competência 2		**Nota Final**	
Métrica	**Valor**	**Métrica**	**Valor**
Quantidade	2653	Quantidade	2653
Média	130,48	Média	618,00
Desvio padrão	36,16	Desvio padrão	168,02
Min	0,00	Min	0,00
25%	120,00	25%	520,00
50%	120,00	50%	640,00
75%	160,00	75%	760,00
Max	200,00	Max	1.000,00

Fonte: elaborado pelo autor

5.2 DISTRIBUIÇÃO DE NOTAS NA COMPETÊNCIA 2

As notas na segunda competência avaliativa variam de 0 a 200 em intervalos de 40 pontos. A Figura 11 demonstra a distribuição de notas em um gráfico do tipo histograma em intervalos de 40 pontos.

Figura 11 – Distribuição de notas na competência 2

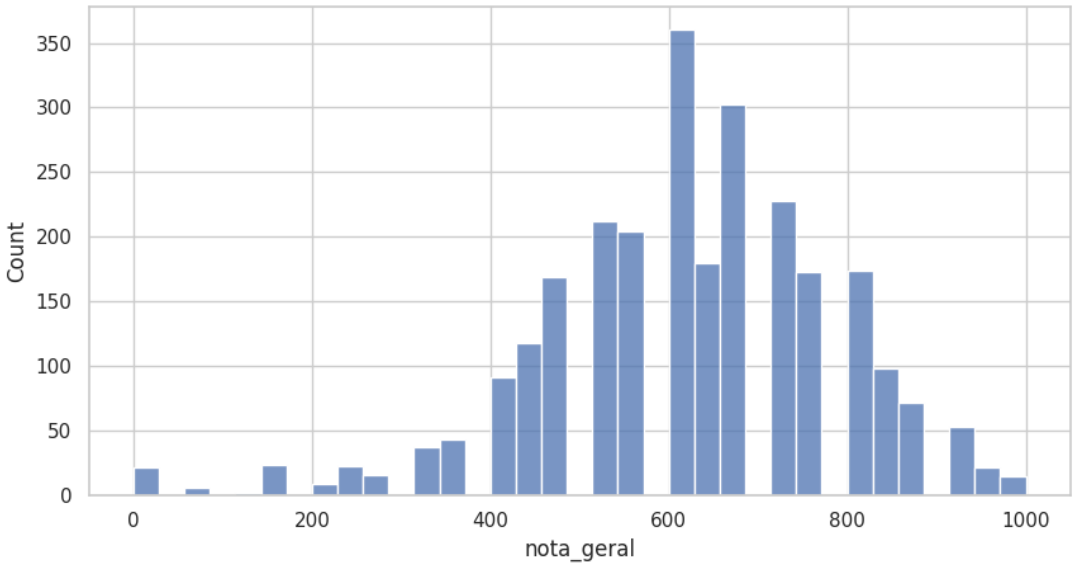


Fonte: elaborado pelo autor

5.3 DISTRIBUIÇÃO DE NOTAS FINAIS

As notas finais variam de 0 a 1000 sem intervalos de pontos. A Figura 12 demonstra a distribuição das notas finais em um gráfico do tipo histograma.

Figura 12 – Distribuição de notas finais



Fonte: elaborado pelo autor

5.4 CORRELAÇÃO ENTRE A NOTA NA COMPETÊNCIA 2 E A NOTA FINAL

Essa correlação pode ser confirmada através da matriz de correlação explicitada na Tabela 9 bem como do gráfico de regressão linear na Figura 13. Valores mais próximos de

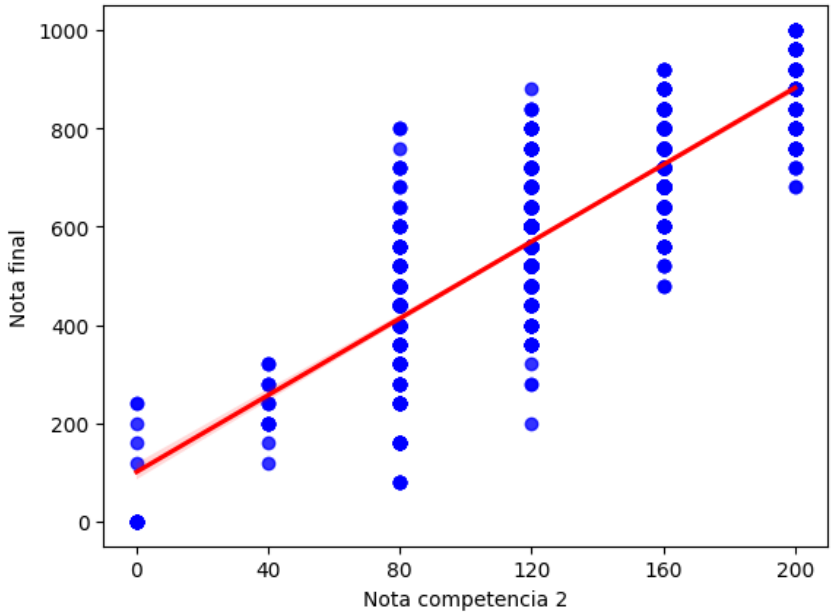
1 indicam uma correlação positiva, ao passo que valores mais próximos de -1 indicam uma correlação negativa. Valores próximos de 0 indica que não há correlação entre as variáveis.

Tabela 9 – Matriz de correlação entre notas

	Nota competência 2	Nota final
Nota competência 2	1,000000	0,831454
Nota final	0,831454	1,000000

Fonte: elaborado pelo autor

Figura 13 – Ajuste do modelo de regressão linear para nota na competência 2 e nota final



Fonte: elaborado pelo autor

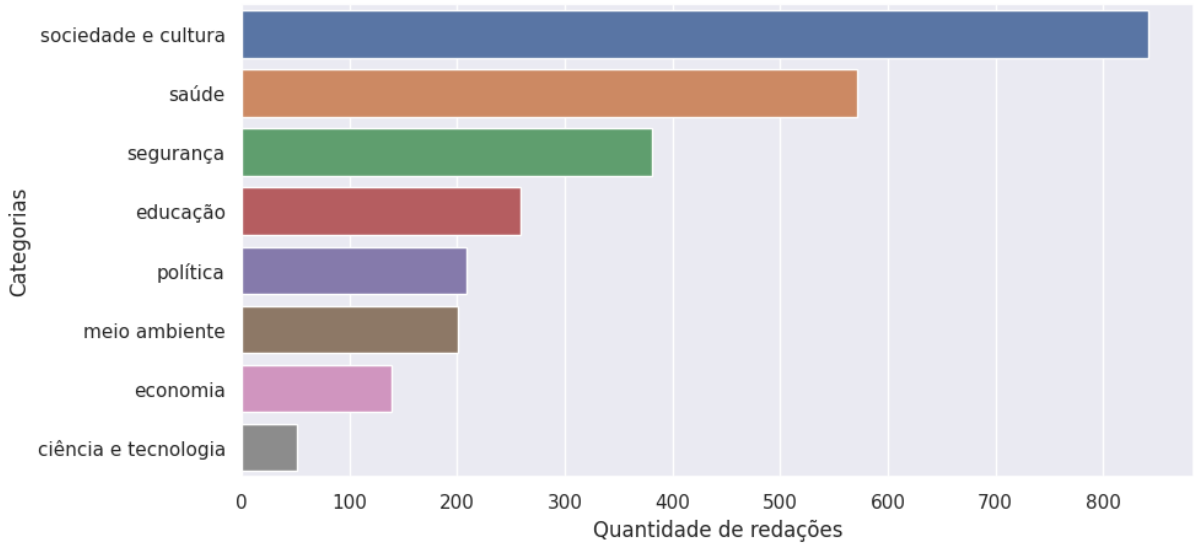
5.5 DISTRIBUIÇÃO DE CATEGORIAS POR TEMAS

Cada redação no conjunto de dados foi escrita sob um tema, e cada tema pertence a uma categoria, como por exemplo: Economia, Política, Educação, etc. A Figura 14 mostra a distribuição por temas ordenados por incidência.

5.6 ANÁLISE COMPARATIVA ENTRE NOTAS NA COMPETÊNCIA 2 E TEMAS

Ao agrupar as notas por categoria de tema e obter suas médias, se torna possível identificar as categorias que estudantes tendem a se distanciar mais ou menos de um dado tema, ou seja, atingirem notas maiores ou menores. Por exemplo, **Saúde** e **Segurança**, são categorias em que as redações no conjunto de dados apresentam maior aderência ao tema, ao passo que **Política** e **Ciência e tecnologia** são categorias mais penalizadas. Tal constatação pode significar

Figura 14 – Distribuição de categorias por tema



Fonte: elaborado pelo autor

que, no geral, estudantes possuem mais dificuldade em discorrer sobre certos temas sem perder de vista a ideia central proposta. A Tabela 10 traz os temas e suas respectivas notas médias ordenadas de forma descendente. E o diagrama de caixas, apresentado pela Figura 15 identifica o desempenho médio dos estudante por temas. Nota-se que os temas **Política** e **Ciência e Tecnologia** possuem os menores limites inferiores, podendo chegar a zero. No subconjunto analisado apenas esses dois temas possuem essa característica.

Tabela 10 – Nota média na competência 2 por categoria de tema

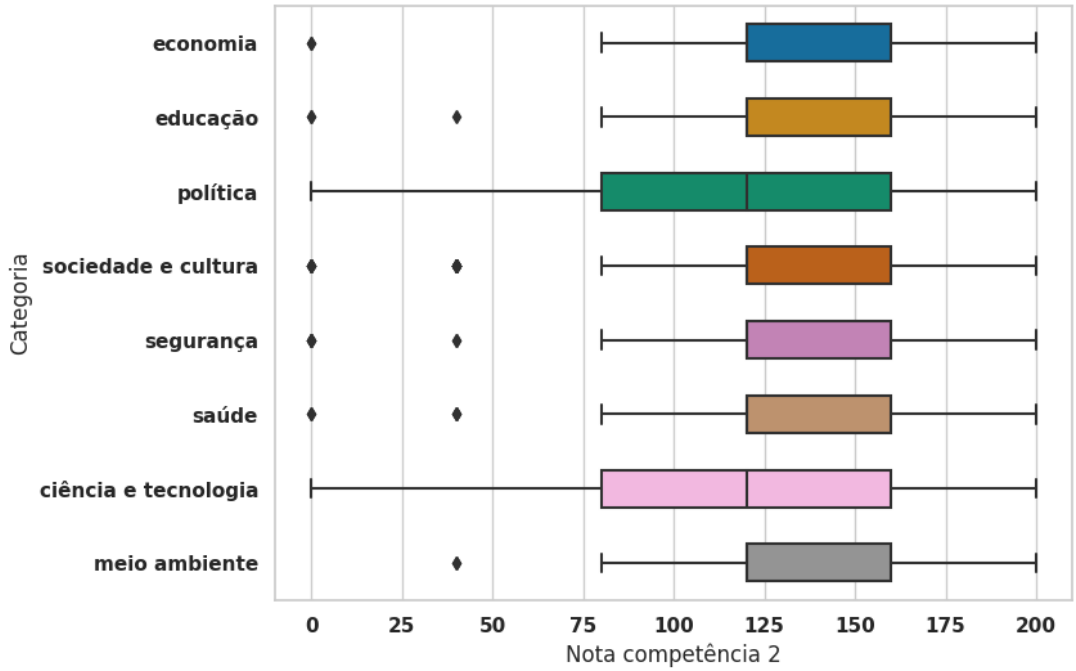
Tema	**Nota**
Saúde	136,88
Segurança	135,54
Meio ambiente	134,13
Educação	127,88
Economia	127,48
Sociedade e cultura	126,46
Política	124,02
Ciência e tecnologia	120,78

Fonte: elaborado pelo autor

5.7 ANÁLISE COMPARATIVA ENTRE NOTAS FINAIS E TEMAS

Ao considerar todas as competências avaliativas, temas como **Saúde**, **Segurança** e **Política** são melhores ranqueados, ao passo que temas como **Economia** e **Ciência e tecnologia** seguem consistentes com as menores pontuações como demonstra a Tabela 11. A Figura 16 traz a representação em formato de gráfico de caixas da distribuição das notas por categoria.

Figura 15 – Nota média na competência 2 por categoria de tema



Fonte: elaborado pelo autor

Tabela 11 – Média da nota final por categoria de tema

Tema	**Nota**
Saúde	665,15
Segurança	658,58
Política	618,56
Educação	605,71
Meio ambiente	604,98
Ciência e tecnologia	585,10
Sociedade e cultura	583,23
Economia	576,69

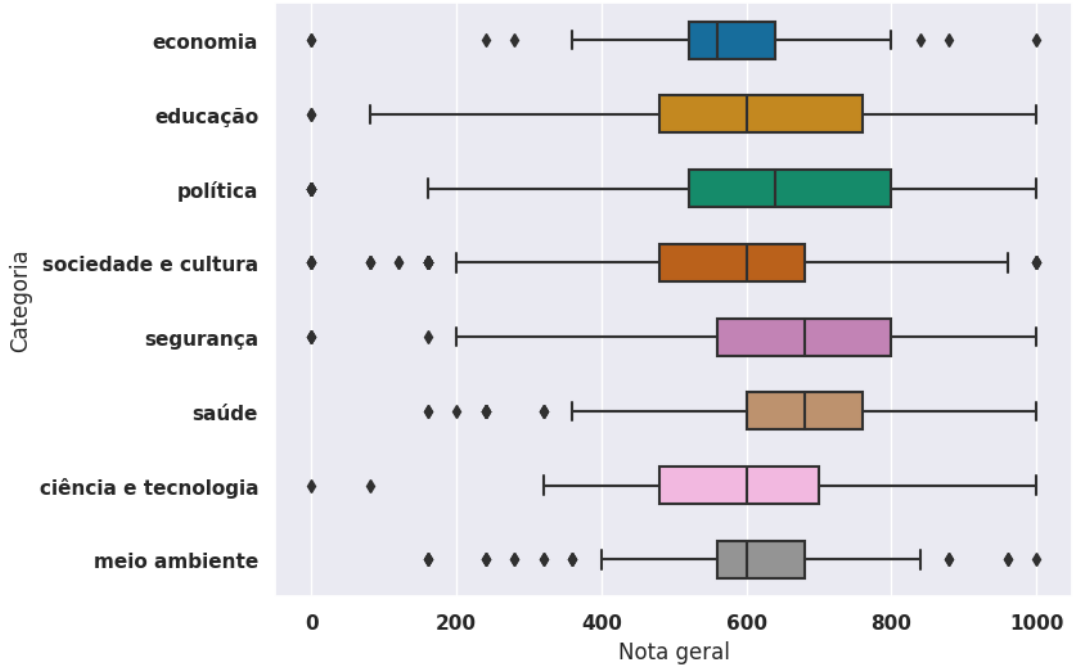
Fonte: elaborado pelo autor

5.8 DISTRIBUIÇÃO DE SENTENÇAS

A quantidade de sentenças em cada documento é uma métrica relevante dado que essa pode ser uma das possíveis estratégias adotadas para o processamento dos textos. Em PLN a definição do que é considerado uma sentença não é fixa e pode variar a depender da tarefa à ser resolvida. O presente trabalho, entretanto, utilizou a biblioteca *spacy sentencizer* [1] que fragmentou os textos em sentenças a partir de sua configuração padrão. A distribuição de sentenças para as redações, Figura 18, revela que a grande maioria possuem entre 4 e 11 sentenças. Contudo, é possível observar algumas anomalias, como algumas redações com um número muito baixo de sentenças, algo entre 1 e 5, e outras com uma quantidades estranhamente altas, entre

[1] https://spacy.io/api/sentencizer

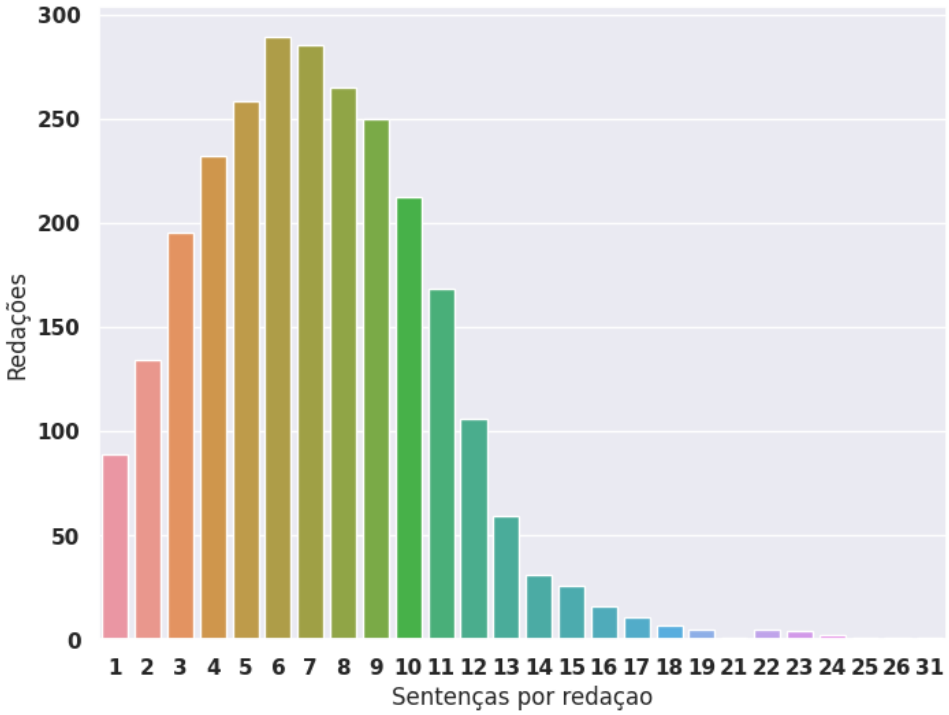
Figura 16 – Média da nota final por categoria de tema



Fonte: elaborado pelo autor

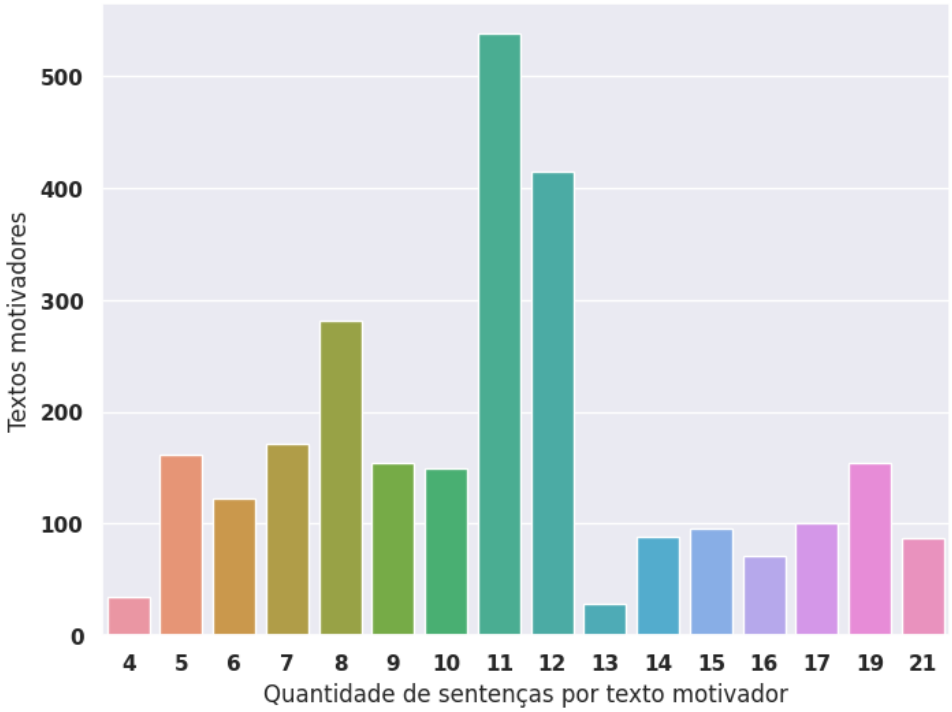
25 e 31 sentenças. Por sua vez, a distribuição de sentenças nos textos motivadores representa uma estabilidade maior em relação a quantidade de sentenças. A maioria dos textos possuem entre 100 e 200 sentenças. E alguns poucos, possuem entre 400 e 500.

Figura 17 – Distribuição da quantidade de sentenças por redação



Fonte: elaborado pelo autor

Figura 18 – Distribuição da quantidade de sentenças por texto motivador

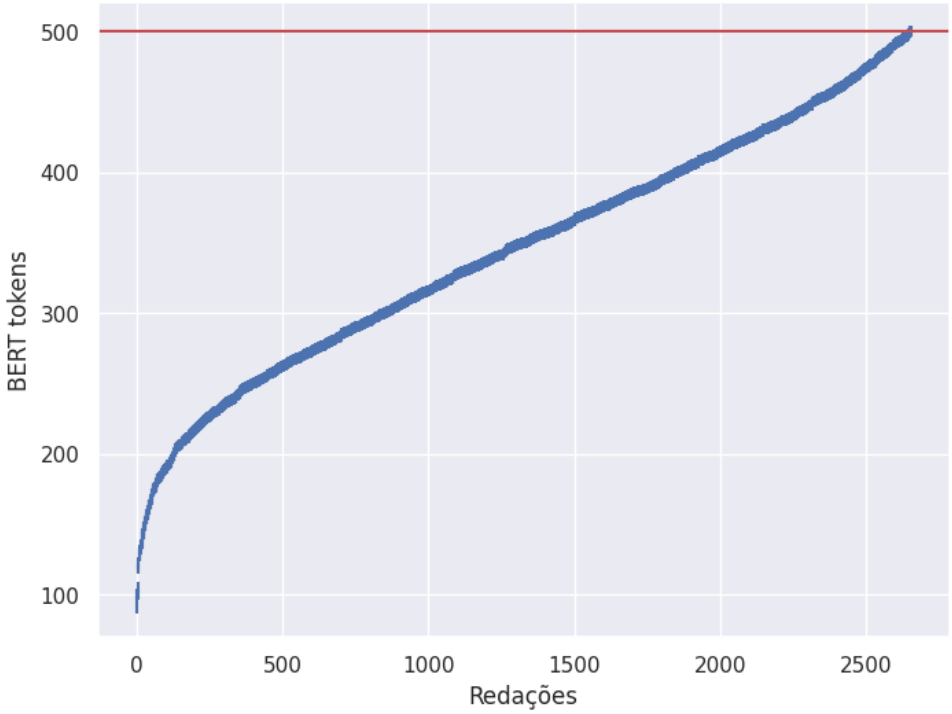


Fonte: elaborado pelo autor

5.9 DISTRIBUIÇÃO POR TOKENS

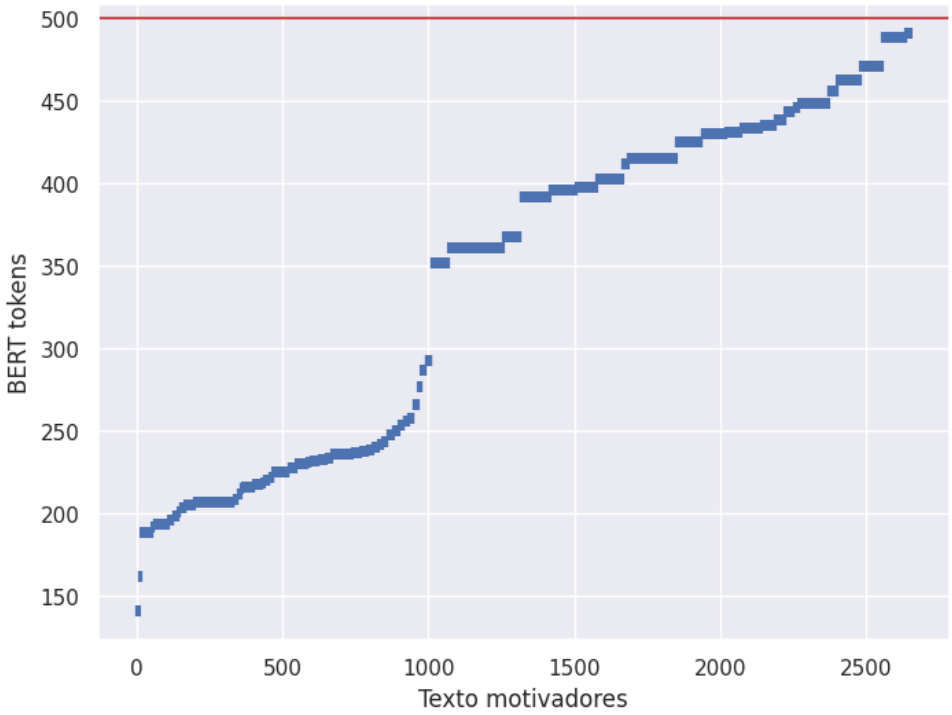
Como dito nas seções anteriores, o modelo BERT possui um limite máximo de 512 *tokens*. Nesse sentido, importa que seja identificado no conjunto de dados os documentos que extrapolam esse limite. A Figura 19 nos permite observar que das poucas mais das 2500 redações no conjunto de dados, nenhuma excede o limite 512 de *tokens*. Similarmente a Figura 20 apresenta o mesmo comportamento, de todos os textos motivadores presentes no conjunto de dados, nenhum ultrapassa o limite máximo de *tokens* estabelecido pelo BERT.

Figura 19 – Distribuição da quantidade de tokens por redação



Fonte: elaborado pelo autor

Figura 20 – Distribuição da quantidade de tokens por texto motivador



Fonte: elaborado pelo autor

6 EXPERIMENTOS E RESULTADOS

O presente capítulo detalha os experimentos, métodos, e resultados obtidos ao longo da construção de um modelo de regressão e classificação capaz de aferir a aderência semântica de uma dada redação ao seu respectivo texto motivador. Inicialmente, será detalhado os experimentos preliminares que respondem a hipótese da proposta. O resultados desses experimentos determinarão a estratégia utilizada para gerar os *embeddings* contextualizados que serão utilizados para treinamento dos modelos. Compararemos os resultados obtidos com o modelo BERTimbau nas versões *base* e *large*, e também com o *framework* SBERT. Após a definição da estratégia a ser utilizada, o capítulo apresentará a série de etapas necessárias para a formatação das entradas de texto para um formato que o BERT seja capaz de processar. Será apresentado também as definições de hiper-parâmetros comuns aos dois modelos desenvolvidos, bem como o embasamento teórico por trás dessas decisões. Finalmente, será detalhado as etapas de treinamento e validação do modelo de regressão e classificação, seguido de seus respectivos desempenhos na tarefa proposta. Além das etapas mencionadas, os resultados de cada modelo será acompanhado de uma breve seção dedicada a discutir estes mesmos resultados, e apontar possíveis limitações e aprimoramentos. Por fim, o capítulo encerra apresentado a possibilidade de reprodução dos experimentos realizados, bem como a disponibilidade de uma função capaz realizar predições através do modelo de regressão e classificação.

6.1 MATERIAIS E MÉTODOS

Os experimentos foram realizados na plataforma *Google colaboratory* na versão PRO+. A plataforma oferece uma série de *Graphic Processing Units* GPUs, entre elas a Tesla T4, utilizada no presente trabalho. Os modelos foram treinados em todas as taxas de aprendizado recomendadas pelos autores, sendo elas: 5e-5, 3e-5, 2e-5 (DEVLIN et al., 2018), por cinco épocas. O número de épocas também está dentro do recomendado, isso porque um modelo pré-treinado não demanda muitas épocas de treinamento até uma possível convergência. O tamanho do *batch*, isto é, a quantidade de entradas que o modelo processará por vez, foi definido em 8. O valor recomendado para o *batch* é 32, porém o custo em memória para esse valor é inviável para o ambiente utilizado. O valor de *épslon* para o otimizador Adam foi definido em 1.00E-08, também por recomendação dos autores (DEVLIN et al., 2018). A linguagem de programação Python foi utilizada ao longo de todo o desenvolvimento na versão 3.10.12. Uma das principais APIs - *Application Programming Interface*, do inglês, utilizadas disponibilizada pela biblioteca *transformer*[1] desenvolvida pela comunidade **HuggingFace**[2]. Atualmente, essa biblioteca está estabelecida como a melhor alternativa para manipular a arquitetura BERT em *PyTorch*. O código fonte e os dados para reproduzir os experimentos e resultados reportados

[1] https://huggingface.co/docs/transformers/index
[2] https://huggingface.co/

neste trabalho estão disponíveis no GitHub[3].

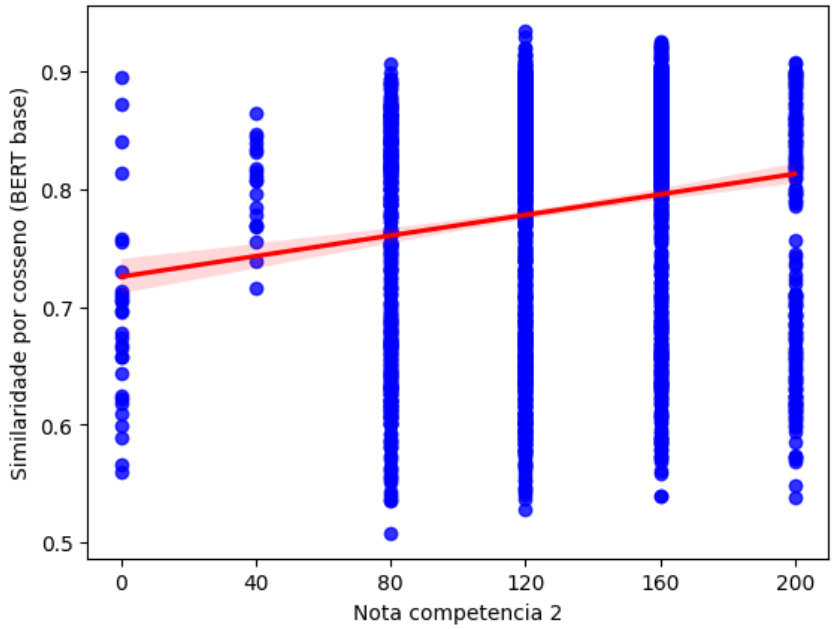
6.2 RESULTADOS COM *EMBEDDINGS* DO BERT DE DOCUMENTOS INTEIROS

Na seções a seguir os resultados dos experimentos utilizando *embeddings* de documentos inteiros com o BERTimbau base e *large*.

6.2.1 BERTimbau base

A Figura 21 revela uma correlação de 0,16 entre as notas na competência 2 e a similaridade por cosseno entre redações e textos motivadores. A linha diagonal indica que na medida em que a nota aumenta a compatibilidade semântica entre redação e texto motivador também tende a subir.

Figura 21 – BERTimbau base: Correlação entre similaridade de cosseno entre redações e textos motivadores e a nota na competência 2



Fonte: elaborado pelo autor

Ainda que tal correlação possa ser considerada sútil, ela parece corroborar a hipótese inicial de que de fato existe um correspondência semântica entre redações e textos motivadores.

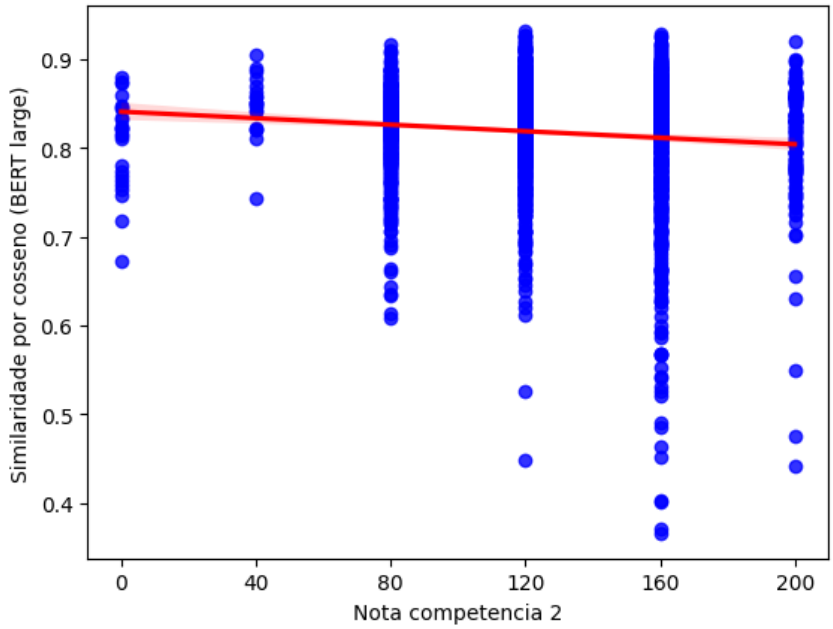
6.2.2 BERTimbau large

O modelo *large* é invariavelmente mais pesado que o *base* devido sua grande quantidade de parâmetros. Exatamente por esse motivo, é possível que o modelo se torne mais sensível à captação de nuances e contextos. Essa hipótese pode ser confirmada através dos

[3] https://github.com/RaphaelSilv/enem-bertimbau

resultados da regressão, uma correlação negativa de -0,10. É como se um modelo com mais parâmetros acabasse abstraindo exageradamente algumas representações. O que para a tarefa de aferir a similaridade entre dois documentos pode ser um problema.

Figura 22 – BERTimbau large: Correlação entre similaridade de cosseno entre redações e textos motivadores e a nota na competência 2



Fonte: elaborado pelo autor

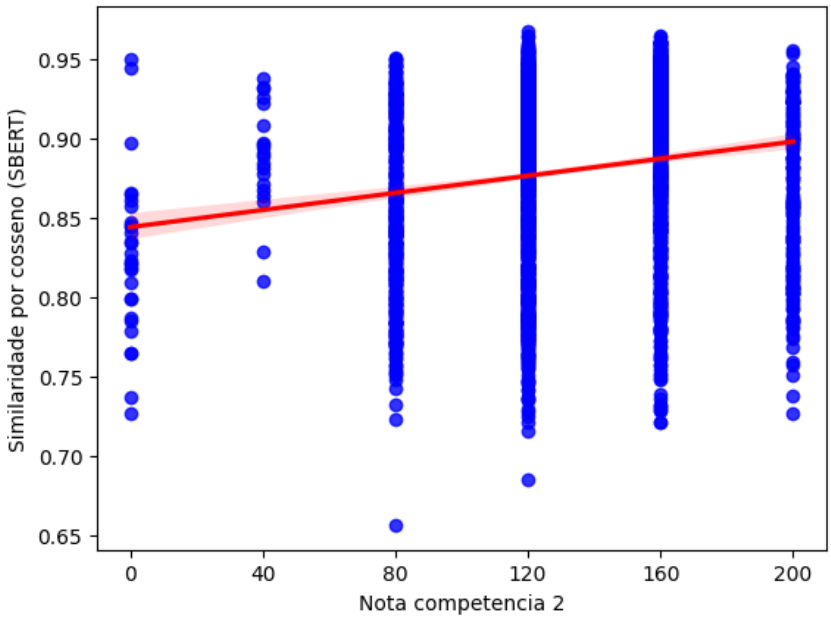
6.3 RESULTADOS COM *EMBEDDINGS* SBERT DE SENTENÇAS

Os resultados com documentos segmentados segundo sentenças e uso de *embeddings* das mesmas revelam uma correlação de 0,15 entre as notas na competência 2 e a similaridade do cosseno gerado a partir da estratégia de *pooling*. *Pooling* é essencialmente uma forma de reduzira múltiplas representações vetoriais em uma representação unidimensional. Existem diferentes estratégias de *pooling* como MEAN, MAX e MIN, para o modelo SBERT a técnica MEAN é a utilizada por padrão (REIMERS; GUREVYCH, 2019). Como é de se esperar, estratégias de agregação e redução de dimensionalidade como a utilizada, implica em alguma perda de informação. Entretanto, percebe-se que hipótese de aderência entre os universos semânticos de redações e textos motivadores continua sendo minimamente reforçada.

A partir dos resultados dos experimentos iniciais, decidimos que técnicas de agregação como as utilizadas pelo modelo SBERT podem representar em significativa perda de informação, devido a natureza de tarefas de agregação e redução de dimensionalidade. Logo, a estratégia de lidar com documentos inteiros se revelou como a mais adequada.

Desse modo, decidimos utilizar a versão *base* do modelo BERTimbau para lidarmos com documentos inteiros uma vez que o modelo *large* apresentou potenciais problemas na

Figura 23 – Correlação entre a nota na competência 2 e similaridade do cosseno para documentos sentenciados



Fonte: elaborado pelo autor

generalização excessiva de contextos. Ademais, com a escolha do modelo na versão *base*, ganha-se também em performance.

6.4 PRÉ-PROCESSAMENTO

Nesta etapa foi realizada a limpeza de todos os textos em nosso conjunto de dados, pela a remoção de carácteres especiais e referencias jornalísticos, como por exemplo, *["UOL Leia o texto na íntegra"]*. Referências como essas são abundantes nos textos motivadores, e como não acrescentam sentido ao texto, podem ser removidas sem maiores prejuízo. Após limpos, os textos foram devidamente processados de modo a estarem no formato requerido pelo modelo de treinamento. Esse processo foi detalhadamente exposto na sub-seção 6.4.1.

6.4.1 Formatação de entradas BERT

Os textos que serão processados pelo BERT durante treinamento precisam estar formatados em conformidade com o que o modelo define. Cada segmento de texto precisa ser segmentado em *tokens* e possuir *tokens* especiais no início e no fim de cada sentença, isso é melhor detalhado na seção 2.6.1; documentos precisam possuir tamanho fixo, o que só pode ser obtido através do truncamento do textos ou pela técnica de *padding*; e *tokens* reais precisam ser diferenciados de *tokens* adicionados pela técnica de *padding*, à saída dessa etapa dá-se o nome de máscaras de auto-atenção.

Os chamados BERT *tokens*, representam a menor parte de uma palavra compreensível pelo modelo. Cada *token* possui um identificador que funciona como um índice no vocabu-

lário do *tokenizador*. O algorítimo de segmentação utilizado pelo *tokenizador* é o *WordPiece* (WANG; YAN; WU, 2018), especialmente útil para segmentação não-supervisionada de textos e identificação de palavras estranhas ao vocabulário do modelo. O algoritmo também é capaz de dividir palavras em sub-palavras ou carácter. Considere o exemplo a seguir extraído de um dos textos motivadores do conjunto de dados:

"Se houver um novo entendimento do STF, a prisão só poderá ocorrer depois do esgotamento das possibilidades de recurso no superior tribunal de justiça e, a seguir, no próprio supremo, que funcionariam como 3ª. e 4ª. instâncias."

O mesmo trecho segmentado pelo BERT:

’[CLS]’, ’se’, ’houver’, ’um’, ’novo’, ’entendimento’, ’do’, ’s’, ’##t’, ’##f’, ’,’, ’a’, ’prisão’, ’só’, ’poderá’, ’ocorrer’, ’depois’, ’do’, ’esgo’, ’##tamento’, ’das’, ’possibilidades’, ’de’, ’recurso’, ’no’, ’superior’, ’tribunal’, ’de’, ’justiça’, ’e’, ’,’, ’a’, ’seguir’, ’,’, ’no’, ’próprio’, ’supremo’, ’,’, ’que’, ’funcionar’, ’##iam’, ’como’, ’[UNK]’, ’.’, ’e’, ’[UNK]’, ’.’, ’instância’, ’##s’, ’.’, ’[SEP]’

Conversão de BERT *tokens* para identificadores no vocabulário, i.e. *Token IDs*:

101, 530, 20075, 222, 1160, 12884, 171, 13196, 22324, 117, 123, 4025, 1203, 8209, 7736, 700, 171, 8633, 1869, 366, 12522, 125, 8190, 202, 2886, 9497, 125, 6722, 122, 117, 123, 3866, 117, 202, 1946, 21688, 117, 179, 11331, 923, 271, 100, 119, 122, 100, 119, 17231, 22281, 119, 102

Mascaramento:

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

Nesse caso, todos os *tokens* identificados são válidos, caso algum não fosse o valor 0 seria encontrado na sequência. A biblioteca *transformers* possibilita que toda essa formatação seja feita em apenas algumas linhas de código:

Algoritmo 6.1 – Tokenização de texto com a biblioteca transformers

```
from transformers import AutoTokenizer

model_name = 'neuralmind/bert-base-portuguese-cased'
tokenizer = AutoTokenizer.from_pretrained(model_name, do_lower_case=False)

for tm in textos_motivadores:
    # encode_plus will:
    #   (1) Tokeniza texto motivador.
    #   (2) Add `[CLS]` token no inicio do texto.
    #   (3) Add `[SEP]` token no fim do texto.
    #   (4) Mapeamento de tokens para seus IDs.
    #   (5) Padding `max_length`
```

```
#   (6) Cria attention masks para os tokens do tipo [PAD].
tm_encoded_dict = tokenizer.encode_plus(
                    tm,                          # Texto motivador.
                    add_special_tokens = True,   # Add '[CLS]' and '[SEP]'.
                    max_length = 512,            # Pad redacao.
                    padding = 'max_length',
                    return_attention_mask = True, # Constroi attn. masks.
                    return_tensors = 'pt',       # Retorna pytorch tensors.
                )


tm_input_ids.append(tm_encoded_dict['input_ids'])
tm_attention_masks.append(tm_encoded_dict['attention_mask'])
```

6.4.2 Definição de hiper-parâmetros

O modelo BERT utilizado será o modelo BERTimbau (MARINHO; ANCHIêTA; MOURA, 2021) em sua versão *base*. A grande maioria dos hiper parâmetros foram definidos com base nas recomendações do artigo oficial do BERT. Alguns outros foram definidos a partir de experimentações e também das limitações de nosso ambiente de teste. A taxa de aprendizagem é um hiper parâmetro crítico no treinamento de modelos de aprendizagem profunda. Ela influencia o tamanho dos passos que o modelo dá em direção ao mínimo da função de perda. Os autores do BERT recomendam três taxas aprendizado, são elas: 5e-5, 3e-5, 2e-5 (DEVLIN et al., 2018).

Ambos os treinamentos utilizaram a técnica de validação *k-fold*, explicitada na Seção 2.9. O número de *folds* definido foi 10, ou seja, 10-*fold*. O número de épocas para cada *fold* foi também 3. Esse valor também segue a recomendação do artigo principal do BERT que salienta que o modelo pré-treinado via de regra não precisa de muitas épocas até uma possível convergência. Para facilitar a visualização dos resultados, apenas as médias das principais métricas serão apresentadas na presente seção.

6.5 MODELO DE REGRESSÃO

O experimento a seguir, busca explorar e avaliar a eficiência do modelo BERT pré-treinado para a língua portuguesa na tarefa de atribuir notas à redações no formato ENEM com base em sua correspondência semântica com seus respectivos textos motivadores. A partir de técnicas de *fine-tunning* o modelo será adaptado para a atribuir um único valor contínuo para redações em uma tarefa de regressão. O treinamento do modelo foi conduzido a partir dos parâmetros definidos na subseção 6.4.2. Os detalhes do treinamento são apresentados a seguir.

6.5.1 Treinamento

O BERT foi treinado para ser essencialmente um classificador, porém o modelo possibilita sua adaptação para uma série de tarefas derivadas em PLN, do inglês *downstream tasks* (DEVLIN et al., 2018). Uma dessas tarefas, é a regressão e para configurá-lo para esse fim,

basta definir a variável **num_labels** (i.e. número de classes) para 1, quando da instanciação do modelo.

Algoritmo 6.2 – Inicialização do modelo BERT para tarefa de regressão

```
from transformers import AutoModelForSequenceClassification

model_name = 'neuralmind/bert-base-portuguese-cased'
bertimbau = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=1).cuda()
```

Uma vez que os textos foram limpos, pré-processados, e formatados para serem processados pelo BERT, inicia-se o processo de treinamento. A partir das entradas formatadas de cada redação e texto motivador, são extraídos os *embeddings* da última camada do BERT. Em seguida, dois vetores de tamanho 768 são gerados. Esse vetores contém toda informação da entrada textual codificada ao longo das 12 camadas do modelo. Cada entrada também acompanha a nota da redação naquela categoria. A essa altura a nota já foi processada e transformada em tensores do tipo ponto flutuante. Sendo assim, o próximo passo é submeter as entradas ao modelo customizado de rede neural.

A principal customização do nosso modelo, consiste em sua habilidade de correlacionar uma dada redação ao seu respectivo texto motivador. Isso acontece através da concatenação dos dois vetores gerados pelo BERT, que transforma dois vetores de tamanho 768 em único vetor de tamanho 1536. Em seguida, o modelo submete o vetor concatenado a duas camadas densas, que progressivamente reduzem sua dimensionalidade para 512, e 128. Finalmente, a última camada densa reduz a dimensionalidade do vetor de entrada para 1, resultando sua saída final, um único valor contínuo, a saber a nota estimada.

Algoritmo 6.3 – Extração de BERT embeddings e concatenação dos vetores de redação e texto motivador

```
def forward(self, r_input_ids, r_attention_masks, tm_input_ids, tm_attention_masks):

    r_outputs = bertimbau(r_input_ids, r_attention_masks, output_hidden_states=True)
    r_class_label_output = r_outputs.hidden_states[-1][:,0,:]

    tm_outputs = bertimbau(tm_input_ids, tm_attention_masks, output_hidden_states=True)
    tm_class_label_output = tm_outputs.hidden_states[-1][:,0,:]

    concatenated_output = torch.cat((r_class_label_output, tm_class_label_output), dim=-1)
    outputs = self.regressor(concatenated_output)

    return outputs
```

A variável **notas** é a variável alvo, e durante o treinamento auxilia o modelo em suas predições a computar o valor de perda e a consequentemente atualizar seus pesos. O código fonte para o experimento está disponível no repositório público **enem-bertimbau**[4]. A seção a seguir apresenta os resultados da etapa apresentada.

[4] https://github.com/RaphaelSilv/enem-bertimbau

6.5.2 Resultados

A Tabela 12 e a Tabela 13 trazem a média dos resultados obtidos para as métricas perda *i.e. Loss* e MAE, durante as etapas de treinamento e validação. Os resultados para diferentes taxas de aprendizado (LR) revelam um padrão. Os valores obtidos tanto para treinamento como para validação estão bem próximos entre si. O *fold* treinado sob a menor taxa de aprendizado, 2e-5, possui o menor valor do Erro Médio Absoluto, 128,05 e a menor perda de Erro Médio Quadrático, 17691,05. Na etapa de validação, a performance média dos *folds* treinados sob a mesma taxa de aprendizado foi de 127,19 para MAE e 17482,00 de perda MSE. A consistência entre os valores indicam que a etapa de validação está coerente com a etapa de treinamento, e inclusive generalizando sutilmente melhor.

Tabela 12 – Performance de treinamento para diferentes taxas de aprendizado ao longo de 10 *folds*

LR	**MSE *Loss***	**MAE**
5e-5	17710,08	128,13
3e-5	17783,27	128,41
2e-5	17691,05	128,06

Fonte: elaborado pelo autor

Tabela 13 – Performance de validação para diferentes taxas de aprendizado ao longo de 10 *folds*

LR	**MSE *Loss***	**MAE**
5e-5	17506,06	127,28
3e-5	17571,78	127,53
2e-5	17482,00	127,19

Fonte: elaborado pelo autor

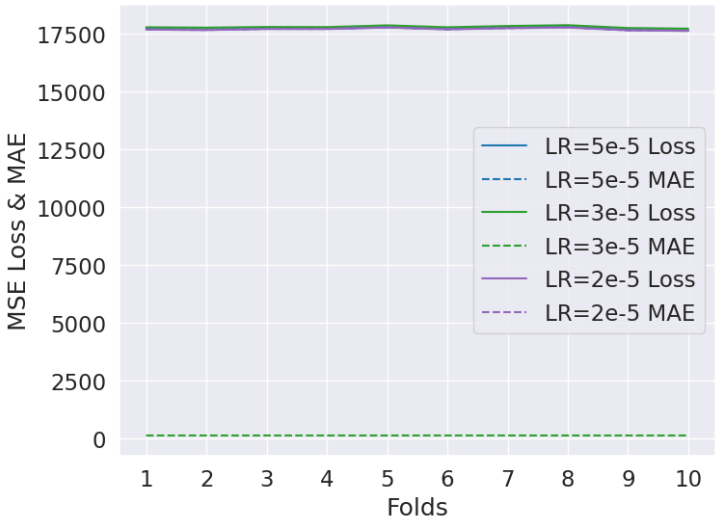
A pior performance média, entretanto, é a taxa de aprendizado 3e-5, uma vez que os valores mais altos de MAE e de MSE são obtidos ao longo dos *folds*. Durante treinamento, o valor da perda atinge 3e-5 17.783,27 contra 128,41 para o MAE. Esses também foram os maiores valores observados na etapa de validação, 17571,78 127,53, respectivamente.

Tanto na Figura 24 quanto na Figura 25 é possível observar o progresso dos *folds* treinados e validados sob diferentes taxas de aprendizado. Ambos os valores de perda e MAE estão próximos e a variação entre os valores de cada taxa é quase imperceptível. O que pode indicar o impacto quase nulo da taxa de aprendizado.

6.5.3 Discussão

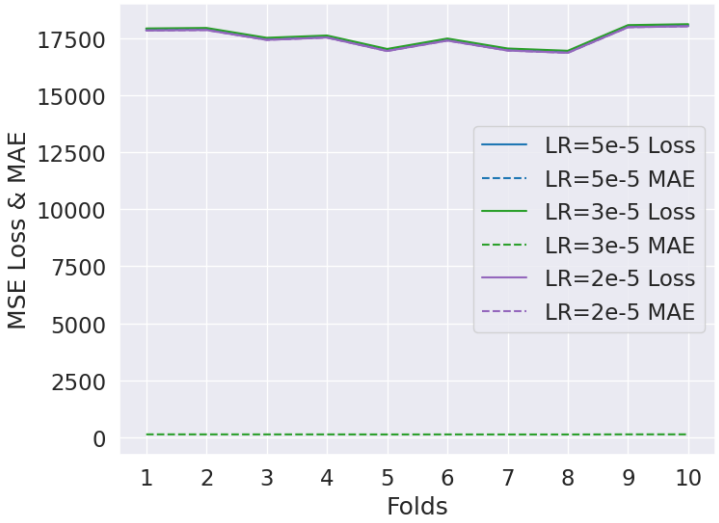
A performance dos modelos ao longo das diferentes taxas de aprendizado *i.e. folds* revelam um desempenho no geral insatisfatório. Entretanto, os resultados apontam que o modelo sob a taxa de aprendizado 5e-5 foi o que atingiu melhor performance durante treinamento e validação. Ou seja, seus valores médios de perda e do MAE foram os menores entre as taxas de

Figura 24 – Performance de treinamento utilizando técnica 10-*fold* ao longo das diferentes taxas de aprendizado



Fonte: elaborado pelo autor

Figura 25 – Performance de validação utilizando técnica 10-*fold* ao longo das diferentes taxas de aprendizado



Fonte: elaborado pelo autor

aprendizado observadas. Isto indica que no geral, as predições deste modelo são mais acuradas e consistentes. Entretanto, os menores valores do MAE encontrados durante treinamento e validação, 128,06 e 127,19, respectivamente, indicam que o erro médio das predições do modelo podem estar até 128 pontos acima ou abaixo da nota real, que varia de 0 à 200.

Esse valor é considerado muito alto, e em última análise apresenta sérios problemas de confiabilidade. Nesse sentido, devido a natureza crítica da tarefa em a ser resolvida, aferir a nota de uma redação a uma dada competência, não é recomendado o uso do modelo de regressão, uma vez que não é possível confiar em seu poder preditivo.

Os motivos por trás da baixa performance do modelo pode estar relacionadas à diferentes fatores, entre estes destacam-se: a possível baixa qualidade dos dados, baixa representação de *features*, ou seja, conjunto de dados desbalanceado, ou ainda a própria arquitetura do modelo.

Aprimorar os resultados obtidos passa por revisitar o processo de extração de *features* e garantir a qualidade dos dados. Uma *pipeline* mais robusta de limpeza dos dados, estratégias para avaliar a qualidade dos *embeddings* de palavras gerados pelo BERT, o uso de diferentes técnicas de extração de *embeddings*, como agregação via *pooling*, ou ainda o uso de janelas deslizantes (DIAS et al., 2022) podem ser parte da remediação do problema. Avaliar a performance do modelo com a variação de outros hiper-parâmetros também deve ser considerado. Por fim, não se deve descartar a possibilidade de a natureza da tarefa ser demasiadamente complexa para o modelo escolhido.

6.6 MODELO DE CLASSIFICAÇÃO

Assim como o modelo de regressão, o presente modelo busca explorar e avaliar a eficiência do BERT pré-treinado para a língua portuguesa na tarefa de atribuição de notas à redações no formato ENEM. A partir de técnicas de *fine-tunning* o modelo será adaptado para para classificar uma dada redação à uma das 6 possíveis classes de notas, a saber [0 ,40, 80, 120, 160, 200]. O treinamento do modelo foi conduzido a partir dos parâmetros definidos na subseção 6.4.2. Os detalhes do treinamento são apresentados a seguir.

6.6.1 Treinamento

Similarmente a configuração realizada durante os experimentos com o modelo de regressão, o modelo BERT também precisa ser inicializado. O Algoritmo 6.4 apresenta o número de *labels* para o modelo de classificação (i.e., 6).

Algoritmo 6.4 – Inicialização do modelo BERT para tarefa de classificação

```
from transformers import AutoModelForSequenceClassification

model_name = 'neuralmind/bert-base-portuguese-cased'
bertimbau = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=6).cuda()
```

Mais tarde, a função de perda, recebe como entrada um conjunto de *logits* e internamente aplica uma função de *softmax*, que converte valores reais em probabilidades antes do calculo de perda.

Logits são vetores de valores reais do tamanho do número de *labels*. Cada valor no vetor, representa a confiança do modelo na escolha de uma determinada classe. Quanto maior o valor de um *logit*, maior a confiança do modelo na escolha daquela classe.

6.6.2 Resultados

Os resultados obtidos para a etapa de treinamento são bastante positivos para todas as taxas de aprendizado. Os valores de acurácia e F1 *score* estão sempre próximos, o que sugere um balanceamento entre a precisão e a sensibilidade (*Recall*). Além disso, a maioria de

seus valores estão consistentemente acima de 70%. Os valores de perda, entretanto podem ser considerados altos.

De modo geral, os resultados sugerem que o melhor desempenho é do modelo treinado sob a taxa de aprendizado em 5e-5. O modelo apresenta o menor valor de perda, 0,63 e os valores de acurácia e F1 score, também são os melhores atingindo 80,81 e 0,79, respectivamente. O segundo melhor resultado, ainda na etapa de treinamento, é do *fold* treinado sob a taxa 3e-5. O valor de perda observado foi 0,72, apenas 0,09 acima do valor encontrado no primeiro resultado e 0,10 abaixo do valor encontrado no *fold* treinado sob a taxa 2e-5.

Similarmente, a mesma tendência é observada nos *folds* de avaliação. O *fold* sob a taxa 5e-5, também registra os melhores resultados, sendo 81,73 de acurácia e 0,80 no F1 score. Esse valor sugere a boa capacidade do modelo em generalizar frente a dados ainda não vistos.

Tabela 14 – Performance de treinamento para diferentes taxas de aprendizado ao longo de 10 *folds*

LR	Avg. Acurácia	Avg. F1-score	Avg. Loss
5e-5	80,81	0,79	0,63
3e-5	76,27	0,73	0,72
2e-5	71,68	0,68	0,82

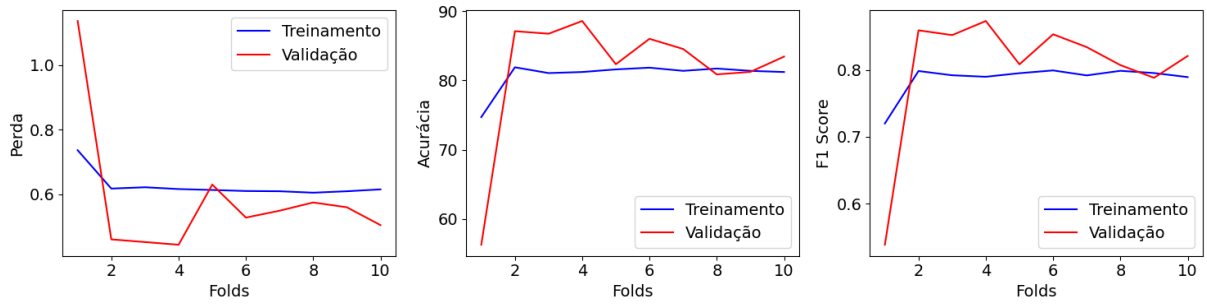
Fonte: elaborado pelo autor

Tabela 15 – Performance de validação para diferentes taxas de aprendizado ao longo de 10 *folds*

LR	Avg. Acurácia	Avg. F1-score	Avg. Loss
5e-5	81,73	0,80	0,58
3e-5	78,71	0,75	0,66
2e-5	72,28	0,69	0,80

Fonte: elaborado pelo autor

Figura 26 – Performance de treinamento e validação utilizando técnica *10-fold* para taxa de aprendizado em 5e-05



Fonte: elaborado pelo autor

6.6.3 Discussão

No geral, percebe-se uma tendência, na medida em que a taxa de aprendizado sobe, acurácias e F1 score também sobem, e o valor de perda decresce. Isto pode implicar que o

modelo se beneficia de maiores atualizações nos pesos com esta taxa de aprendizagem. É como se o modelo fosse capaz de aprender mais eficazmente com taxas mais altas capturando melhor as nuances semânticas dos textos.

A Figura 26 apresenta o desempenho das métricas do modelo treinado sob a taxa de aprendizado em 5e05 ao longo de 10 *folds*. Na figura mais a direita, os resultados de perda, do inglês *loss*, sugerem uma média de aproximadamente 0,60 ao longo dos 10 *folds*. A tabela ao meio, apresenta a variação da acurácia e sugere uma estabilidade levemente superior a 80% para ambas as etapas de treinamento e validação ao longo dos 10 *folds*. A tabela mais à esquerda, apresenta os resultados para o F1 *score*, e sua variação sugere uma performance média acima de 70% ao longo dos 10 *folds*. Para todas as métricas nota-se que geralmente, os conjuntos de validação possuem desempenho levemente superior ao o seu respectivo conjunto de treinamento. Isso sugere boa capacidade de generalização do modelo. Percebe-se também que os valores de todas as métricas citadas são compatíveis com os resultados apresentados nas tabelas Tabela 14 e Tabela 15.

Os resultados observados também são apoiados pelo método de validação cruzada 10-*fold* que possibilita avaliar a capacidade do modelo de lidar com diferentes subconjuntos de dados a partir de diferentes partições.

É importante salientar entretanto, que a taxa de aprendizado isoladamente pode não ser um fator relevante o suficiente para impactar significativamente o desempenho do modelo. É possível que a combinação de ajustes em diferentes hiper-parâmetros, aliado a técnicas de processamento alternativas possam resultar em melhores resultados.

6.7 REPRODUÇÃO DOS EXPERIMENTOS E APLICAÇÕES

Todos os experimentos detalhados, poderão ser fielmente reproduzidos uma vez que os dados, inclusive dos *folds*, para cada taxa de aprendizado serão disponibilizados quando da publicação deste trabalho. Para fins de aplicação prática, ambos os modelos disponibilizam uma função, para predizer a nota ou a classe de nota que uma dada redação pertence. Para essa funções foram utilizados os modelos cujo melhor desempenho foi identificado durante as etapas de treinamento e validação.

Para validar a aplicação prática do desempenho dos modelos na aferição de notas, foi definida o Algoritmo 6.5 que randomicamente seleciona uma redação e seu respectivo texto motivador do conjunto de dados original e os submete a função **predict**. O valor da nota real e o valor predito são apresentados lado a lado.

A seguir, analisaremos o desempenho dessas funções no contexto de cada modelo.

Algoritmo 6.5 – Saída da função de predição para o modelo de classificação

```
import random

rand = random.randrange(0, len(df))

redacao = df.loc[rand, 'redacao']
texto_motivador = df.loc[rand, 'texto_motivador']
nota = df.loc[rand, 'nota_competencia_2']

print(f'redacao: {redacao[:100]}')
print(f'texto motivador: {texto_motivador[:100]}')
print(f'nota: {nota}')

pred = predict(model, tokenizer, redacao, texto_motivador)
print(f'pred: {pred}')
```

A função de predição para o modelo de classificação, Algoritmo 6.6, revela a capacidade do modelo em predizer corretamente a classe de notas que uma determinada redação pertence, nesse caso a segunda classe, que corresponde a nota 80. Esse desempenho corrobora os resultados identificados e discutidos na Seção 6.6 e endossa a viabilidade do modelo de classificação para a solução do problema proposto.

Algoritmo 6.6 – Saída da função de predição para o modelo de classificação

```
# código de preparação de parâmetros omitidos intencionalmente
pred = predict(model, tokenizer, redacao, texto_motivador)

# Saida:
> entrada redacao: no dia 7 de janeiro deste ano na redacao do jornal "charlie hebdo", em paris,
> entrada texto motivador: no dia 7 de janeiro deste ano, terroristas islamicos mataram a tiros
------------------------------
> classe real: 2
> nota real: 80
------------------------------
> classe predita: 2
> nota predita: 80
```

A mesma função porém agora adaptada para o modelo de regressão, apresenta a sua tentativa de predição da nota de uma dada redação e seu respectivo texto motivador. Percebe-se que a predição do modelo para uma redação cuja a nota é 120 foi consideravelmente distante, cerca de 117 pontos abaixo da nota real. Tal constatação, corrobora os resultados apresentados e discutidos na Seção 6.5 e confirma a inadequação do modelo de regressão para resolver o problema proposto.

Algoritmo 6.7 – Saída da função de predição para o modelo de regressão

```
# código e preparação de parâmetros omitidos intencionalmente
pred = predict(model, tokenizer, redacao, texto_motivador)

# Saida:
> entrada redacao: atualmente, o brasil com os programas e pesquisas sobre as dsts, intensificam as
> entrada texto motivador: as doencas sexualmente transmissiveis, ou dsts, sao aquelas doencas
------------------------------
> nota real: 120
------------------------------
> nota predita: 2.648085117340088
```

7 CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho utilizou o modelo de linguagem BERT*imbau*, modificação do BERT pré-treinado para a língua portuguesa do Brasil, para representar semanticamente redações e textos motivadores a partir de representações de *embeddings* contextualizados. Para atingir os objetivos do trabalho, dois modelos treinados em cima desses *embeddings* em Aprendizado Profundo foram criados: um de regressão e outro de classificação. Ao fim dos experimentos, o presente trabalho disponibiliza duas funções capazes de aferir ou classificar o desempenho que uma dada redação e seu respectivo texto motivador obteve dentro dos limites avaliativos da segunda competência da redação do ENEM. Desse modo, o trabalho atinge os objetivos a que se propôs.

A utilização de *embeddings* contextualizados, geradas pelo modelo BERT, estado da arte em PLN, para capturar e correlacionar a correspondência semântica entre dois textos, é inédita. Tal tarefa não apenas figura entre os esforços mais atuais na utilização de Inteligência Artificial para resolver problemas educacionais, como também preenche uma lacuna de aplicações de PLN especificamente adaptadas para a língua portuguesa do Brasil.

Os resultados obtidos revelam que a arquitetura de um modelo de classificação apresenta desempenho significativamente superior a de um modelo de regressão. Os experimentos conduzidos avaliaram o desempenho das duas arquiteturas e em todas as métricas o modelo de classificação se mostrou superior. O grau de confiança encontrado em nosso classificador é significativo e indica boa adequabilidade do modelo para lidar com a tarefa proposta. O modelo de regressão entretanto, se mostrou inviável sugerindo pouca adaptabilidade para resolver o problema proposto.

Entre as limitações encontradas no desenvolvimento dos trabalhos, destacam-se a baixa qualidade ortográfica em algumas redações do conjunto de dados. É possível que a estratégia de extração e limpeza utilizadas pelos autores do *dataset* quando da compilação das redações, corrompeu algumas palavras com caracteres especiais. Este é um fator relevante, pois palavras não compreendidas pelo modelo são ignoradas. Logo, alguns textos certamente tiveram suas representações comprometidas por essa razão. Outro fator limitante, foi a restrição do modelo BERT em processar documentos com no máximo 512 *tokens*. Por essa razão, apenas cerca de 40% do conjunto de dados do Essay-BR (MARINHO; ANCHIêTA; MOURA, 2021) foi utilizado.

Os potenciais trabalhos futuros identificados, são: *i)* utilizar técnicas eficientes para lidar com extração de *embeddings* de textos com mais de 512 (SUN et al., 2019) *tokens* enriquecendo o conjunto de treinamento; *ii)* aprimorar a etapa de limpeza de dados e extração de informações não-relevantes; *iii)* garantir a qualidade gramatical dos textos, uma vez que erros ortográficos implicarão em uma representação de *embeddings* empobrecida; *iv)* realizar o *fine-tuning* dos modelos utilizando diferentes combinações de hiper-parâmetros, especialmente no contexto de mais dados sendo processados; e ainda utilizar técnicas de avaliação como Máquinas de Vetores de Suporte, da sigla em inglês SVM ou ainda Regressão de Vetor de Suporte, da sigla também em inglês SVR, para definir uma baseline para os experimentos.

REFERÊNCIAS

BAEZA-YATES, R.; RIBEIRO-NETO, B. **Recuperação de Informação - 2ed: Conceitos e Tecnologia das Máquinas de Busca**. Bookman Editora, 2013. ISBN 9788582600498. Disponível em: https://books.google.com.br/books?id=YWk3AgAAQBAJ.

BERTUCCI, R. A. Propriedades linguísticas da redação do enem: uma análise computacional. **REVISTA DE ESTUDOS DA LINGUAGEM**, REVISTA DE ESTUDOS DA LINGUAGEM, v. 1, n. 1, p. 1–31, 2021.

BOJANOWSKI, P. et al. Enriching word vectors with subword information. **Transactions of the association for computational linguistics**, MIT Press, v. 5, p. 135–146, 2017.

BRAZ, O. O.; FILETO, R. Investigando coerência em postagens de um fórum de dúvidas em ambiente virtual de aprendizagem com o bert. In: SBC. **Anais do XXXII Simpósio Brasileiro de Informática na Educação**. [S.l.], 2021. p. 749–759.

CAVALCANTI, A. et al. In: **Anais do XXXI Simpósio Brasileiro de Informática na Educação**. Porto Alegre, RS, Brasil: SBC, 2020. p. 892–901. ISSN 0000-0000. Disponível em: https://sol.sbc.org.br/index.php/sbie/article/view/12845.

CAVALCANTI, A. P.; MELLO, F. L. d. R.; MIRANDA PERICLES BARBOSA CUNHA DE AND, F. L. G. d. F. Análise automática de feedback em ambientes de aprendizagem online. **IX Congresso Brasileiro de Informática na Educação**, Anais do XXXI Simpósio Brasileiro de Informática na Educação, v. 1, n. 1, 2020.

DEMERS, D.; COTTRELL, G. Non-linear dimensionality reduction. **Advances in neural information processing systems**, v. 5, 1992.

DEVLIN, J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. **arXiv preprint arXiv:1810.04805**, 2018.

DIAS, L. S. et al. Mineração de padrões morfo-semânticos em textos literários com o bert. Florianópolis, SC., 2022.

EVERT, S. **Outliers or Key Profiles? Understanding Distance Measures for Authorship Attribution**. 2016. Disponível em: https://dh2016.adho.org/abstracts/253.

FILHO, A. H. et al. An approach to evaluate adherence to the theme and the argumentative structure of essays. **Procedia Computer Science**, v. 126, p. 788–797, 2018. ISSN 1877-0509. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 22nd International Conference, KES-2018, Belgrade, Serbia. Disponível em: https://www.sciencedirect.com/science/article/pii/S1877050918312912.

FREIRE, J.; PINHEIRO, V.; FEITOSA, D. Flexsts: Um framework para similaridade semântica textual. **Linguamática**, v. 8, n. 2, p. 23–31, 2016.

GRAESSER, A. C.; MCNAMARA, D. S.; KULIKOWICH, J. M. Coh-metrix: Providing multilevel analyses of text characteristics. **Educational researcher**, Sage Publications Sage CA: Los Angeles, CA, v. 40, n. 5, p. 223–234, 2011.

HAGIWARA, M. **Real-World Natural Language Processing: Practical Applications with Deep Learning**. Manning, 2021. ISBN 9781617296420. Disponível em: https://books.google.com.br/books?id=Ok5NEAAAQBAJ.

HALDAR, R. et al. A multi-perspective architecture for semantic code search. **arXiv preprint arXiv:2005.06980**, 2020.

HARTMANN, N. et al. Portuguese word embeddings: Evaluating on word analogies and natural language tasks. **arXiv preprint arXiv:1708.06025**, 2017.

HUTCHINS, J. The history of machine translation in a nutshell. **Retrieved December**, v. 20, n. 2009, p. 1–1, 2005.

INEP. Censo da educaÇÃo superior 2020. **Ministério Da Educação**, Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira | Inep, v. 1, n. 1, 2020.

INEP. **A REDAÇÃO DO ENEM 2022 CARTILHA DO PARTICIPANTE**. 2022. Disponível em: https://download.inep.gov.br/download/enem/cartilha_do_participante_enem_2022.pdf.

JUNIOR, C. R. C. A.; SPALENZA, M. A.; OLIVEIRA, E. de. Proposta de um sistema de avaliacao automática de redações do enem utilizando tecnicas de aprendizagem de máquina e processamento de linguagem natural. **Communications of the ACM**, Anais do Computer on the Beach, mai 2017. Disponível em: https://doi.org/10.14210/cotb.v0n0.p474-483.

LEG/UFPR - Métodos de reamostragem. 2023. Accessed: 2023-11-12. Disponível em: http://cursos.leg.ufpr.br/ML4all/apoio/reamostragem.html#valida%C3%A7%C3%A3o_cruzada_holdout.

LING, W. et al. Two/too simple adaptations of word2vec for syntax problems. In: **Proceedings of the 2015 conference of the North American chapter of the association for computational linguistics: human language technologies**. [S.l.: s.n.], 2015. p. 1299–1304.

MARINHO, J.; ANCHIêTA, R.; MOURA, R. Essay-br: a brazilian corpus of essays. In: **Anais do III Dataset Showcase Workshop**. Online: Sociedade Brasileira de Computação, 2021. p. 53–64. Disponível em: https://sol.sbc.org.br/index.php/dsw/article/view/17414.

MARINHO, J. C.; ANCHIÊTA, R. T.; MOURA, R. S. Essay-br: a brazilian corpus of essays. **arXiv preprint arXiv:2105.09081**, 2021.

MCCORMICK, C. **The Inner Workings of BERT**. 1. ed. Chris McCormick, 2023. v. 1. (1, v. 1). Disponível em: https://www.chrismccormick.ai/bert-ebook.

MIKOLOV, T. et al. Efficient estimation of word representations in vector space. **arXiv preprint arXiv:1301.3781**, 2013.

OLIVEIRA, D. C. d.; POZZEBON, E.; SANTOS, T. N. d. Aplicação de técnicas de processamento de linguagem natural na automatização de correção de questões discursivas. **IX Congresso Brasileiro de Informática na Educação**, Anais do XXXI Simpósio Brasileiro de Informática na Educação, v. 1, n. 1, 2020.

OLIVEIRA, H. et al. Estimando coesão textual em redações no contexto do enem utilizando modelos de aprendizado de máquina. In: SBC. **Anais do XXXIII Simpósio Brasileiro de Informática na Educação**. [S.l.], 2022. p. 883–894.

PENNINGTON, J.; SOCHER, R.; MANNING, C. D. Glove: Global vectors for word representation. In: **Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)**. [S.l.: s.n.], 2014. p. 1532–1543.

PRAKOSO, D. W.; ABDI, A.; AMRIT, C. Short text similarity measurement methods: a review. **Soft Computing**, Springer, v. 25, n. 6, p. 4699–4723, 2021.

REIMERS, N.; GUREVYCH, I. Sentence-bert: Sentence embeddings using siamese bert-networks. **arXiv preprint arXiv:1908.10084**, 2019.

REINSEL JOHN GANTZ, J. R. D. The digitization of the world: From edge to core. In: . [s.n.], 2022. Disponível em: http://www-cs-faculty.stanford.edu/ uno/abcde.html.

SMILKOV, D. et al. Embedding projector: Interactive visualization and interpretation of embeddings. **arXiv preprint arXiv:1611.05469**, 2016.

SORATO, D.; GOULARTE, F. B.; FILETO, R. Short semantic patterns: A linguistic pattern mining approach for content analysis applied to hate speech. **International Journal on Artificial Intelligence Tools**, World Scientific, v. 29, n. 02, p. 2040002, 2020.

SOUZA, F.; NOGUEIRA, R.; LOTUFO, R. Bertimbau: pretrained bert models for brazilian portuguese. In: SPRINGER. **Brazilian conference on intelligent systems**. [S.l.], 2020. p. 403–417.

SUN, C. et al. How to fine-tune bert for text classification? In: SPRINGER. **Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings 18**. [S.l.], 2019. p. 194–206.

VASWANI, A. et al. Attention is all you need. **Advances in neural information processing systems**, v. 30, 2017.

VERLEYSEN, M.; LEE, J. A. Nonlinear dimensionality reduction for visualization. In: SPRINGER. **International Conference on Neural Information Processing**. [S.l.], 2013. p. 617–622.

WANG, W.; YAN, M.; WU, C. Multi-granularity hierarchical attention fusion networks for reading comprehension and question answering. **arXiv preprint arXiv:1811.11934**, 2018.

WIRTH, R.; HIPP, J. Crisp-dm: Towards a standard process model for data mining. In: MANCHESTER. **Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining**. [S.l.], 2000. v. 1, p. 29–39.

APÊNDICE A – CÓDIGO FONTE

A.1 CLASSIFICADOR

```
import os
import pandas as pd
import tensorflow as tf
import numpy as np

from tqdm import tqdm

tqdm.pandas()

from google.colab import drive

drive.mount("/content/drive")

DIRETORIO_COHEBERT = "data"

DIRETORIO_LOCAL = "/content/" + DIRETORIO_COHEBERT + "/"

# Diretório no google drive com os arquivos pré-processados
DIRETORIO_DRIVE = (
        "/content/drive/MyDrive/Colab Notebooks/BERT_classificador/bert_base/"
        + DIRETORIO_COHEBERT
)

NOME_ARQUIVO_INPUT = "bert_base_redacoes_cos_sim_texto_motivador_categoria.csv"

##############################################################################################

! pip install sklearn
! pip install -U sentence-transformers
! python -m spacy download pt_core_news_sm
! pip install tensorflow h5py
! pip install --upgrade tensorflow h5py

import nltk
nltk.download('punkt')

##############################################################################################

import tensorflow as tf

# Get the GPU device name.
device_name = tf.test.gpu_device_name()

# The device name should look like the following:
if device_name == "/device:GPU:0":
    print("Found GPU at: {}".format(device_name))
else:
    raise SystemError("GPU device not found")

##############################################################################################

df = pd.read_csv(f"{DIRETORIO_DRIVE}/{NOME_ARQUIVO_INPUT}")
df.info()
df = df[["redacao", "texto_motivador", "nota_competencia_2"]]
```

```
55  df.sample(5)
56
57
58  ##############################################################################################
59
60  def map_score_to_class(score):
61      return score // 40
62
63
64  df["nota_competencia_2"] = df.nota_competencia_2.apply(map_score_to_class)
65
66  ##############################################################################################
67
68  # Extrai redacoes, textos motivadores e notas para treinamento
69
70  redacoes = df.redacao.values
71  textos_motivadores = df.texto_motivador.values
72  notas = df.nota_competencia_2.values
73
74  ##############################################################################################
75
76  from transformers import AutoTokenizer  # Or BertTokenizer
77  from transformers import (
78      AutoModelForSequenceClassification,
79  )  # Or BertForPreTraining for loading pretraining heads
80
81  model_name = "neuralmind/bert-base-portuguese-cased"
82
83  tokenizer = AutoTokenizer.from_pretrained(model_name, do_lower_case=False)
84
85  # num_labels corresponde ao intervalo de notas possíveis na categoria 2 i.e. [0, 40, 80, 120, 160, 200], ou
86  bertimbau = AutoModelForSequenceClassification.from_pretrained(
87      model_name, num_labels=6
88  ).cuda()
89
90  ##############################################################################################
91
92  # Tokenize todas as redações e realiza o mapeamento de tokens para os seus respectivos input_ids
93
94  r_input_ids = []
95  r_attention_masks = []
96
97  for redacao in redacoes:
98      # `encode_plus` will:
99      #   (1) Tokeniza redação.
100     #   (2) Add `[CLS]` token no inicio do texto.
101     #   (3) Add `[SEP]` token no fim do texto.
102     #   (4) Mapeamento de tokens para seus IDs.
103     #   (5) Padding `max_length`
104     #   (6) Cria attention masks para os tokens do tipo [PAD].
105     r_encoded_dict = tokenizer.encode_plus(
106         redacao,  # Redação.
107         add_special_tokens=True,  # Add '[CLS]' and '[SEP]'
108         max_length=512,  # Pad redação.
109         padding="max_length",
110         return_attention_mask=True,  # Constrói attn. masks.
111         return_tensors="pt",  # Retorna pytorch tensors.
112     )
113
114     r_input_ids.append(r_encoded_dict["input_ids"])
```

```
115     r_attention_masks.append(r_encoded_dict["attention_mask"])
116 
117 ###############################################################################################
118 
119 # Tokenize todas as redações e realiza o mapeamento de tokens para os seus respectivos input_ids
120 
121 tm_input_ids = []
122 tm_attention_masks = []
123 
124 for tm in textos_motivadores:
125     # `encode_plus` will:
126     #   (1) Tokeniza texto motivador.
127     #   (2) Add `[CLS]` token no inicio do texto.
128     #   (3) Add `[SEP]` token no fim do texto.
129     #   (4) Mapeamento de tokens para seus IDs.
130     #   (5) Padding `max_length`
131     #   (6) Cria attention masks para os tokens do tipo [PAD].
132     tm_encoded_dict = tokenizer.encode_plus(
133         tm,  # Texto motivador
134         add_special_tokens=True,  # Add '[CLS]' and '[SEP]'
135         max_length=512,  # Pad redação.
136         padding="max_length",
137         return_attention_mask=True,  # Constrói attn. masks.
138         return_tensors="pt",  # Retorna pytorch tensors.
139     )
140 
141     tm_input_ids.append(tm_encoded_dict["input_ids"])
142     tm_attention_masks.append(tm_encoded_dict["attention_mask"])
143     notas = torch.tensor(notas)
144 
145 ###############################################################################################
146 
147 tm_input_ids = torch.cat(tm_input_ids, dim=0)
148 tm_attention_masks = torch.cat(tm_attention_masks, dim=0)
149 
150 r_input_ids = torch.cat(r_input_ids, dim=0)
151 r_attention_masks = torch.cat(r_attention_masks, dim=0)
152 
153 ###############################################################################################
154 
155 notas = notas.to(device)
156 
157 r_input_ids = r_input_ids.to(device)
158 tm_input_ids = tm_input_ids.to(device)
159 
160 r_attention_masks = r_attention_masks.to(device)
161 tm_attention_masks = tm_attention_masks.to(device)
162 
163 ###############################################################################################
164 
165 notas = notas.float()
166 
167 r_input_ids = r_input_ids.long()
168 tm_input_ids = tm_input_ids.long()
169 
170 r_attention_masks = r_attention_masks.float()
171 tm_attention_masks = tm_attention_masks.float()
172 
173 notas = notas.long()
174 
```

```
175 ##########################################################################################
176
177 from torch.utils.data import Dataset, DataLoader
178
179
180 class PreTokenizedDataset(Dataset):
181     def __init__(
182             self,
183             r_input_ids_list,
184             r_attention_masks_list,
185             tm_input_ids_list,
186             tm_attention_masks_list,
187             notas,
188     ):
189         self.r_input_ids_list = r_input_ids_list
190         self.r_attention_masks_list = r_attention_masks_list
191         self.tm_input_ids_list = tm_input_ids_list
192         self.tm_attention_masks_list = tm_attention_masks_list
193         self.notas = notas
194
195     def __len__(self):
196         return len(self.notas)
197
198     def __getitem__(self, idx):
199         return {
200             "r_input_ids": self.r_input_ids_list[idx],
201             "r_attention_mask": self.r_attention_masks_list[idx],
202             "tm_input_ids": self.tm_input_ids_list[idx],
203             "tm_attention_mask": self.tm_attention_masks_list[idx],
204             "nota_competencia_2": self.notas[idx],
205         }
206
207
208 ##########################################################################################
209
210 BATCH_SIZE = 8
211
212 complete_dataset = PreTokenizedDataset(
213     r_input_ids, r_attention_masks, tm_input_ids, tm_attention_masks, notas
214 )
215 complete_dataloader = DataLoader(complete_dataset, batch_size=BATCH_SIZE, shuffle=True)
216
217 ##########################################################################################
218
219 import torch
220 from torch import nn
221
222 NUM_LABELS = 6  # (0, 40, 80, 120, 160, 200)
223 INPUT_FEATURE_SIZE = 768
224
225
226 class BertForEssayScoring(nn.Module):
227     def __init__(self):
228         super(BertForEssayScoring, self).__init__()
229         self.bert_encoder = bertimbau
230         self.classification_head = nn.Sequential(
231             nn.Linear(
232                 2 * self.bert_encoder.config.hidden_size, 512
233             ),  # 2 * para a concatenação
234             nn.ReLU(),
```

```
235             nn.Linear(512, NUM_LABELS),
236         )
237
238     def forward(self, r_input_ids, r_attention_masks, tm_input_ids, tm_attention_masks):
239         r_outputs = self.bert_encoder(
240             r_input_ids, r_attention_masks, output_hidden_states=True
241         )
242         r_class_label_output = r_outputs.hidden_states[-1][:, 0, :]
243
244         tm_outputs = self.bert_encoder(
245             tm_input_ids, tm_attention_masks, output_hidden_states=True
246         )
247         tm_class_label_output = tm_outputs.hidden_states[-1][:, 0, :]
248
249         concatenated_output = torch.cat(
250             (r_class_label_output, tm_class_label_output), dim=-1
251         )
252
253         logits = self.classification_head(concatenated_output)
254         return logits
255
256
257 model = BertForEssayScoring()
258 model.to(device)
259
260 ###############################################################################################
261
262 # Get all of the model's parameters as a list of tuples.
263 params = list(bertimbau.named_parameters())
264
265 print("The BERT model has {:} different named parameters.\n".format(len(params)))
266
267 print("==== Embedding Layer ====\n")
268
269 for p in params[0:5]:
270     print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
271
272 print("\n==== First Transformer ====\n")
273
274 for p in params[5:21]:
275     print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
276
277 print("\n==== Output Layer ====\n")
278
279 for p in params[-4:]:
280     print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
281
282
283 ###############################################################################################
284
285 def eval_model(model, loss_function, test_dataloader, device):
286     model = model.eval()
287
288     total_eval_accuracy = []
289     total_eval_loss = 0
290     total_eval_f1 = 0
291
292     all_preds = []
293     all_labels = []
294
```

```
295     with torch.no_grad():
296         for batch in test_dataloader:
297             batch_r_input_ids = batch["r_input_ids"].to(device)
298             batch_r_attention_masks = batch["r_attention_mask"].to(device)
299             batch_tm_input_ids = batch["tm_input_ids"].to(device)
300             batch_tm_attention_masks = batch["tm_attention_mask"].to(device)
301             batch_labels = batch["nota_competencia_2"].to(device)
302 
303             logits = model(
304                 r_input_ids=batch_r_input_ids,
305                 r_attention_masks=batch_r_attention_masks,
306                 tm_input_ids=batch_tm_input_ids,
307                 tm_attention_masks=batch_tm_attention_masks,
308             )
309 
310             loss = loss_function(logits, batch_labels)
311             total_eval_loss += loss.item()
312 
313             preds = torch.argmax(logits, dim=1).flatten()
314             all_preds.extend(preds.cpu().numpy())
315             all_labels.extend(batch_labels.cpu().numpy())
316 
317     # Calculate the average loss over all batches
318     avg_test_loss = total_eval_loss / len(test_dataloader)
319 
320     # Calculate the accuracy and F1 score using the aggregated predictions and labels
321     accuracy = accuracy_score(all_labels, all_preds) * 100
322     f1 = f1_score(all_labels, all_preds, average="weighted")
323 
324     return avg_test_loss, accuracy, f1
325 
326 
327 ##########################################################################################
328 
329 import datetime
330 
331 
332 def format_time(elapsed):
333     """
334     Takes time in seconds and returns a string hh:mm:ss
335     """
336 
337     elapsed_rounded = int(round(elapsed))
338 
339     return str(datetime.timedelta(seconds=elapsed_rounded))
340 
341 
342 ##########################################################################################
343 
344 import time
345 import torch
346 from sklearn.metrics import accuracy_score, f1_score
347 from sklearn.model_selection import KFold
348 import numpy as np
349 import copy
350 import os
351 
352 # Define the number of folds for cross-validation
353 NUM_FOLDS = (
354     10  # You can adjust this number based on your dataset size and requirements.
```

```
)

overall_train_loss_values = []
overall_val_loss_values = []
overall_train_accuracy_values = []
overall_val_accuracy_values = []
overall_train_f1_values = []
overall_val_f1_values = []
all_fold_time = 0


def train_with_cross_validation(
        model,
        optimizer,
        scheduler,
        loss_function,
        epochs,
        dataset,
        device,
        save_path,
        clip_value=2,
        logging_interval=40,
):
    # Initialize the K-fold cross-validator
    kf = KFold(n_splits=NUM_FOLDS, shuffle=True, random_state=42)

    loss_values = []
    val_loss_values = []
    accuracy_train_values = []
    accuracy_val_values = []
    f1_train_values = []
    f1_val_values = []
    global all_fold_time

    best_model_wts = copy.deepcopy(model.state_dict())
    best_val_f1 = float("inf")  # Initialize with a high number

    for fold, (train_indices, val_indices) in enumerate(kf.split(dataset)):
        fold_t0 = time.time()
        print(
            f"\n==================== Fold {fold + 1} / {NUM_FOLDS} ======================"
        )

        train_sampler = torch.utils.data.SubsetRandomSampler(train_indices)
        val_sampler = torch.utils.data.SubsetRandomSampler(val_indices)

        train_dataloader = DataLoader(
            dataset, batch_size=BATCH_SIZE, sampler=train_sampler
        )
        val_dataloader = DataLoader(dataset, batch_size=BATCH_SIZE, sampler=val_sampler)

        assert max(train_indices) < len(dataset) and max(val_indices) < len(dataset)

        # Save current fold datasets
        folds_save_dir = os.path.join(save_path, f"00_LR_{str(LR).replace('-', '_')}")
        if not os.path.exists(folds_save_dir):
            os.makedirs(folds_save_dir)

        torch.save(
            train_indices,
```

```
415                 os.path.join(folds_save_dir, f"train_indices_fold_{fold + 1}.pt"),
416             )
417             torch.save(
418                 val_indices, os.path.join(folds_save_dir, f"val_indices_fold_{fold + 1}.pt")
419             )
420 
421             for epoch in range(epochs):
422                 epoch_start_time = time.time()
423                 fold_time = time.time()
424 
425                 # Reset as losses for the current epoch
426                 total_loss = 0
427                 total_accuracy = 0
428                 total_f1 = 0
429 
430                 # Put the model into training mode.
431                 model.train()
432 
433                 for step, batch in enumerate(train_dataloader):
434                     # Unpack this training batch from our dataloader and copy each tensor to the GPU.
435                     batch_r_input_ids = batch["r_input_ids"].to(device)
436                     batch_r_attention_masks = batch["r_attention_mask"].to(device)
437                     batch_tm_input_ids = batch["tm_input_ids"].to(device)
438                     batch_tm_attention_masks = batch["tm_attention_mask"].to(device)
439                     batch_labels = batch["nota_competencia_2"].to(device)
440 
441                     model.zero_grad()  # Clear any previously calculated gradients before performing a backward
442                     optimizer.zero_grad()
443 
444                     # Perform a forward pass. This will return logits.
445                     logits = model(
446                         r_input_ids=batch_r_input_ids,
447                         r_attention_masks=batch_r_attention_masks,
448                         tm_input_ids=batch_tm_input_ids,
449                         tm_attention_masks=batch_tm_attention_masks,
450                     )
451 
452                     # Calculate loss and accumulate the loss value.
453                     loss = loss_function(logits, batch_labels)
454                     total_loss += loss.item()
455 
456                     # Perform a backward pass to calculate gradients.
457                     loss.backward()
458 
459                     # Clip the norm of the gradients to prevent the "exploding gradients" problem.
460                     torch.nn.utils.clip_grad_norm_(model.parameters(), clip_value)
461 
462                     # Update parameters and the learning rate.
463                     optimizer.step()
464                     scheduler.step()
465 
466                     # Convert logits to predicted class.
467                     preds = torch.argmax(logits, dim=1).flatten().to(torch.int64)
468 
469                     # Calculate the accuracy and f1 score.
470                     accuracy = (
471                             accuracy_score(batch_labels.cpu().numpy(), preds.cpu().numpy())
472                             * 100
473                     )
474                     f1 = f1_score(
```

```
                    batch_labels.cpu().numpy(), preds.cpu().numpy(), average="weighted"
                )

                total_accuracy += accuracy
                total_f1 += f1

            # Report the final accuracy, f1, and loss for this training run.
            avg_train_accuracy = total_accuracy / len(train_dataloader)
            avg_train_f1 = total_f1 / len(train_dataloader)
            avg_train_loss = total_loss / len(train_dataloader)

            accuracy_train_values.append(avg_train_accuracy)
            f1_train_values.append(avg_train_f1)
            loss_values.append(avg_train_loss)

            # #========================================================
            # #                    Validation
            # #========================================================

            # Put the model in evaluation mode
            model.eval()

            # Tracking variables
            total_eval_accuracy = 0
            total_eval_loss = 0
            total_eval_f1 = 0

            # Evaluate data for one epoch.
            for batch in val_dataloader:
                # Unpack this validation batch from our dataloader and copy each tensor to the GPU.
                batch_r_input_ids = batch["r_input_ids"].to(device)
                batch_r_attention_masks = batch["r_attention_mask"].to(device)
                batch_tm_input_ids = batch["tm_input_ids"].to(device)
                batch_tm_attention_masks = batch["tm_attention_mask"].to(device)
                batch_labels = batch["nota_competencia_2"].to(device)

                with torch.no_grad():
                    # Perform a forward pass. This will return logits.
                    logits = model(
                        r_input_ids=batch_r_input_ids,
                        r_attention_masks=batch_r_attention_masks,
                        tm_input_ids=batch_tm_input_ids,
                        tm_attention_masks=batch_tm_attention_masks,
                    )

                    # Calculate loss and accumulate the loss value.
                    loss = loss_function(logits, batch_labels)
                    total_eval_loss += loss.item()

                    # Convert logits to predicted class.
                    preds = torch.argmax(logits, dim=1).flatten().to(torch.int64)

                    # Calculate the accuracy and f1 score.
                    accuracy = (
                            accuracy_score(batch_labels.cpu().numpy(), preds.cpu().numpy())
                            * 100
                    )
                    f1 = f1_score(
                        batch_labels.cpu().numpy(),
                        preds.cpu().numpy(),
```

```
                        average="weighted",
                    )

                    total_eval_accuracy += accuracy
                    total_eval_f1 += f1

            # Report the final accuracy, f1, and loss for this validation run.
            avg_val_accuracy = total_eval_accuracy / len(val_dataloader)
            avg_val_f1 = total_eval_f1 / len(val_dataloader)
            avg_val_loss = total_eval_loss / len(val_dataloader)

            if avg_val_f1 < best_val_f1:
                best_model_wts = copy.deepcopy(model.state_dict())
                print(
                    f"New best model found at fold {fold + 1} with F1: {avg_val_f1}\n"
                )

            accuracy_val_values.append(avg_val_accuracy)
            f1_val_values.append(avg_val_f1)
            val_loss_values.append(avg_val_loss)

            epoch_end_time = time.time()
            epoch_time = epoch_end_time - epoch_start_time
            print(f"Epoch {epoch + 1}/{epochs} took: {format_time(epoch_time)}")

        # Measure how long this fold took.
        fold_time = format_time(time.time() - fold_t0)
        all_fold_time += time.time() - fold_t0

        print("================================")
        print("\nTraining performance")
        print(f"Accuracy: {accuracy_train_values[-1]:.2f}%")
        print(f"F1 Score: {f1_train_values[-1]:.2f}")
        print(f"Loss: {loss_values[-1]:.2f}")
        print("================================")
        print("\nValidation performance")
        print(f"Accuracy: {accuracy_val_values[-1]:.2f}")
        print(f"F1 Score: {f1_val_values[-1]:.2f}")
        print(f"Loss: {val_loss_values[-1]:.2f}")
        print(f"\nFold {fold + 1} / {NUM_FOLDS} took: {fold_time}")

        overall_train_loss_values.append(loss_values[-1])
        overall_train_accuracy_values.append(accuracy_train_values[-1])
        overall_train_f1_values.append(f1_train_values[-1])
        overall_val_loss_values.append(val_loss_values[-1])
        overall_val_accuracy_values.append(accuracy_val_values[-1])
        overall_val_f1_values.append(f1_val_values[-1])

    print(f"overall_train_loss_values: {overall_train_loss_values}")
    print(f"overall_val_loss_values: {overall_val_loss_values}")
    print(f"overall_train_accuracy_values: {overall_train_accuracy_values}")
    print(f"overall_val_accuracy_values: {overall_val_accuracy_values}")
    print(f"overall_train_f1_values: {overall_train_f1_values}")
    print(f"overall_val_f1_values: {overall_val_f1_values}")

    # Calculate overall average metrics
    avg_train_loss = np.mean(overall_train_loss_values)
    avg_val_loss = np.mean(overall_val_loss_values)
    avg_train_accuracy = np.mean(overall_train_accuracy_values)
    avg_val_accuracy = np.mean(overall_val_accuracy_values)
```

```
595         avg_train_f1 = np.mean(overall_train_f1_values)
596         avg_val_f1 = np.mean(overall_val_f1_values)
597 
598         print(
599             f"\n================================ Training complete! ================================"
600         )
601         print("Overall summary:")
602         print(f"Average Training Loss: {avg_train_loss:.2f}")
603         print(f"Average Validation Loss: {avg_val_loss:.2f}")
604         print(f"Average Training Accuracy: {avg_train_accuracy:.2f}")
605         print(f"Average Validation Accuracy: {avg_val_accuracy:.2f}")
606         print(f"Average Training F1 Score: {avg_train_f1:.2f}")
607         print(f"Average Validation F1 Score: {avg_val_f1:.2f}")
608         print(f"All folds took: {format_time(all_fold_time)}")
609         print("Training complete!")
610 
611         # Load best model
612         model.load_state_dict(best_model_wts)
613         print(f"Best model was from fold with F1 score: {best_val_f1:.2f}")
614 
615         return (
616             model,
617             overall_train_loss_values,
618             overall_val_loss_values,
619             overall_train_accuracy_values,
620             overall_val_accuracy_values,
621             overall_train_f1_values,
622             overall_val_f1_values,
623         )
624 
625 
626 ################################################################################################
627 
628 from transformers import get_linear_schedule_with_warmup
629 
630 LR = 2e-5  # recommended lrs -> 5e-5, 3e-5, 2e-5
631 optimizer = AdamW(bertimbau.parameters(), lr=LR, eps=1e-8)
632 
633 EPOCHS = 3  # O artigo do BERT recomenda de 2-4 epocas. Checar se 4 épocas não está causando over-fitting.
634 
635 # Total number of training steps is [number of batches] x [number of epochs].
636 # (Note that this is not the same as the number of training samples).
637 total_steps = len(complete_dataloader) * EPOCHS
638 
639 # Create the learning rate scheduler.
640 scheduler = get_linear_schedule_with_warmup(
641     optimizer,
642     num_warmup_steps=0,  # Default value in run_glue.py
643     num_training_steps=total_steps,
644 )
645 
646 loss_function = torch.nn.CrossEntropyLoss()
647 
648 ################################################################################################
649 
650 save_dir = f"/content/drive/MyDrive/Colab Notebooks/BERT_classificador/bert_base/treinamentos/k-folds"
651 if not os.path.exists(save_dir):
652     os.makedirs(save_dir)
653 
654 ################################################################################################
```

```
655
656 (
657     model,
658     avg_train_loss,
659     avg_val_loss,
660     avg_train_accuracy,
661     avg_val_accuracy,
662     avg_train_f1,
663     avg_val_f1,
664 ) = train_with_cross_validation(
665     model,
666     optimizer,
667     scheduler,
668     loss_function,
669     EPOCHS,
670     complete_dataset,
671     device,
672     save_dir,
673     clip_value=2,
674     logging_interval=40,
675 )
676
677 ##########################################################################################
678
679 import matplotlib.pyplot as plt
680
681
682 def plot_performance(
683         train_loss, val_loss, train_accuracy, val_accuracy, train_f1, val_f1
684 ):
685     epochs = range(1, len(train_loss) + 1)
686
687     plt.figure(figsize=(12, 4))
688
689     # Plot training and validation loss
690     plt.subplot(1, 3, 1)
691     plt.plot(epochs, train_loss, "b-", label="Training Loss")
692     plt.plot(epochs, val_loss, "r-", label="Validation Loss")
693     plt.title("Training & Validation Loss")
694     plt.xlabel("Epochs")
695     plt.ylabel("Loss")
696     plt.legend()
697
698     # Plot training and validation accuracy
699     plt.subplot(1, 3, 2)
700     plt.plot(epochs, train_accuracy, "b-", label="Training Accuracy")
701     plt.plot(epochs, val_accuracy, "r-", label="Validation Accuracy")
702     plt.title("Training & Validation Accuracy")
703     plt.xlabel("Epochs")
704     plt.ylabel("Accuracy")
705     plt.legend()
706
707     # Plot training and validation F1 Score
708     plt.subplot(1, 3, 3)
709     plt.plot(epochs, train_f1, "b-", label="Training F1 Score")
710     plt.plot(epochs, val_f1, "r-", label="Validation F1 Score")
711     plt.title("Training & Validation F1 Score")
712     plt.xlabel("Epochs")
713     plt.ylabel("F1 Score")
714     plt.legend()
```

```
715
716         plt.tight_layout()
717         plt.show()
718
719
720     ##########################################################################################
721
722     def predict(model, tokenizer, essay, auxiliary_text):
723         model.to(device)
724
725         essay_tokens = tokenizer(
726             essay, return_tensors="pt", truncation=True, padding=True, max_length=500
727         ).to(device)
728         auxiliary_tokens = tokenizer(
729             auxiliary_text,
730             return_tensors="pt",
731             truncation=True,
732             padding=True,
733             max_length=500,
734         ).to(device)
735
736         # Ensure model is in evaluation mode
737         model.eval()
738
739         # No gradient computation needed during prediction
740         with torch.no_grad():
741             logits = model(
742                 essay_tokens["input_ids"],
743                 essay_tokens["attention_mask"],
744                 auxiliary_tokens["input_ids"],
745                 auxiliary_tokens["attention_mask"],
746             )
747             prediction = torch.argmax(logits, dim=1)
748         return prediction.item()
749
750
751     ##########################################################################################
752
753     import random
754
755     rand = random.randrange(0, len(df))
756
757     redacao = df.loc[rand, "redacao"]
758     texto_motivador = df.loc[rand, "texto_motivador"]
759     nota = df.loc[rand, "nota_competencia_2"]
760
761     print(f"redacao: {redacao[:100]}")
762     print(f"texto motivador: {texto_motivador[:100]}")
763     print(f"classe: {nota // 40}")
764     print(f"nota: {nota}")
765
766     print("-" * 30)
767
768     pred = predict(model, tokenizer, redacao, texto_motivador)
769     print(f"classe pred: {pred}")
770     print(f"nota: {pred * 40}")
771
772     ##########################################################################################
773
774     save_dir = f"/content/drive/MyDrive/Colab Notebooks/BERT_classificador/bert_base/treinamentos/k-folds"
```

```
775 if not os.path.exists(save_dir):
776     os.makedirs(save_dir)
777
778 ##############################################################################################
779
780 torch.save(
781     model.state_dict(),
782     f"{save_dir}/00_LR_{str(LR).replace('-', '_')}/bert_base_cls_tokens_lr_{str(LR)}_{EPOCHS}_epochs.json",
783 )
784
785 ##############################################################################################
786
787 bert_classifier = BertForEssayScoring()
788 bert_classifier.load_state_dict(
789     torch.load(
790         f"{save_dir}/00_LR_{str(LR).replace('-', '_')}/bert_base_cls_tokens_lr_{str(LR)}_{EPOCHS}_epochs.js
791     )
792 )
793 bert_classifier.eval()
794
795 ##############################################################################################
```

A.2 REGRESSOR

```
1  import os
2  import pandas as pd
3  import tensorflow as tf
4
5  from tqdm import tqdm
6
7  tqdm.pandas()
8
9  from google.colab import drive
10
11 drive.mount('/content/drive')
12
13 DIRETORIO_COHEBERT = "data"
14
15 DIRETORIO_LOCAL = "/content/" + DIRETORIO_COHEBERT + "/"
16
17 # Diretório no google drive com os arquivos pré-processados
18 DIRETORIO_DRIVE = (
19     "/content/drive/MyDrive/Colab Notebooks/BERT_regressor/BERT_base_experimentos/", DIRETORIO_LOCAL
20 )
21
22 NOME_ARQUIVO_INPUT = 'bert_base_redacoes_cos_sim_texto_motivador_categoria.csv'
23
24 import tensorflow as tf
25
26 # Get the GPU device name.
27 device_name = tf.test.gpu_device_name()
28
29 # The device name should look like the following:
30 if device_name == '/device:GPU:0':
31     print('Found GPU at: {}'.format(device_name))
32 else:
33     raise SystemError('GPU device not found')
34
35 import torch
```

```
36
37  torch.cuda.empty_cache()
38
39  # If there's a GPU available...
40  if torch.cuda.is_available():
41
42      # Tell PyTorch to use the GPU.
43      device = torch.device("cuda")
44
45      print('There are %d GPU(s) available.' % torch.cuda.device_count())
46
47      print('We will use the GPU:', torch.cuda.get_device_name(0))
48
49  # If not...
50  else:
51      print('No GPU available, using the CPU instead.')
52      device = torch.device("cpu")
53
54  ##############################################################################################
55
56  ! pip
57  install
58  sklearn
59  ! pip
60  install -U
61  sentence-transformers
62  ! python -m
63  spacy
64  download
65  pt_core_news_sm
66  ! pip
67  install
68  tensorflow
69  h5py
70  ! pip
71  install --upgrade
72  tensorflow
73  h5py
74
75  import nltk
76
77  nltk.download('punkt')
78
79  ##############################################################################################
80
81  df = pd.read_csv(f"{DIRETORIO_DRIVE}/{NOME_ARQUIVO_INPUT}")
82  df.info()
83  df.sample(5)
84
85  ##############################################################################################
86
87  # Extrai redacoes, textos motivadores e notas para treinamento
88
89  redacoes = df.redacao.values
90  textos_motivadores = df.texto_motivador.values
91  notas = df.nota_competencia_2.values
92
93  ##############################################################################################
94  # Or BertTokenizer
95  from transformers import AutoTokenizer
```

```
# Or BertForPreTraining for loading pretraining heads
from transformers import AutoModelForSequenceClassification

model_name = 'neuralmind/bert-base-portuguese-cased'

tokenizer = AutoTokenizer.from_pretrained(model_name, do_lower_case=False)
bertimbau = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=1).cuda()

################################################################################################

# Tokenize todas as redações e realiza o mapeamento de tokens para os seus respectivos input_ids

r_input_ids = []
r_attention_masks = []

for redacao in redacoes:
    # `encode_plus` will:
    #   (1) Tokeniza redação.
    #   (2) Add `[CLS]` token no inicio do texto.
    #   (3) Add `[SEP]` token no fim do texto.
    #   (4) Mapeamento de tokens para seus IDs.
    #   (5) Padding `max_length`
    #   (6) Cria attention masks para os tokens do tipo [PAD].
    r_encoded_dict = tokenizer.encode_plus(
        redacao,  # Redação.
        add_special_tokens=True,  # Add '[CLS]' and '[SEP]'
        max_length=512,  # Pad redação.
        padding='max_length',
        return_attention_mask=True,  # Constrói attn. masks.
        return_tensors='pt',  # Retorna pytorch tensors.
    )

    r_input_ids.append(r_encoded_dict['input_ids'])
    r_attention_masks.append(r_encoded_dict['attention_mask'])

################################################################################################

# Tokenize todas as redações e realiza o mapeamento de tokens para os seus respectivos input_ids

tm_input_ids = []
tm_attention_masks = []

for tm in textos_motivadores:
    # `encode_plus` will:
    #   (1) Tokeniza texto motivador.
    #   (2) Add `[CLS]` token no inicio do texto.
    #   (3) Add `[SEP]` token no fim do texto.
    #   (4) Mapeamento de tokens para seus IDs.
    #   (5) Padding `max_length`
    #   (6) Cria attention masks para os tokens do tipo [PAD].
    tm_encoded_dict = tokenizer.encode_plus(
        tm,  # Texto motivador
        add_special_tokens=True,  # Add '[CLS]' and '[SEP]'
        max_length=512,  # Pad redação.
        padding='max_length',
        return_attention_mask=True,  # Constrói attn. masks.
        return_tensors='pt',  # Retorna pytorch tensors.
    )

    tm_input_ids.append(tm_encoded_dict['input_ids'])
```

```
156         tm_attention_masks.append(tm_encoded_dict['attention_mask'])
157
158     ################################################################################################
159
160     from torch.utils.data import TensorDataset, DataLoader, random_split
161
162     # Transoforma lista de notas em uma torch
163     notas = torch.tensor(notas)
164
165     # Concatena lista de torchs
166
167     r_input_ids = torch.cat(r_input_ids, dim=0)
168     tm_input_ids = torch.cat(tm_input_ids, dim=0)
169     r_attention_masks = torch.cat(r_attention_masks, dim=0)
170     tm_attention_masks = torch.cat(tm_attention_masks, dim=0)
171
172     ################################################################################################
173
174     # Adiciona torches para rodarem na GPU
175
176     notas = notas.to(device)
177     r_input_ids = r_input_ids.to(device)
178     tm_input_ids = tm_input_ids.to(device)
179     r_attention_masks = r_attention_masks.to(device)
180     tm_attention_masks = tm_attention_masks.to(device)
181
182     ################################################################################################
183
184     # Set types
185
186     notas = notas.float()
187     r_input_ids = r_input_ids.long()
188     tm_input_ids = tm_input_ids.long()
189     r_attention_masks = r_attention_masks.float()
190     tm_attention_masks = tm_attention_masks.float()
191
192     ################################################################################################
193
194     from torch.utils.data import Dataset, DataLoader
195
196
197     class PreTokenizedDataset(Dataset):
198         def __init__(self, r_input_ids_list, r_attention_masks_list,
199                      tm_input_ids_list, tm_attention_masks_list, notas):
200             self.r_input_ids_list = r_input_ids_list
201             self.r_attention_masks_list = r_attention_masks_list
202             self.tm_input_ids_list = tm_input_ids_list
203             self.tm_attention_masks_list = tm_attention_masks_list
204             self.notas = notas
205
206         def __len__(self):
207             return len(self.notas)
208
209         def __getitem__(self, idx):
210             return {
211                 'r_input_ids': self.r_input_ids_list[idx],
212                 'r_attention_mask': self.r_attention_masks_list[idx],
213                 'tm_input_ids': self.tm_input_ids_list[idx],
214                 'tm_attention_mask': self.tm_attention_masks_list[idx],
215                 'nota_competencia_2': self.notas[idx]
```

```
216         }
217
218
219 ##########################################################################################
220
221 from torch.utils.data import RandomSampler, SequentialSampler
222
223 """
224   32 é o tamanho recomendado pelo artigo do Devlin para fine-tunning,
225   porém batches maiores que 8 tem consumido mais RAM do que o colab (até mesmo o PRO oferece) que é
    de 15-20 GB
226 """
227 BATCH_SIZE = 8
228
229 complete_dataset = PreTokenizedDataset(r_input_ids, r_attention_masks,
230                                        tm_input_ids, tm_attention_masks, notas)
231 complete_dataloader = DataLoader(complete_dataset,
232                                  sampler=RandomSampler(complete_dataset), batch_size=BATCH_SIZE)
233
234 print('{:>5,} complete training samples'.format(len(complete_dataset)))
235 print('{:>5,} complete dataloader batches'.format(len(complete_dataloader)))
236
237 ##########################################################################################
238
239 import datetime
240 import matplotlib.pyplot as plt
241
242 ##########################################################################################
243
244 import torch.nn as nn
245
246
247 class BertimbauRegressor(nn.Module):
248
249     def __init__(self, drop_rate=0.2, freeze_bertimbau=False):
250         super(BertimbauRegressor, self).__init__()
251         # 768 * 2 por causa da concatenação
252         D_in, D_out = 1536, 1
253
254         # self.regressor = nn.Sequential(
255         #     nn.Dropout(drop_rate),
256         #     nn.Linear(D_in, D_out))
257
258         self.regressor = nn.Sequential(
259             nn.Dropout(drop_rate),
260             nn.Linear(1536, 512),  # First dense layer
261             nn.ReLU(),  # Activation function
262             nn.Dropout(drop_rate),
263             nn.Linear(512, 128),  # Second dense layer
264             nn.ReLU(),  # Activation function
265             nn.Dropout(drop_rate),
266             nn.Linear(128, 1)  # Final layer to predict grade
267         )
268
269     def forward(self, r_input_ids, r_attention_masks, tm_input_ids, tm_attention_masks):
270         r_outputs = bertimbau(r_input_ids, r_attention_masks, output_hidden_states=True)
271         r_class_label_output = r_outputs.hidden_states[-1][:, 0, :]
272
273         tm_outputs = bertimbau(tm_input_ids, tm_attention_masks, output_hidden_states=True)
274         tm_class_label_output = tm_outputs.hidden_states[-1][:, 0, :]
```

```
275
276         concatenated_output = torch.cat((r_class_label_output, tm_class_label_output), dim=-1)
277
278         outputs = self.regressor(concatenated_output)
279         return outputs
280
281
282 model = BertimbauRegressor(drop_rate=0.2)
283
284 # tells model to run on GPU
285 model.to(device)
286
287 ###############################################################################################
288
289 # Get all of the model's parameters as a list of tuples.
290 params = list(bertimbau.named_parameters())
291
292 print('The BERT model has {:} different named parameters.\n'.format(len(params)))
293
294 print('==== Embedding Layer ====\n')
295
296 for p in params[0:5]:
297     print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
298
299 print('\n==== First Transformer ====\n')
300
301 for p in params[5:21]:
302     print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
303
304 print('\n==== Output Layer ====\n')
305
306 for p in params[-4:]:
307     print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
308
309 ###############################################################################################
310
311 from transformers import AdamW
312 from transformers import get_linear_schedule_with_warmup
313
314 # O artigo do BERT recomenda de 2-4 epocas. Checar se 4 épocas não está causando over-fitting.
315 EPOCHS = 3
316
317 loss_function = nn.MSELoss()
318
319 LR = 5e-5  # recommended lrs -> 5e-5, 3e-5, 2e-5
320
321 total_steps = len(complete_dataloader) * EPOCHS
322
323 optimizer = AdamW(bertimbau.parameters(), lr=LR, eps=1e-8)
324
325 # Create the learning rate scheduler.
326 scheduler = get_linear_schedule_with_warmup(optimizer,
327                                             num_warmup_steps=0,  # Default value in run_glue.py
328                                             num_training_steps=total_steps)
329
330 loss_function = torch.nn.MSELoss()
331
332 ###############################################################################################
333
334 import time
```

```
335 import torch
336 from torch.utils.data import DataLoader, SubsetRandomSampler, Subset
337 from sklearn.model_selection import KFold
338 import numpy as np
339 import copy
340 import os
341
342 NUM_FOLDS = 10  # Adjust based on your dataset size and requirements
343
344 overall_train_loss_values = []
345 overall_val_loss_values = []
346 overall_train_mae_values = []
347 overall_val_mae_values = []
348
349 all_fold_time = 0
350
351
352 def train_with_cross_validation(model, optimizer, scheduler, loss_function, epochs, dataset, device, save_p
353                                 clip_value=2, logging_interval=40):
354     best_model_wts = copy.deepcopy(model.state_dict())
355     best_val_mae = float('inf')  # Initialize with a high number
356
357     kf = KFold(n_splits=NUM_FOLDS, shuffle=True, random_state=42)
358
359     for fold, (train_indices, val_indices) in enumerate(kf.split(dataset)):
360         print(f"\n==================== Fold {fold + 1} / {NUM_FOLDS} ======================")
361
362         fold_train_loss = 0
363         fold_train_mae = 0
364         fold_val_loss = 0
365         fold_val_mae = 0
366         global all_fold_time
367
368         fold_t0 = time.time()
369
370         assert max(train_indices) < len(dataset) and max(val_indices) < len(dataset)
371
372         train_sampler = torch.utils.data.SubsetRandomSampler(train_indices)
373         train_dataloader = DataLoader(dataset, sampler=train_sampler, batch_size=BATCH_SIZE)
374
375         val_subset = torch.utils.data.Subset(dataset, val_indices)
376         val_dataloader = DataLoader(val_subset, batch_size=BATCH_SIZE, shuffle=False)
377
378         # Save current fold datasets
379         folds_save_dir = os.path.join(save_dir, "folds")
380         if not os.path.exists(folds_save_dir):
381             os.makedirs(folds_save_dir)
382
383         torch.save(train_indices, os.path.join(folds_save_dir, f'train_indices_fold_{fold + 1}.pt'))
384         torch.save(val_indices, os.path.join(folds_save_dir, f'val_indices_fold_{fold + 1}.pt'))
385
386         for epoch in range(epochs):
387             print(f"\n========== Epoch {epoch + 1} / {epochs} ==========")
388
389             epoch_start_time = time.time()
390
391             # Training Phase
392             model.train()
393             total_loss = 0
394             mean_absolute_error = 0
```

```
395
396             for batch in train_dataloader:
397                 model.zero_grad()  # Zero the gradients on each iteration
398                 optimizer.zero_grad()
399
400                 # Unpack this training batch from our dataloader and copy each tensor to the GPU.
401                 batch_r_input_ids = batch['r_input_ids'].to(device)
402                 batch_r_attention_masks = batch['r_attention_mask'].to(device)
403                 batch_tm_input_ids = batch['tm_input_ids'].to(device)
404                 batch_tm_attention_masks = batch['tm_attention_mask'].to(device)
405                 batch_labels = batch['nota_competencia_2'].to(device)
406
407                 outputs = model(
408                     r_input_ids=batch_r_input_ids,
409                     r_attention_masks=batch_r_attention_masks,
410                     tm_input_ids=batch_tm_input_ids,
411                     tm_attention_masks=batch_tm_attention_masks
412                 )
413
414                 loss = loss_function(outputs.squeeze(), batch_labels.squeeze())
415                 total_loss += loss.item()
416
417                 mean_absolute_error += get_mean_absolute_error(outputs, batch_labels)
418                 loss.backward()
419
420                 torch.nn.utils.clip_grad_norm_(model.parameters(), clip_value)
421                 optimizer.step()
422                 scheduler.step()
423
424             avg_train_loss = total_loss / len(train_dataloader)
425             avg_train_mae = mean_absolute_error / len(train_dataloader)
426
427             overall_train_loss_values.append(avg_train_loss)
428             overall_train_mae_values.append(avg_train_mae)
429
430             # ========================================================
431             #                      Validation
432             # ========================================================
433
434             model.eval()
435
436             total_val_loss = 0
437             mean_absolute_error = 0
438
439             for batch in val_dataloader:
440                 batch_r_input_ids = batch['r_input_ids'].to(device)
441                 batch_r_attention_masks = batch['r_attention_mask'].to(device)
442                 batch_tm_input_ids = batch['tm_input_ids'].to(device)
443                 batch_tm_attention_masks = batch['tm_attention_mask'].to(device)
444                 batch_labels = batch['nota_competencia_2'].to(device)
445
446                 with torch.no_grad():
447                     outputs = model(
448                         r_input_ids=batch_r_input_ids,
449                         r_attention_masks=batch_r_attention_masks,
450                         tm_input_ids=batch_tm_input_ids,
451                         tm_attention_masks=batch_tm_attention_masks
452                     )
453
454                     loss = loss_function(outputs.squeeze(), batch_labels.squeeze())
```

```
                    total_val_loss += loss.item()
                    mean_absolute_error += get_mean_absolute_error(outputs, batch_labels)

            avg_val_loss = total_val_loss / len(val_dataloader)
            avg_val_mae = mean_absolute_error / len(val_dataloader)

            overall_val_loss_values.append(avg_val_loss)
            overall_val_mae_values.append(avg_val_mae)

            fold_train_loss += avg_train_loss
            fold_train_mae += avg_train_mae
            fold_val_loss += avg_val_loss
            fold_val_mae += avg_val_mae

            if avg_val_mae < best_val_mae:
                best_val_mae = avg_val_mae
                best_model_wts = copy.deepcopy(model.state_dict())
                print(f"New best model found at fold {fold + 1} with MAE: {avg_val_mae}\n")

            print(f"Training Loss: {avg_train_loss:.2f}, MAE: {avg_train_mae:.2f}")
            print(f"Validation Loss: {avg_val_loss:.2f}, MAE: {avg_val_mae:.2f}")

            epoch_end_time = time.time()
            epoch_time = epoch_end_time - epoch_start_time
            print(f"Epoch {epoch + 1}/{epochs} took: {format_time(epoch_time)}")

        fold_avg_train_loss = fold_train_loss / epochs
        fold_avg_train_mae = fold_train_mae / epochs
        fold_avg_val_loss = fold_val_loss / epochs
        fold_avg_val_mae = fold_val_mae / epochs

        print("================================")
        print(f"\nFold {fold + 1} Average Metrics\n")

        print(f"fold_{fold + 1}_train_loss_values: {overall_train_loss_values[-epochs:]}")
        print(f"fold_{fold + 1}_val_loss_values: {overall_val_loss_values[-epochs:]}")
        print(f"fold_{fold + 1}_train_mae_values: {overall_train_mae_values[-epochs:]}")
        print(f"fold_{fold + 1}_val_mae_values: {overall_val_mae_values[-epochs:]}")

        print(f"Average Training Loss: {fold_avg_train_loss:.2f}")
        print(f"Average Training MAE: {fold_avg_train_mae:.2f}")
        print(f"Average Validation Loss: {fold_avg_val_loss:.2f}")
        print(f"Average Validation MAE: {fold_avg_val_mae:.2f}")

        fold_time = time.time() - fold_t0
        all_fold_time += fold_time
        print(f"\nFold {fold + 1} took: {format_time(fold_time)}")

    # Print overeall train and validation loss and mae
    print(f"overall_train_loss_values{overall_train_loss_values}")
    print(f"overall_val_loss_values{overall_val_loss_values}")
    print(f"overall_train_mae_values{overall_train_mae_values}")
    print(f"overall_val_mae_values{overall_val_mae_values}")

    # Calculate overall average metrics
    avg_train_loss = np.mean(overall_train_loss_values)
    avg_val_loss = np.mean(overall_val_loss_values)
    avg_train_mae = np.mean(overall_train_mae_values)
    avg_val_mae = np.mean(overall_val_mae_values)

```

```
515     print(f"\n============================== Training complete! ==============================")
516     print("Overall summary:")
517     print(f"Average Training Loss: {avg_train_loss:.2f}")
518     print(f"Average Validation Loss: {avg_val_loss:.2f}")
519     print(f"Average Training MAE: {avg_train_mae:.2f}")
520     print(f"Average Validation MAE: {avg_val_mae:.2f}")
521     print(f"All folds took: {format_time(all_fold_time)}")
522
523     # Load best model
524     model.load_state_dict(best_model_wts)
525     print(f"Best model was from fold with MAE: {best_val_mae:.2f}")
526
527     return model, overall_train_loss_values, overall_val_loss_values, overall_train_mae_values, overall_val
528
529
530 ##############################################################################################
531
532 save_dir = f"/content/drive/MyDrive/Colab Notebooks/BERT_regressor/BERT_base_experimentos/treinamentos/k-fo
533 if not os.path.exists(save_dir):
534     os.makedirs(save_dir)
535
536 ##############################################################################################
537
538 model, train_loss_values, val_loss_values, mae_train_values, mae_val_values = train_with_cross_validation(m
539                                                                                                             c
540                                                                                                             s
541                                                                                                             l
542                                                                                                             E
543                                                                                                             c
544                                                                                                             d
545                                                                                                             s
546                                                                                                             c
547
548 ##############################################################################################
549
550 import matplotlib.pyplot as plt
551
552
553 def plot_train_val_losses(train_losses, val_losses, title="Loss vs Epochs", xlabel="Epochs", ylabel="RMSE L
554     """
555         Plots training and validation losses over epochs
556     """
557
558     epochs = range(1, len(train_losses) + 1)
559     print(epochs)
560     # Plot curves
561     plt.plot(epochs, train_losses, color='r', label="MSE Loss")
562     plt.plot(epochs, val_losses, color='b', label="MAE")
563
564     # Naming the axis
565     plt.xlabel(xlabel)
566     plt.ylabel(ylabel)
567     plt.title(title)
568
569     ax = plt.gca()
570     ax.set_xticks(np.arange(1, 6, 1))
571
572     plt.legend()
573     plt.show()
574
```

```
575
576 ##########################################################################################
577
578 def predict(model, tokenizer, essay, auxiliary_text):
579     """
580     Make predictions with the BertimbauRegressor model.
581
582     Args:
583     - model (torch.nn.Module): The trained BertimbauRegressor model.
584     - tokenizer (Transformers Tokenizer): The tokenizer used for preprocessing.
585     - essay (str): The main text of the essay.
586     - auxiliary_text (str): Additional or auxiliary information about the essay.
587
588     Returns:
589     - float: The predicted grade for the essay.
590     """
591
592     model.to(device)
593
594     # Tokenize the main essay and auxiliary text separately
595     essay_tokens = tokenizer(essay, return_tensors="pt", truncation=True, padding=True, max_length=500).to(
596     auxiliary_tokens = tokenizer(auxiliary_text, return_tensors="pt", truncation=True, padding=True, max_le
597         device)
598
599     # Ensure model is in evaluation mode
600     model.eval()
601
602     # No gradient computation needed during prediction
603     with torch.no_grad():
604         prediction = model(essay_tokens['input_ids'], essay_tokens['attention_mask'], auxiliary_tokens['inp
605                            auxiliary_tokens['attention_mask'])
606
607     # Extract the grade from the tensor
608     return prediction.item()
609
610
611 ##########################################################################################
612
613 import random
614
615 rand = random.randrange(0, len(df))
616
617 redacao = df.loc[rand, 'redacao']
618 texto_motivador = df.loc[rand, 'texto_motivador']
619 nota = df.loc[rand, 'nota_competencia_2']
620
621 print(f'redacao: {redacao[:100]}')
622 print(f'texto motivador: {texto_motivador[:100]}')
623 print(f'nota: {nota}')
624
625 pred = predict(model, tokenizer, redacao, texto_motivador)
626 print(f'pred: {pred}')
627
628 ##########################################################################################
629
630 save_dir = f"/content/drive/MyDrive/Colab Notebooks/BERT_regressor/bert_base/treinamentos/k-fold/LR_{str(LR
631 if not os.path.exists(save_dir):
632     os.makedirs(save_dir)
633
634 ##########################################################################################
```

```
635
636 torch.save(model.state_dict(), f"{save_dir}/bert_base_cls_tokens_lr_{str(LR)}_{EPOCHS}_epochs.json")
637
638 ##############################################################################################
639
640 bertimbau_regressor = BertimbauRegressor()
641 bertimbau_regressor.load_state_dict(torch.load(f"{save_dir}/bert_base_cls_tokens_lr_{str(LR)}_{EPOCHS}_epoc
642 bertimbau_regressor.eval()
643
644 ##############################################################################################
645
646 import matplotlib.pyplot as plt
647 import itertools
648 import numpy as np
649
650
651 def plot_lr_performance_comparison(lr_metrics, chart_type='line'):
652     """
653     Plots a comparison of validation loss and MAE for different
654     learning rates across folds using unique colors for each line.
655
656     Args:
657     lr_metrics (dict): A dictionary where keys are learning rates and values are dictionaries
658     containing lists of validation loss and MAE for each fold.
659
660     chart_type (str): Type of chart to plot ('line' or 'bar').
661     """
662     learning_rates = list(lr_metrics.keys())
663     n_folds = len(next(iter(lr_metrics.values()))['loss'])  # Assuming all learning rates have the same nun
664     color_cycle = plt.cm.tab10(np.linspace(0, 1, len(learning_rates) * 2))  # Create a color cycle
665
666     plt.figure(figsize=(12, 6))
667     color_iter = itertools.cycle(color_cycle)  # Iterator for colors
668
669     for lr in learning_rates:
670         folds = range(1, n_folds + 1)
671         color = next(color_iter)
672
673         if chart_type == 'line':
674             plt.plot(folds, lr_metrics[lr]['loss'], label=f'LR={lr} Loss', color=color)
675             plt.plot(folds, lr_metrics[lr]['mae'], label=f'LR={lr} MAE', color=color, linestyle='--')
676         elif chart_type == 'bar':
677             plt.bar([x - 0.1 + 0.2 * i for i, x in enumerate(folds)],
678                     lr_metrics[lr]['loss'], width=0.1, label=f'LR={lr} Loss', color=color)
679             plt.bar([x + 0.1 + 0.2 * i for i, x in enumerate(folds)],
680                     lr_metrics[lr]['mae'], width=0.1, label=f'LR={lr} MAE', color=color)
681
682     plt.title('Treinamento Loss vs MAE - comparação ao longo de diferentes taxas de aprendizado')
683     plt.xlabel('Folds')
684     plt.ylabel('MSE Loss & MAE')
685     plt.xticks(range(1, n_folds + 1))
686     plt.legend()
687     plt.tight_layout()
688
689     plt.show()
690
691
692 ##############################################################################################
693
694 lr_metrics = {
```

```
    '5e-5': {'loss': [17721.79, 17804.2, 17870.17, 17936.18, 17646.00], 'mae': [128.1, 128.42, 128.76, 129.
    '3e-5': {'loss': [17735.11, 17806.66, 17886.36, 17945.02, 17659.43],
             'mae': [128.15, 128.44, 128.82, 129.19, 127.89]},
    '2e-5': {'loss': [17633.15, 17675.1, 17761.4, 17821.22, 17550.11], 'mae': [127.76, 127.93, 128.36, 128.
}

plot_lr_performance_comparison(lr_metrics, chart_type='line')  # or 'bar'

##############################################################################################
```

APÊNDICE B – ARTIGO DO TABALHO

Análise da aderência semântica de redações do ENEM ao tema: uma abordagem baseada no BERTimbau

Raphael Ramos da Silva[1], **Osmar de Oliveira Braz Junior**[2], **Renato Fileto**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul

[2]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brasil

r.r.raphael.silva@gmail.com, {osmar.braz}@posgrad.ufsc.br

{renato.fileto}@posgrad.ufsc.br

Abstract. *Every year, thousands of Brazilian students undergo the largest teaching assessment in the country, the National Secondary Education Exam (ENEM). The exam not only assesses the quality of national basic education but is also used for admission to higher education institutions. In addition to multiple-choice questions covering major areas of knowledge, the test also consists of an essay that must be written in accordance with the dissertation-argumentative style. The manual process of evaluating essays is expensive. The estimated cost in 2015 for each writing correction was R$15.88. That same year, 6.4 million essays were corrected, and the exam cost to govern around R$ 1 billion reais. Taking this into consideration, the present work proposes the use of contextualized embeddings of words generated by the pre-trained BERT model for the Brazilian Portuguese language. State-of-the-art technology in the NLP field. To this end, one regression models and another classification model will be developed based on fine-tuning of the language model. The classification model presents good generalization capacity to new data, reaching an accuracy of 81.73 and an F1-score of 0.80, for the learning rate 5e-5 in the validation dataset. The results of the regression model suggest low adaptability to solve the proposed problem, with its MAE greater than 120 for training and validation.*

Keywords: *Adherence of essays no themes, ENEM essays, semantic similatiry, contextualizaed language models, BERT.*

Resumo. *Todos os anos, milhares de estudantes brasileiros se submetem à maior avaliação de ensino do país, o Exame Nacional do Ensino Médio (ENEM). O exame avalia não só a qualidade da educação básica nacional, mas também é utilizado para o ingresso em instituições de ensino superior. Além de questões de múltipla escolha abrangendo as grandes áreas do conhecimento, a prova também é composta por uma redação que deve ser redigida obedecendo o estilo dissertativo-argumentativo. O processo manual de avaliação das redações é dispendioso. O custo estimado em 2015 para cada correção de redação era de R$15,88. Nesse mesmo ano, 6,4 milhões de redações foram corrigidas. Levando isso em consideração, o presente trabalho propõe o uso de embeddings contextualizados de palavras gerados pela modelo BERT pré-treinado para a língua portuguesa do Brasil. Tecnologia em estado-da-arte em PLN.*

Para tal, será desenvolvido um modelo de regressão e outro de classificação a do fine-tuning do modelo de linguagem. O modelo de classificação apresenta boa capacidade de generalização a novos dados, atingindo a acurácia 81,73 e F1-score de 0,80, para a taxa de aprendizado 5e-5 no conjunto de validação. Já os resultados do modelo de regressão sugerem baixa adaptabilidade para resolver o problema proposto tendo seu MAE superior a 120 para treinamento e validação. ***Palavras-chave****: Aderência de ensaios/redações a temas, redações do ENEM, similaridade semântica, modelos contextualizados de linguagem, BERT, regressão, classificação.*

1. Introdução

O advento das *redes sociais*, *mídias digitais* e o *big data* propiciou nas últimas décadas um aumento considerável no volume de dados gerados por usuários na Internet. A maior parte desses dados estão em formato texto e seu volume estimado é na ordem de petabytes (Baeza-Yates and Ribeiro-Neto 2013). Estima-se que até 2025, o volume de dados no mundo atingirá a marca de 175 *zetabytes* (David Reinsel 2022), sendo 1 *zetabyte* o equivalente a 1 trilhão de *gigabytes.*(Baeza-Yates and Ribeiro-Neto 2013).

Nesse cenário, ferramentas computacionais capazes de processar a linguagem humana têm ganhado destaque, e a demanda pela solução de problemas reais a partir de técnicas de PLN emergem aos montes nos mais diferentes setores da sociedade. Atualmente, técnicas de PLN são utilizadas em diversas áreas como, por exemplo, na automatização de correção de questões discursivas (Oliveira et al. 2020), na análise de *feedbacks* em ambientes de aprendizagem virtual (Cavalcanti et al. 2020b), na análise de propriedades linguísticas em redações do ENEM (Bertucci 2021), na análise de coerência na postagens de fóruns de dúvidas (Braz and Fileto 2021) ou ainda na mineração de padrões linguísticos em discursos proferidos por políticos durante a corrida eleitoral americana de 2016 (Sorato et al. 2020).

A popularização de plataformas de ensino à distância também tem demandado ferramentas capazes de dar suporte ao processamento e análise de textos produzidos por estudantes, devido ao seu grande potencial em amenizar o esforço humano empregado na realização das tarefas de avaliação de textos. Um dos maiores expoentes dessa demanda é o Exame Nacional do Ensino Médio (ENEM), visto que em 2015 o custo para a correção manual de cada redação era de *R$15,88*. Naquele mesmo ano, foram registradas *6,4 milhões* de redações corrigidas, com um custo total de pelo menos *101 milhões* aos cofres públicos.

1.1. Metodologia

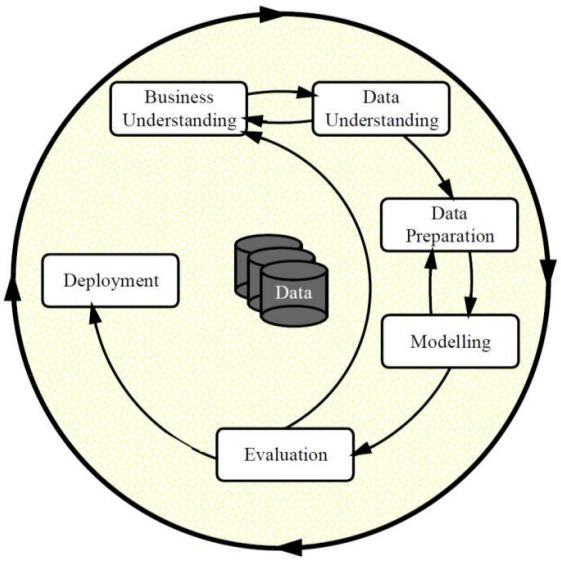
O presente trabalho será orientado pela pelo processo de referência *Cross-Industry Standard Process for Data Mining* - CRISP-DM (Wirth and Hipp 2000). Tal processo consiste de uma sucessão de etapas iterativas que buscam sistematizar a solução de problemas dirigidos a dados. As etapas compreendem: 1) entendimento do negócio; 2) entendimento dos dados; 3) preparação dos dados; 4) modelagem; 5) avaliação e 6) implantação;

2. Fundamentação Teórica

2.1. A redação do ENEM

A redação do ENEM é um componente obrigatório da prova de língua portuguesa e literatura e exige do candidato a elaboração de um texto em prosa, do tipo dissertativo-argumentativo, coerente e bem estruturado sobre um determinado tema. O tema da

Figura 1. Fases do CRISP-DM



Fonte: (Wang et al. 2018)

redação geralmente está relacionado a uma questão social, cultural ou política relevante para a sociedade brasileira.

A Tabela 1 descreve os critérios para avaliar essas redações que são divididos em 5 competências. Cada uma das competências pode receber de 0 a 200 pontos, e a soma dos pontos em todas as competências comporá a notal final.

Tabela 1. Competências gerais avaliadas em redações do ENEM (INEP 2022)

Competência 1	Demonstrar domínio da modalidade escrita formal da língua portuguesa.
Competência 2	Compreender a proposta de redação e aplicar conceitos das várias áreas de conhecimento para desenvolver o tema, dentro dos limites estruturais do texto dissertativo-argumentativo em prosa.
Competência 3	Selecionar, relacionar, organizar e interpretar informações, fatos, opiniões e argumentos em defesa de um ponto de vista.
Competência 4	Demonstrar conhecimento dos mecanismos linguísticos necessários para a construção da argumentação
Competência 5	5 Elaborar proposta de intervenção para o problema abordado, respeitando os direitos humanos.

Fonte: elaborada pelo autor

A redação é uma parte importante do exame porque avalia não apenas o conhecimento do aluno em língua portuguesa, mas também sua capacidade de expressar suas ideias de forma clara e persuasiva. Também é visto como uma forma de promover o pensamento crítico e a consciência social entre os estudantes. Além da estrutura obrigatória apresentada, o texto deverá respeitar uma série de outros critérios para não ser atribuído nota 0

(zero), sendo alguns deles: fuga total ao tema, não obediência ao tipo textual dissertativo-argumentativo, textos com menos 7 (sete) linhas de extensão, etc. (INEP 2022)

2.2. Competência 2

A segunda competência avaliativa é a responsável por identificar se o candidato compreendeu a proposta da redação, composta por um tema específico, e se desenvolveu sua argumentação a partir disso. Conjuntamente, é verificado se o texto redigido está na forma dissertativo-argumentativa. O que faz dessa competência uma das mais importantes é o fato de que ela é a única capaz de anular uma redação, isto é, atribuir nota 0 (zero) as todas as competências. Isso ocorrerá caso o conteúdo do texto redigido não tenha compatibilidade com a proposta de redação, caracterizando fuga total ao tema.

2.3. Essay-BR

O Essay-BR é um *corpus* de ensaios dissertativos escritos em língua portuguesa cujos textos seguem o padrão mais atual do ENEM. A iniciativa se deu quando um grupo de pesquisa da Universidade Federal do Piauí (UFPI), em contribuição para os avanços nas pesquisas em Avaliação Automática de Redações (do inglês, *Automatic Essays Scoring - AES*), decidiu criar um *corpus* atualizado com textos escritos em português do Brasil. O *corpus* contém 4,570 documentos, divididos em 86 tópicos, que abarcam temas desde direitos humanos, política e cultura a *fake news*, saúde e COVID-19.

As redações foram coletadas durante o período de Dezembro de 2015 a Abril de 2020, e em sua maioria vieram do já citado site, Banco de redações UOL e também do Vestibular UOL [1]. Os textos coletados foram então pré-processados, onde se removeu *tags* HTML e anotações dos avaliadores e as pontuações foram normalizadas, já que o site utilizava uma estratégia diferente da empregada pelo exame oficial.

No momento em que este trabalho é realizado, o *dataset* do foi atualizado[2] e o número de textos e temas passam a ser 6.577 e 150, respectivamente. A Tabela 2 traz detalhes descritivos sobre o *corpus* atualizado.

Tabela 2. Descrição do *copus* Essay-BR

Detalhes	Essay-BR *corpus*
Tipo textual	Dissertativo-argumentativo
Nível de linguagem dos autores	Ensino Médio
Pontuação	Holística
Número de textos	6.577
Número de tópicos	150
Número de competências	5
Intervalo de pontuação por competência	[0 - 200]
Pontuações	[0,40,80,120,160,200]
Intervalo da pontuação geral	[0 - 1000]

Fonte: elaborada pelo autor

[1] ⟨https://vestibular.brasilescola.uol.com.br/banco-de-redacoes⟩
[2] ⟨https://github.com/lplnufpi/essay-br⟩

2.4. BERT

BERT, do inglês, *Bidirectional Encoder Representations from Transformers*, é um modelo pré-treinado de representação linguística lançado pela *Google AI* em 2018. O modelo traz avanços consideráveis para tarefas de PLN, sendo uma das mais relevantes a utilização de *embeddings* contextualizados em que o sentido de uma palavra é influenciado pelas palavras ao seu redor. Ao contrário de modelos anteriores, como o Word2Vec[3] e o Glove (Pennington et al. 2014) que geravam apenas *embeddings* estáticos e fixos, a preservação de contexto entre palavras possibilita a captação de nuances e desambiguação de sentido.

A arquitetura que viabiliza a eficiência do modelo é chamada de *Tranformers* (Vaswani et al. 2017), que utiliza mecanismos de auto-atenção ou bidirecionalidade para aplicar pesos às palavras de uma sentença. Esse processo ocorre durante a etapa de *encoding*.

O BERT é treinado em cima de grandes bases de dados textuais sobretudo nas tarefas de predição de palavras faltantes i.e. *Masked Language Model (MLM)* e predição de próxima sentença, do inglês *Next Sentence Prediction (NSP)* (Devlin et al. 2018). Tais tarefas, ainda que não resolvam problemas críticos em PLN, foram as responsáveis pela capacidade sofisticada do modelo em compreender nunces da línguagem (McCormick 2023) devido a complexidade inerente das tarefas. Uma outra vantagem dessa estratégia, é que ela viabilizou o treinamento de um grande quantidade dados não classificados. Uma vez pré-treinamento, o modelo pode ser ajustado, isto é, *fine-tuned*, para uma série de tarefas em como: Análise de sentimento, máquinas de tradução, Reconhecimento de Entidade Nomeadas (NER), Q&A, classificação de textos, entre outras.

2.4.1. Símbolos especiais

Os *tokens [CLS]* e *[SEP]* são símbolos especiais utilizados para que o modelo identifique o início e o fim de cada entrada (Devlin et al. 2018). O também símbolo especial *[UNK]* representa palavras desconhecidas ao vocabulário. Por fim, *tokens* iniciados pelos caracteres "*##*" representam palavras que foram subdivididas em sub-palavras ou ainda em um único carácter. Essa é uma das formas que o modelo utiliza para lidar com palavras desconhecidas, do inglês *Out-of-Vocabulary - OOV words*. Desse modo, se uma palavra não é encontrada no vocabulário, a chance de alguma de suas sub-palavras ser encontrada aumenta.

Durante a etapa de treinamento o *token* especial [CLS] é também utilizado pelo classificador para armazenar o agregamento da representação da sequencia de texto processada ao longo das 12 camadas do modelo. Os *embeddings* utilizados no presente trabalho são extraídos da última camada do , como recomenda os autores (Devlin et al. 2018).

Outro destaque que precisa ser dado, é a forma como o BERT lida com textos de tamanhos diferentes, o que inevitavelmente é o caso no presente experimento. Textos com mais *tokens* que o limite máximo, 512, são truncados, e textos com menos *tokens* que o limite utilizam a técnica de *padding*. Essa técnica consiste em adicionar o *token* especial [PAD] à representação do texto. Dessa forma, *tokens* desse tipo são ignorados pelo mecanismo de auto-atenção do modelo.

[3] ⟨https://code.google.com/archive/p/word2vec/⟩

2.5. BERTimbau

BERTimbau (Souza et al. 2020) é um modelo treinado em cima do BERT que utilizou técnicas de transferência de conhecimento, do inglês *Transfer Learning*, para ser adaptado para a língua portuguesa do Brasil. O modelo alcançou performances em estado-da-arte em três tarefas: Reconhecimento de Entidades Nomeadas (NER), Similaridade Textual entre Sentenças, e Inferência de Linguagem Natural (ILN) (Souza et al. 2020). O modelo é disponibilizado em dois tamanhos: *Base* (L = 12 camadas, H = 768, 12 *attention heads*, e 110M parâmetros) e o Large co(L = 12 camadas, H = 1024, 24 *attention heads*, e 330M parâmetros)(Souza et al. 2020). Nosso objetivo é experimentar diferentes formas de obter *embeddings* contextualizados dos textos processados. Durante o processo de *tokenização* o *token* **CLS** é colocado no início de cada sequencia de texto. Este é um *token* especial que o modelo foi condicionado a utilizar para codificar a representação das sequencias de entrada para tarefas de classificação (Devlin et al. 2018). A saída da última camada do modelo consiste na agregação de todos essas representações e por isso é a forma mais comum de se obter *embeddings* contextualizados.

3. Desenvolvimento

A presente seção reporta as principais etapas realizadas no desenvolvimento do trabalho, descrevendo processos, atividades, métodos, técnicas e critérios relevantes utilizados em cada etapa.

Inicialmente estabelecemos a hipótese de que quanto maior a nota na categoria 2 de uma dada redação, maior será a similaridade do cosseno entre os *embeddings* redação e os de seus respectivos textos motivadores. Esta hipótese foi testada calculando similaridades baseadas em *embeddings* de documentos e de sentenças, como detalhado nas subseções a seguir. A partir desses experimentos, foi possível definir a melhor abordagem para a extração dos *embeddings* a serem utilizados nos experimentos. Os resultados dessa primeira etapa são detalhados no capítulo a seguir.

Foram inicialmente gerados *embeddings* para documentos inteiros com duas versões do modelo , *base* e *large*. Em seguida foi calculada a similaridade de cosseno entre cada redação e seu respectivo texto motivador. Logo após, uma regressão linear foi traçada entre as similaridades obtidas e a nota da respectiva redação na categoria 2. Os resultados obtidos nessa etapa são melhor detalhados na Subseção 3.1.

3.1. Análise exploratória

Etapa crucial em qualquer tarefa de PLN, a EDA permite a visualização e a sumarização das principais características de um *dataset*. Além de validar se as hipóteses iniciais do problema identificado faz sentido, essa técnica também possibilita uma melhor compreensão dos dados através da identificação de padrões, correlações, anomalias e entre outros. Os resultados aqui descritos refletem apenas o conjunto de dados efetivamente utilizado ao longo de todo o experimento. Isso porque, devido a limitação do BERT quanto ao número máximo de *tokens* processados por entrada, realizamos um filtro onde apenas redações e textos motivadores com menos de 500 *tokens* foram levadas em consideração (**??**). O limite máximo do BERT é 512.

3.1.1. Estatísticas descritivas

A estatística descritiva busca descrever um conjunto de dados a partir de várias técnicas de sumarização. Algumas das métricas produzidas são medidas de tendência central, como

a média, mediana, e moda, medidas de variabilidade como desvio padrão e variância, e os valores máximos e mínimos. A Tabela 3 apresenta as principais dessas métricas para a nota na competência 2 e a nota final.

Tabela 3. Estatísticas descritivas para as notas

Nota competência 2	
Métrica	**Valor**
Quantidade	2653
Média	130,48
Desvio padrão	36,16
Min	0,00
25%	120,00
50%	120,00
75%	160,00
Max	200,00

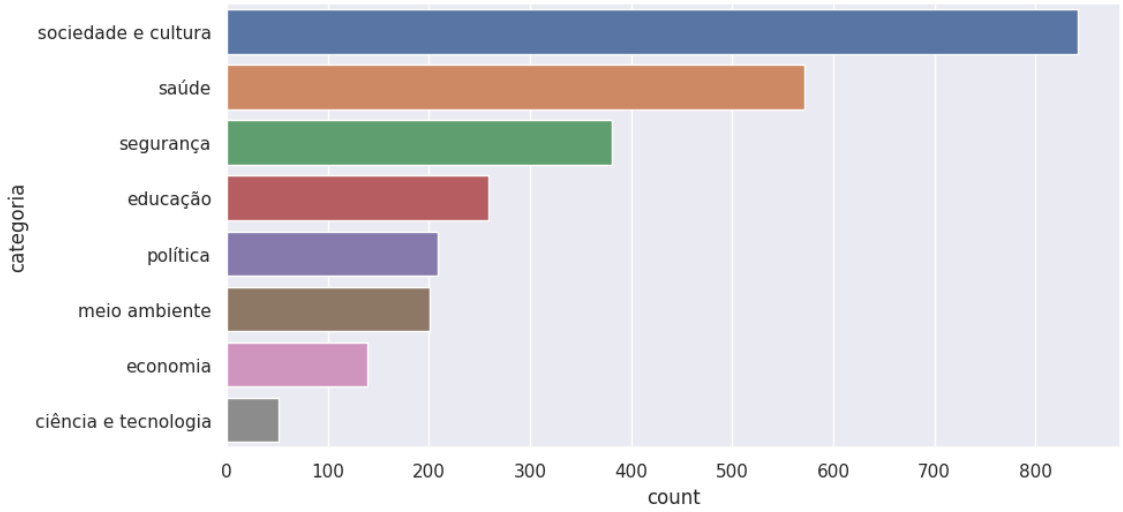
Nota Final	
Métrica	**Valor**
Quantidade	2653
Média	618,00
Desvio padrão	168,02
Min	0,00
25%	520,00
50%	640,00
75%	760,00
Max	1.000,00

Fonte: elaborado pelo autor

3.1.2. Distribuição de categorias por temas

Cada redação no conjunto de dados foi escrita sob um tema, e cada tema pertence a uma categoria, como por exemplo: Economia, Política, Educação, etc. A Figura 2 mostra a distribuição por temas ordenados por incidência.

Figura 2. Distribuição de categorias por tema



Fonte: elaborado pelo autor

3.1.3. Análise comparativa entre notas na competência 2 e temas

Ao agrupar as notas por categoria de tema e obter suas médias, se torna possível identificar as categorias que estudantes tendem a se distanciar mais ou menos de um dado

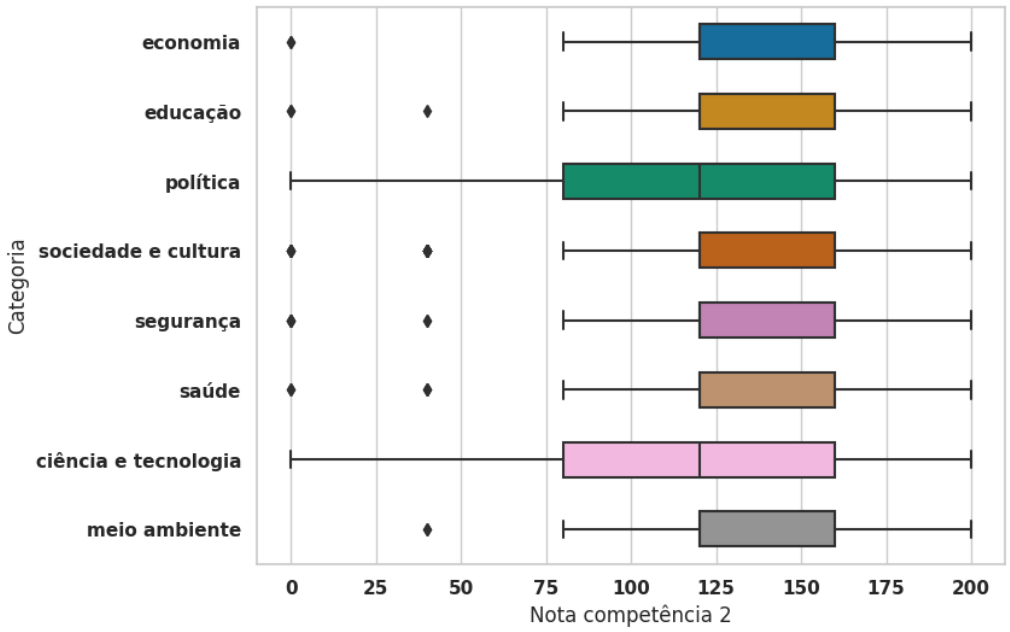
tema, ou seja, atingirem notas maiores ou menores. Por exemplo, **Saúde** e **Segurança**, são categorias em que as redações no conjunto de dados apresentam maior aderência ao tema, ao passo que **Política** e **Ciência e tecnologia** são categorias mais penalizadas. Tal constatação pode significar que, no geral, estudantes possuem mais dificuldade em discorrer sobre certos temas sem perder de vista a ideia central proposta. A Tabela 4 traz os temas e suas respectivas notas médias ordenadas de forma descendente. E o diagrama de caixas, apresentado pela Figura 3 identifica o desempenho médio dos estudante por temas. Nota-se que os temas **Política** e **Ciência e Tecnologia** possuem os menores limites inferiores, podendo chegar a zero. No subconjunto analisado apenas esses dois temas possuem essa característica.

Tabela 4. Nota média na competência 2 por categoria de tema

Tema	Nota
Saúde	136,88
Segurança	135,54
Meio ambiente	134,13
Educação	127,88
Economia	127,48
Sociedade e cultura	126,46
Política	124,02
Ciência e tecnologia	120,78

Fonte: elaborado pelo autor

Figura 3. Nota média na competência 2 por categoria de tema



Fonte: elaborado pelo autor

3.2. Materiais e Métodos

Os experimentos foram realizados na plataforma *Google colaboratory* na versão PRO+. A plataforma oferece uma série de *Graphic Processing Units* GPUs, entre elas a Tesla T4, utilizada no presente trabalho. A linguagem de programação Python foi utilizada ao longo de todo o desenvolvimento na versão 3.10.12. Uma das principais APIs - *Application Programming Interface*, do inglês, utilizadas disponibilizada pela biblioteca *transformers*[4] desenvolvida pela comunidade **HuggingFace**[5]. O código fonte e os dados para reproduzir os experimentos e resultados reportados neste trabalho estão disponíveis no GitHub[6].

3.3. Preparação dos dados

Nesta etapa foi realizada a limpeza de todos os textos em nosso conjunto de dados, pela a remoção de carácteres especiais e referencias jornalísticos, como por exemplo, *["UOL Leia o texto na íntegra"]*. Referências como essas são abundantes nos textos motivadores, e como não acrescentam sentido ao texto, podem ser removidas sem maiores prejuízo. Após limpos, os textos foram devidamente processados de modo a estarem no formato requerido pelo modelo de treinamento.

Os textos que serão processados pelo BERT durante treinamento precisam estar formatados em conformidade com o que o modelo define. Cada segmento de texto precisa ser segmentado em *tokens* e possuir os *tokens* especiais "[CLS]" e "[SEP]" no início e no fim de cada sentença. Os documentos também precisam possuir tamanho fixo, o que só pode ser obtido através do truncamento dos textos ou pela técnica de *padding*; *tokens* que representam palavras reais precisam ser diferenciados desse último tipo e o responsável por realizar essa distinção são as máscaras de atenção.

3.4. Definição de hiper-parâmetros

O modelo BERT utilizado será o modelo BERTimbau (Marinho et al. 2021) em sua versão *base*. A grande maioria dos hiper parâmetros foram definidos com base nas recomendações do artigo oficial do modelo (Devlin et al. 2018). Alguns outros hiper-parâmetros foram definidos a partir de experimentações e também das limitações de nosso ambiente de desenvolvimento. A taxa de aprendizagem, por exemplo, é um hiper parâmetro crítico no treinamento de modelos de aprendizagem profunda. Ela influencia o tamanho dos passos que o modelo dá em direção ao mínimo da função de perda. Os autores do BERT recomendam três taxas aprendizado, e são elas: 5e-5, 3e-5, 2e-5 (Devlin et al. 2018).

Ambos os treinamentos utilizaram a técnica de validação cruzada conhecida como *k-fold validation*. O número de *folds* escolhido foi 10, ou seja, 10-*fold*. O número de épocas para cada *fold* foi 3. Esse valor também segue a recomendação do artigo principal do BERT que salienta que o modelo pré-treinado via de regra não precisa de muitas épocas até uma possível convergência. Para facilitar a visualização dos resultados, apenas as médias das principais métricas de cada tipo de modelo serão apresentadas na presente seção.

3.5. Modelo de regressão

O experimento a seguir, busca explorar e avaliar a eficiência do modelo BERT pré-treinado para a língua portuguesa na tarefa de atribuir notas à redações no formato ENEM

[4] ⟨https://huggingface.co/docs/transformers/index⟩
[5] ⟨https://huggingface.co/⟩
[6] ⟨https://github.com/RaphaelSilv/enem-bertimbau⟩

com base em sua correspondência semântica com seus respectivos textos motivadores. A partir de técnicas de *fine-tunning* o modelo será adaptado para a atribuir um único valor contínuo para redações em uma tarefa de regressão. O treinamento do modelo foi conduzido a partir dos parâmetros definidos na subseção 3.4.

O BERT foi treinado para ser essencialmente um classificador, porém o modelo possibilita sua adaptação para uma série de tarefas derivadas em PLN, do inglês *downstream tasks* (Devlin et al. 2018). Uma dessas tarefas, é a regressão e para configurá-lo para esse fim, basta definir a variável **num_labels** (i.e. número de classes) para 1, quando da instanciação do modelo.

Listing 1. Inicialização do modelo BERT para tarefa de regressão

```
from transformers import AutoModelForSequenceClassification

model_name = 'neuralmind/bert-base-portuguese-cased'
bertimbau = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=1).cuda()
```

Uma vez que os textos foram limpos, pré-processados, e formatados para serem processados pelo modelo, inicia-se o processo de treinamento. A partir das entradas formatadas de cada redação e texto motivador, são extraídos os *embeddings* da última camada do BERT. Em seguida, dois vetores de tamanho 768 são gerados. Esse vetores contém toda informação da entrada textual codificada ao longo das 12 camadas do modelo. Cada entrada também acompanha a nota da redação naquela categoria. A essa altura a nota já foi processada e transformada em tensores do tipo ponto flutuante. Sendo assim, o próximo passo é submeter as entradas ao modelo customizado de rede neural.

A principal customização do nosso modelo, consiste em sua habilidade de correlacionar uma dada redação ao seu respectivo texto motivador. Isso acontece através da concatenação dos dois vetores gerados pelo BERT, que transforma dois vetores de tamanho 768 em único vetor de tamanho 1536. Em seguida, o modelo submete o vetor concatenado a duas camadas densas, que progressivamente reduzem sua dimensionalidade para 512, e 128. Finalmente, a última camada densa reduz a dimensionalidade do vetor de entrada para 1, resultando em sua saída final, um único valor contínuo, a saber a nota estimada.

3.5.1. Resultados

A Tabela 5 e a Tabela 6 trazem a média dos resultados obtidos para as métricas perda *i.e. Loss* e do Erro Médio Absoluto (MAE), durante as etapas de treinamento e validação. Os resultados para diferentes taxas de aprendizado (LR) revelam um padrão. Os valores obtidos tanto para treinamento como para validação estão bem próximos entre si, o que revela consistência por parte do modelo entre as etapas. O *fold* treinado sob a menor taxa de aprendizado, 2e-5, possui o menor valor do MAE, 128,05 e a menor perda de Erro Médio Quadrático (MSE), 17691,05. Na etapa de validação, a performance média dos *folds* treinados sob a mesma taxa de aprendizado foi de 127,19 para o MAE e 17482,00 de perda o MSE. A consistência entre os valores indicam que a etapa de validação está coerente com a etapa de treinamento, inclusive generalizando sutilmente melhor.

3.6. Modelo de classificação

Assim como o modelo de regressão, o presente modelo busca explorar e avaliar a eficiência do BERT pré-treinado para a língua portuguesa na tarefa de atribuição de notas

Tabela 5. Performance de treinamento para diferentes taxas de aprendizado ao longo de 10 *folds*

LR	MSE *Loss*	MAE
5e-5	17710,08	128,13
3e-5	17783,27	128,41
2e-5	17691,05	128,06

Fonte: elaborado pelo autor

Tabela 6. Performance de validação para diferentes taxas de aprendizado ao longo de 10 *folds*

LR	MSE *Loss*	MAE
5e-5	17506,06	127,28
3e-5	17571,78	127,53
2e-5	17482,00	127,19

Fonte: elaborado pelo autor

à redações no formato ENEM. A partir de técnicas de *fine-tunning* o modelo será adaptado para para classificar uma dada redação à uma das 6 possíveis classes de notas, a saber [0 ,40, 80, 120, 160, 200]. O treinamento do modelo foi conduzido a partir dos parâmetros definidos na subseção 3.4. Os detalhes do treinamento são apresentados a seguir.

Similarmente a configuração realizada durante os experimentos com o modelo de regressão, o modelo BERT também precisa ser inicializado. O código 2 apresenta o número de *labels* para o modelo de classificação (i.e., 6).

Listing 2. Inicialização do modelo BERT para tarefa de classificação

```
from transformers import AutoModelForSequenceClassification

model_name = 'neuralmind/bert-base-portuguese-cased'
bertimbau = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=6).cuda()
```

Similarmente ao modelo de regressão, a estratégia de extração dos *embeddings* da última camada BERT é mantida, bem como o método de induzir o modelo a correlacionar redação e texto motivador a partir da concatenação de seus vetores de *embeddings*.

A principal diferença entretanto quanto ao, é que o valor retornado por essa função passa a ser os chamados *logits*. Os *logits* são vetores de valores reais do tamanho do número de *labels*. Cada valor no vetor, representa a confiança do modelo na escolha de uma determinada classe. Quanto maior o valor de um *logit*, maior a confiança do modelo na escolha daquela classe.

Mais tarde, a função de perda, recebe como entrada um conjunto de *logits* e internamente aplica uma função de *softmax*, que converte valores reais em probabilidades antes do calculo de perda. t*Logits* são vetores de valores reais do tamanho do número de *labels*. Cada valor no vetor, representa a confiança do modelo na escolha de uma determinada classe. Quanto maior o valor de um *logit*, maior a confiança do modelo na escolha daquela classe.

3.6.1. Resultados

Os resultados obtidos para a etapa de treinamento são bastante positivos para todas as taxas de aprendizado. Os valores de acurácia e F1 *score* estão sempre próximos, o que sugere um balanceamento entre a precisão e a sensibilidade (*Recall*). Além disso, a maioria de seus valores estão consistentemente acima de 70%. Os valores de perda, entretanto são significativamente altos.

De modo geral, os resultados sugerem que o melhor desempenho é do modelo treinado sob a taxa de aprendizado em 5e-5. O modelo apresenta o menor valor de perda, 0,63 e os valores de acurácia e F1 score, também são os melhores atingindo 80,81 e 0,79, respectivamente. O segundo melhor resultado, ainda na etapa de treinamento, é do *fold* treinado sob a taxa 3e-5. O valor de perda observado foi 0,72, apenas 0,09 acima do valor encontrado no primeiro resultado e 0,10 abaixo do valor encontrado no *fold* treinado sob a taxa 2e-5.

Tabela 7. Performance de treinamento para diferentes taxas de aprendizado ao longo de 10 *folds*

LR	Avg. Acurácia	Avg. F1-score	Avg. Loss
5e-5	80,81	0,79	0,63
3e-5	76,27	0,73	0,72
2e-5	71,68	0,68	0,82

Fonte: elaborado pelo autor

Tabela 8. Performance de validação para diferentes taxas de aprendizado ao longo de 10 *folds*

LR	Avg. Acurácia	Avg. F1-score	Avg. Loss
5e-5	81,73	0,80	0,58
3e-5	78,71	0,75	0,66
2e-5	72,28	0,69	0,80

Fonte: elaborado pelo autor

4. Análise dos resultados

Quanto ao modelo de classificação, percebe-se uma tendência, na medida em que a taxa de aprendizado sobe, acurácias e F1 score também sobem, e o valor de perda decresce. Isto pode implicar que o modelo se beneficia de maiores atualizações nos pesos com esta taxa de aprendizagem. É como se o modelo fosse capaz de aprender mais eficazmente com taxas mais altas capturando melhor as nuances semânticas dos textos.

Para todas as métricas nota-se que geralmente, os conjuntos de validação possuem desempenho levemente superior ao o seu respectivo conjunto de treinamento. Isso sugere boa capacidade de generalização do modelo para dados não vistos.

O modelo de regressão entretanto, revela um desempenho no geral insatisfatório. Os menores valores do MAE encontrados durante treinamento e validação, 128,06 e 127,19, respectivamente, indicam que o erro médio das predições do modelo podem estar até 128 pontos acima ou abaixo da nota real, que varia de 0 à 200.

Os motivos por trás da baixa performance do modelo de regressão podem estar relacionadas à diferentes fatores, entre estes destacam-se: a possível baixa qualidade dos dados, baixa representação de *features*, ou seja, conjunto de dados desbalanceado, ou ainda a própria arquitetura do modelo.

5. Considerações finais

O presente trabalho utilizou o modelo de linguagem BERT*imbau*, adptação do BERT pré-treinado para a língua portuguesa do Brasil, para representar semanticamente redações e textos motivadores a partir de de *embeddings* contextualizados de palavras. Para atingir os objetivos do trabalho, dois modelos treinados a partir de técnicas de Aprendizado Profundo foram criados: um de regressão e outro de classificação. Os resultados sugerem boa adaptção do modelo de regressão e performance no geral insatisfatória para o modelo de regressão. Ao fim dos experimentos, o presente trabalho disponibiliza duas funções capazes de aferir ou classificar o desempenho de uma dada redação e seu respectivo texto motivador obteve dentro dos limites avaliativos da segunda competência da redação do ENEM.

Entre as limitações encontradas no desenvolvimento dos trabalhos, destacam-se a baixa qualidade ortográfica em algumas redações do conjunto de dados. É possível que a estratégia de extração e limpeza utilizadas pelos autores do *dataset*, corrompeu algumas palavras com caracteres especiais. Este é um fator relevante, pois palavras não compreendidas pelo modelo são ignoradas o que implica em perda de informação. Logo, alguns textos certamente tiveram suas representações comprometidas por essa razão. Outro fator limitante, foi a restrição do modelo BERT em processar documentos com no máximo 512 *tokens*. Por essa razão, apenas cerca de 40% do conjunto de dados.

Os potenciais trabalhos futuros identificados, são: *i)* utilizar técnicas eficientes para lidar com extração de *embeddings* de textos com mais de 512 *tokens* enriquecendo o conjunto de treinamento; *ii)* aprimorar a etapa de limpeza de dados e extração de informações não-relevantes; *iii)* garantir a qualidade gramatical dos textos, uma vez que erros ortográficos implicarão em uma representação de *embeddings* empobrecida; *iv)* realizar o *fine-tuning* dos modelos utilizando diferentes combinações de hiper-parâmetros, especialmente no contexto de mais dados sendo processados.

Referências

aaaa.

Baeza-Yates, R. and Ribeiro-Neto, B. (2013). *Recuperação de Informação - 2ed: Conceitos e Tecnologia das Máquinas de Busca*. Bookman Editora.

Bertucci, R. A. (2021). Propriedades linguísticas da redação do enem: uma análise computacional. *REVISTA DE ESTUDOS DA LINGUAGEM*, 1(1):1–31.

Braz, O. O. and Fileto, R. (2021). Investigando coerência em postagens de um fórum de dúvidas em ambiente virtual de aprendizagem com o bert. In *Anais do XXXII Simpósio Brasileiro de Informática na Educação*, pages 749–759. SBC.

Cavalcanti, A., Mello, R., Miranda, P., and Freitas, F. (2020a). In *Anais do XXXI Simpósio Brasileiro de Informática na Educação*, pages 892–901, Porto Alegre, RS, Brasil. SBC.

Cavalcanti, A. P., Rafael Mello, F. L. d., and Miranda, Pericles Barbosa Cunha de and, F. L. G. d. F. (2020b). Análise automática de feedback em ambientes de aprendizagem online. *IX Congresso Brasileiro de Informática na Educação*, 1(1).

David Reinsel, John Gantz, J. R. (2022). The digitization of the world: From edge to core.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

INEP (2020). Censo da educaÇÃo superior 2020. *Ministério Da Educação*, 1(1).

INEP (2022). A redaÇÃo do enem 2022 cartilha do participante.

Marinho, J., Anchiêta, R., and Moura, R. (2021). Essay-br: a brazilian corpus of essays. In *Anais do III Dataset Showcase Workshop*, pages 53–64, Online. Sociedade Brasileira de Computação.

McCormick, C. (2023). *The Inner Workings of BERT*, volume 1 of *1*. Chris McCormick, 1 edition.

Oliveira, D. C. d., Pozzebon, E., and Santos, T. N. d. (2020). Aplicação de técnicas de processamento de linguagem natural na automatização de correção de questões discursivas. *IX Congresso Brasileiro de Informática na Educação*, 1(1).

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Sorato, D., Goularte, F. B., and Fileto, R. (2020). Short semantic patterns: A linguistic pattern mining approach for content analysis applied to hate speech. *International Journal on Artificial Intelligence Tools*, 29(02):2040002.

Souza, F., Nogueira, R., and Lotufo, R. (2020). Bertimbau: pretrained bert models for brazilian portuguese. In *Brazilian conference on intelligent systems*, pages 403–417. Springer.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, W., Yan, M., and Wu, C. (2018). Multi-granularity hierarchical attention fusion networks for reading comprehension and question answering. *arXiv preprint arXiv:1811.11934*.

Wirth, R. and Hipp, J. (2000). Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, volume 1, pages 29–39. Manchester.