INSTITUTO FEDERAL DE SANTA CATARINA

JOÃO LEONARDO MARTINS

Aplicação web para análise de sentimentos utilizando a tecnologia *Transformers*

São José - SC

Julho/2023

APLICAÇÃO WEB PARA ANÁLISE DE SENTIMENTOS UTILIZANDO A TECNOLOGIA *TRANSFORMERS*

Monografia apresentada ao Curso de Engenharia de Telecomunicações do campus São José do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheiro de Telecomunicações.

Orientador: Prof. Mário de Noronha Neto, Dr.

São José - SC

Julho/2023

CDD 410.028
M386a

Martins, João Leonardo
Aplicação web para análise de sentimentos utilizando a tecnologia Transformers [TCC] / João Leonardo Martins; orientação de Mário de Noronha Neto. – São José, 2023.

1 v. : il.

Trabalho de Conclusão de Curso Superior (Engenharia de Telecomunicações) – Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Inclui referências.

1. Processamento de linguagem natural. 2. Transformers. 3. Generative pretrained transformer. 4. Aplicação web.

Sistema de Bibliotecas Integradas do IFSC
Biblioteca IFSC Campus São José
Catalogado por: Kênia Raupp Coutinho CRB14/951

João Leonardo Martins

Aplicação web para análise de sentimentos utilizando a tecnologia *Transformers*

Este trabalho foi julgado adequado para obtenção do título de Engenheiro de Telecomunicações, pelo Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, e aprovado na sua forma final pela comissão avaliadora abaixo indicada.

São José - SC, 7 de julho de 2023:

IFSC
Documento assinado digitalmente
MARIO DE NORONHA NETO
Data: 24/07/2023 17:03:17-0300
CPF: 003.859.519-22
Verifique as assinaturas em https://v.ifsc.edu.br

Prof. Mário de Noronha Neto, Dr.
Orientador
Instituto Federal de Santa Catarina

gov.br
Documento assinado digitalmente
ERALDO SILVEIRA E SILVA
Data: 24/07/2023 21:52:57-0300
Verifique em https://validar.iti.gov.br

Prof. Eraldo Silveira e Silva, Dr.
Instituto Federal de Santa Catarina

gov.br
Documento assinado digitalmente
ROBERTO WANDERLEY DA NOBREGA
Data: 25/07/2023 12:52:15-0300
Verifique em https://validar.iti.gov.br

Prof. Roberto Wanderley da Nóbrega, Dr.
Instituto Federal de Santa Catarina

Este trabalho é dedicado à minha querida avó Natalina (in memoriam).

AGRADECIMENTOS

Agradeço primeiramente a Deus.

Aos meus pais, Leonardo e Sueli, que não mediram esforços para que este sonho se tornasse possível.

Àqueles que lutam e fazem o IFSC ser sinônimo de educação pública de qualidade, em especial ao meu orientador, Professor Noronha.

RESUMO

O emprego de técnicas de Inteligência Artificial tem permitido avanços muito relevantes em nosso cotidiano, nos mais diversos setores: desde assistentes virtuais, passando por anúncios e *feeds* personalizados em redes sociais, até chegar nos atendentes virtuais ou *chatbots*, como também são conhecidos. Muito disso se dá pela compreensão profunda das intenções de buscas e interações dos usuários, analisado atentamente pelos algoritmos utilizados por empresas, graças ao uso de técnicas de Processamento de Linguagem Natural. Neste trabalho busca-se entender os conceitos estudados nos principais ramos de Inteligência Artificial relacionados ao Processamento de Linguagem Natural, com enfoque no funcionamento em modelos baseados em *Transformers*, como o *Generative Pre-trained Transformer*, da empresa OpenAI. Por fim, uma Interface de Programação de Aplicação será desenvolvida para que possamos observar o modelo treinado na prática. Com isso será possível, através de requisições via web, classificar o sentimento por trás de frases em português, as identificando como negativas, neutras ou positivas.

Palavras-chave: Processamento de Linguagem Natural. *Transformers*. *Generative Pre-trained Transformer*. Aplicação Web.

ABSTRACT

The use of Artificial Intelligence techniques has enabled significant advancements in our daily lives across various sectors: virtual assistants, personalized ads, social media feeds, and virtual attendants, also known as chatbots. Much of this relies on a deep understanding of user search intentions and interactions, carefully analyzed by algorithms employed by companies, thanks to the use of Natural Language Processing techniques. In this work, our aim is to understand the concepts studied in the main topics of Artificial Intelligence related to Natural Language Processing, with a focus on transformer-based models, such as the Generative Pre-trained Transformer developed by OpenAI. To observe these concepts in practice, an Application Programming Interface will be developed to allow the trained model's usage, enabling it to classify the sentiment behind Portuguese sentences as negative, neutral, or positive.

Keywords: Natural Language Processing. Transformer. Generative Pre-trained Transformer. Web Application.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

AI *Artificial Intelligence.*

API *Application Programming Interface.*

BERT *Bidirectional Encoder Representations from Transformers.*

CNN *Convolutional Neural Networks.*

CPU *Central Processing Unit.*

DL *Deep Learning.*

DTMC *Discrete-Time Markov Chains.*

GPT *Generative Pre-Trained Transformer.*

GPT-2 *Generative Pre-Trained Transformer-2.*

GPT-3 *Generative Pre-Trained Transformer-3.*

GPU *Graphics Processing Unit.*

HTML *HyperText Markup Language.*

HTTP *Hypertext Transfer Protocol .*

IA Inteligência Artificial.

IFSC Instituto Federal de Santa Catarina.

IoT *Internet of Things.*

JSON *JavaScript Object Notation.*

LSTM *Long Short-Term Memory.*

ML *Machine Learning.*

NLP *Natural Language Processing.*

PLN Processamento de Linguagem Natural.

REST *Representational State Transfer.*

RNN *Recurrent Neural Networks.*

SPAM *Sending and Posting Advertisement in Mass.*

TTS *Text to Speech.*

SUMÁRIO

1 INTRODUÇÃO

Recentemente pudemos acompanhar avanços importantes em Inteligência Artificial (IA), cada vez mais presente no nosso cotidiano, seja em algoritmos de busca, detecção de doenças ou em atendimentos de empresas, através de *chatbots* (ALBAYRAK; ÖZDEMIR; ZEYDAN, 2018). Com a parte econômica de diversas empresas prejudicada pela pandemia, surge a necessidade de investimentos pontuais que impliquem num maior retorno com a menor contrapartida possível. É aí que entra a IA, com ferramentas que as auxiliam a fazer publicidade de forma mais eficiente, impactando um público mais específico e com grande probabilidade de retorno (JAREK; MAZUREK, 2019).

Sempre quando há um avanço tecnológico que possibilite que uma tarefa passe a ser executada por uma máquina, há um receio geral de que pessoas perderão seus empregos. Porém, o que é possível observar é que, num primeiro momento, os frutos da IA são utilizados como auxiliares, como por exemplo, em um pré-atendimento, seja por web chat ou através da telefonia convencional. Isso se traduz em produtividade que, por sua vez, possibilita a implantação de uma estratégia em voga: o conceito de *Omnichannel*, que é, basicamente, a convergência de todos os canais utilizados por uma empresa (SCHROTENBOER, 2019).

Tão presentes em nosso cotidiano que um deles é utilizado como verbo, os buscadores de internet também se valem, e muito, de IA. Por exemplo, ao acessar os mecanismos de buscas do Google, estamos em contato com um algoritmo chamado *Bidirectional Encoder Representations from Transformers* (BERT), que é um dos modelos mais modernos em Processamento de Linguagem Natural (PLN) e que permite uma busca muito mais refinada, com resultados muito mais próximos ao que o usuário realmente se refere (daí o conceito de Processamento de Linguagem Natural). O autopreenchimento em buscas é mais um recurso possível graças ao emprego do BERT (DEVLIN et al., 2018).

Lançado no fim de 2022, o ChatGPT, popular *chatbot* da OpenAI, bateu recorde como o aplicativo com crescimento mais rápido da história, registrando 100 milhões de usuários ativos mensais em janeiro de 2023 (FORBES, 2023). Com isso, tomaram conta dos debates a respeito de IA temas como ética e, novamente, profissões que podem se tornar obsoletas com o aperfeiçoamento dessa ferramenta (LUND; WANG, 2023).

Outro conceito interessante que observamos nos buscadores, porém não apenas nos mesmos, é o de Visão Computacional, que é o ramo de IA que estuda o processamento de imagens pelos computadores. Com isso, é possível realizarmos uma busca a partir de uma figura, garantir a entrada de pessoas através de reconhecimento facial, prevenção de acidentes através da inspeção de construções, leitura de placas de forma automatizada,

entre outras aplicações (MARTINEZ; AL-HUSSEIN; AHMAD, 2019).

Sobre PLN, como mencionado, houve um avanço bem interessante desde o início da pandemia de Covid-19, em que muitas empresas buscaram formas de automatizar o seu atendimento, fomentando o setor (ZAGO, 2021). Além do emprego em atendimento ao cliente e buscas, como já citado, o PLN pode ser aplicado em tradução de idiomas, filtro de *Sending and Posting Advertisement in Mass* (SPAM), reconhecimento de fala, aplicativos assistentes pessoais, entre outros. Tudo isso teve início na década de 1940, quando surge uma maior necessidade, com os aprendizados da Segunda Guerra Mundial, de máquinas que pudessem traduzir de uma língua para outra rapidamente.

Ainda que o conceito de PLN esteja se popularizando e cada vez mais presente em nossas vidas, há muitas possibilidades de aperfeiçoamento e implementação dessa tecnologia, que cada vez mais vem atuando como uma interface entre máquinas e humanos. Assim sendo, gradativamente iremos observar avanços nessa área, seja no aperfeiçoamento do entendimento das solicitações de usuários em buscadores, por exemplo, como também numa maior integração com outros segmentos de IA, além de outras áreas, como *Internet of Things* (IoT).

1.1 Objetivo geral

Este trabalho objetiva o desenvolvimento de uma interface web para que seja possível realizar consultas, através de um modelo *Generative Pre-Trained Transformer* (GPT), que retornem qual o sentimento presente na frase submetida. Além disso, o entendimento básico dos modelos modernos para PLN, assim como suas interações com outros campos de IA e seus impactos na sociedade também será buscado.

Será abordada com maior detalhamento a técnica GPT e, como resultado, espera-se obter uma ferramenta, baseada nesse modelo, que nos dê um embasamento para determinar se as pessoas estão gostando de determinada experiência (que pode ser um serviço ou um produto). Tudo isso com base num treinamento prévio e uma base de dados de avaliações dos mesmos.

1.2 Objetivos específicos

Alguns pontos foram pensados como meta para este trabalho, de modo a atingir o entendimento buscado de forma geral:

1. Obtenção de uma base de dados, em português, para refinamento do modelo
2. Preparação da base de dados em formato legível pela linguagem de programação e modelo escolhido

3. Escolha de um modelo pré-treinado em português
4. *Fine-tuning* ou refinamento do modelo escolhido para análise de sentimentos em português
5. Avaliação do desempenho do modelo refinado para a tarefa de análise de sentimentos
6. Desenvolvimento de uma web API que simplifique a utilização do modelo
7. Desenvolvimento de uma interface web para que usuários finais também possam realizar consultas

2 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo 2 serão abordados, com maiores detalhes, os assuntos pertinentes ao desenvolvimento deste trabalho. Foram realizadas revisões bibliográficas para conceitos como Inteligência Artificial (IA), de uma forma geral, bem como *Natural Language Processing* (NLP) e *Transformers*.

2.1 Fundamentos gerais de IA

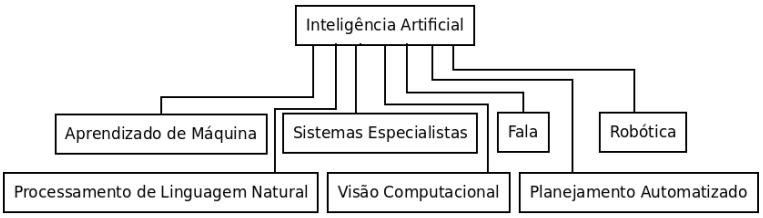
Inteligência Artificial (IA) é uma parte da Ciência da Computação que, principalmente, trabalha com a questão da automação do comportamento inteligente, seja do domínio humano, animal ou vegetal, conforme Chowdhary (2020). Nesse contexto, comportamento inteligente é definido pelo autor como uma combinação de percepção, análise e reação. Na prática, isso tudo implica em ricas experiências para os usuários de assistentes virtuais (como a Siri e a Alexa), carros autônomos ou semiautomatizados, mecanismos de buscas e tantos outros que diariamente temos contato.

É válido afirmar que essas diversas facilidades são possíveis graças aos recursos de IA, que permitem aos sistemas aprenderem através de experiências e memorizá-las, sendo capazes de tomarem decisões com base num histórico e até mesmo preverem situações. Além do mais, apresentam traços de inteligência humana, através de interação por linguagem natural. São resultados da combinação de avanços de hardware e software, por meio de evolução do poder de processamento e algoritmos, respectivamente (XAVIER, 2020).

De acordo com Villanueva e Salenga (2018), há sete áreas de estudo em IA, conforme a Figura 1. Cada subconjunto, combinados ou não, permitem o desenvolvimento de facilidades citadas nesta seção e inúmeras outras. Exemplificando o produto de algumas das principais áreas de Inteligência Artificial (IA), podemos citar para o domínio de Aprendizado de Máquina os alertas de tráfego, muito comuns nos aplicativos de mapas. Para o âmbito de Processamento de Linguagem Natural, uma forma de emprego é para análise de sentimento, como veremos mais adiante. Já para Sistemas Especialistas, através de detectores de vírus.

Além dos campos de IA mencionados, há também a possibilidade de divisão sob outra perspectiva, segundo a abordagem de Searle (1980), que são os conceitos de IA Geral e IA Estreita, ou IA Forte e IA Fraca, respectivamente. A primeira traz a ideia de uma Inteligência Artificial (IA) capaz de raciocinar de fato, atuando como uma mente propriamente dita. Já a Inteligência Artificial (IA) Fraca é caracterizada como uma ferramenta para resolver problemas pontuais. Naturalmente, a segunda forma está muito

Figura 1 – Áreas de estudo de IA



Fonte: O próprio autor

mais desenvolvida, sendo possível observá-la através das aplicações citadas anteriormente, enquanto a primeira, por se tratar de um ramo que visa dar consciência às máquinas, ainda é bastante prematura e incerta.

2.2 Aprendizado de Máquina

Machine Learning (ML) em inglês é o segmento de IA que se busca programar computadores, não necessariamente de uma forma explícita, para que possam aprender como seres humanos, mas como não são capazes de raciocinar, é correto afirmar que o aprendizado vem de treinamentos. Portanto, esse conhecimento não vem apenas do armazenamento de (muitas vezes) grandes quantidades de dados, mas sim da extração de padrões de variadas formas de fontes de dados, como, por exemplo, vídeos, interações de redes sociais, falas de seres humanos, dados climáticos, entre outros (KAPLAN, 2016).

De acordo com Wakefield (2019) e Bernard (2021), há quatro paradigmas de aprendizagem principais:

1. **Aprendizagem supervisionada**: esse modelo é o mais aplicado atualmente. Em suma, a máquina aprende através de exemplos rotulados, em que as entradas e saídas são conhecidas. Com isso, o algoritmo deve encontrar uma forma de como chegar nos resultados de acordo com os dados fornecidos. Seu nome se dá pelo fato de que, enquanto o código não obtém um nível satisfatório de acerto em suas tarefas, o operador atua de modo a refiná-lo. Pode ser utilizado para realizar **classificações**, cujo objetivo é definir qual categoria corresponde às novas ocorrências (com base num conjunto de dados) ou então para **regressões**, que buscam traduzir em números alguma questão, portanto, seu trabalho é prever um valor numérico;

2. **Aprendizagem não supervisionada**: apesar de não ser tão utilizado quanto a supervisionada, a aprendizagem não supervisionada é considerada pelos autores como chave para o desenvolvimento de sistemas artificialmente inteligentes. Este modelo é utilizado em situações mais específicas e funciona, basicamente, analisando conjuntos de dados e identificando padrões, buscando relações entre eles de modo a

catalogá-los. Isso é chamado de **agrupamento** e é utilizado para separar indivíduos por seus gostos para utilização em campanhas de marketing, por exemplo. Pode ser utilizada também para **redução de dimensionalidade**, que é uma técnica de aprendizagem não supervisionada empregada para restrição de variáveis dentro de bases de dados, preservando o que realmente importa. Pode ser observada em sistemas de recomendações;

3. **Treinamento por reforço**: trata-se de um modelo de aprendizagem consideravelmente diferente dos anteriores, tendo em vista que a aquisição dos conjuntos de dados não ocorre manualmente através de entradas convencionais, mas sim através de interações com outros sistemas, chamados de **ambiente** neste contexto. Simplificando, pode-se dizer que a máquina aprende através de tentativa e erro, capacitando-se através de experiências passadas e se adaptando através dos resultados obtidos. Uma aplicação deste modelo pode ser observada em alguns data centers da empresa Google, em que o sistema de arrefecimento é controlado por algoritmos treinados por reforço, e os resultados são positivos, tendo em vista que o consumo energético foi otimizado;

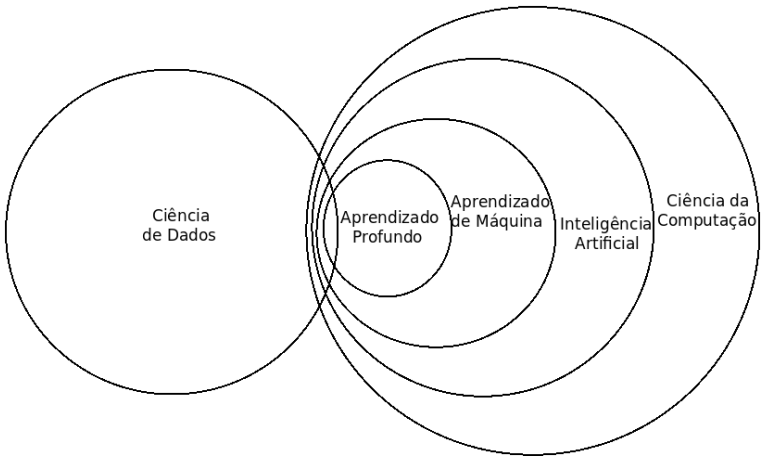
4. **Treinamento por transferência**: é uma forma de aprendizagem que busca associar um conhecimento já adquirido em determinada tarefa em outra, facilitando a absorção do conteúdo. É similar ao que os seres humanos fazem o tempo todo: nós reutilizamos nossos saberes de modo a acelerar novos afazeres. Em Inteligência Artificial (IA), treinamento por transferência é utilizado no ramo de Visão Computacional - através de comparações de imagens e no campo de Processamento de Linguagem Natural (PLN), sintetizando textos, por exemplo.

2.2.1 Aprendizado Profundo

Em inglês *Deep Learning* (DL), pode ser definido como uma técnica para implementação de *Machine Learning* (ML), técnica essa que permite a utilização em larga escala de *Machine Learning* (ML) através de seus algoritmos, que dividem tarefas de formas simples. Isso tudo é possível graças às redes neurais, que formam diversas camadas de processamento e possibilitam o aprendizado através de um grande volume de dados. São aplicações de *Deep Learning* (DL) os campos de Visão Computacional e PLN. Um exemplo clássico de uma aplicação prática proporcionada por DL foram vídeos de gatos no YouTube sendo classificados e sugeridos aos usuários de forma automática (ONGSULEE, 2017).

Conforme a Figura 2, *Deep Learning* (DL) é uma das formas possíveis de execução de *Machine Learning* (ML), que por sua vez é uma abordagem para chegar à *Artificial Intelligence* (AI).

Figura 2 – Relação de Aprendizado Profundo



Fonte: O próprio autor

2.2.1.1 Redes Neurais

Conforme adiantado na seção 2.2.1, as redes neurais artificiais possibilitam a implementação de sistemas artificialmente inteligentes e são inspiradas no cérebro humano. Ou melhor, no modo em que são compostos, através de neurônios, e organizados de modo a formar uma rede. De maneira semelhante, os neurônios artificiais se comunicam, permitindo o processamento de um volume grande de informação (BERNARD, 2021).

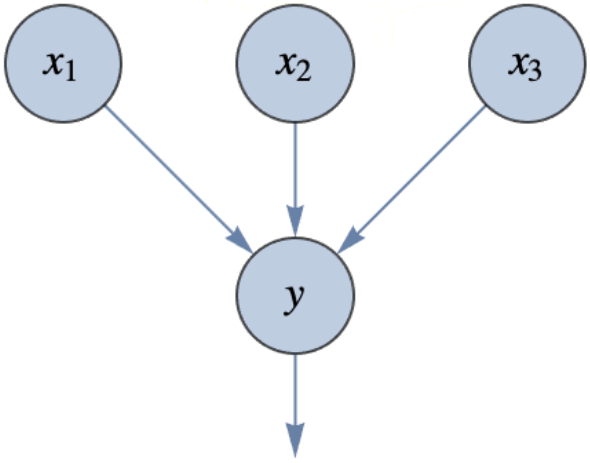
A Figura 3 representa, sinteticamente, a forma mais básica das redes neurais, que são chamadas de **unidades**, ou unidades de processamento. Nela podemos observar, denominados por x, os sinais de entrada aplicados à célula artificial, que, após cálculos envolvendo uma soma ponderada, de acordo com a influência dos sinais, é apresentada uma saída y. (BERNARD, 2021). Outra nomenclatura comum é *Perceptron*, de acordo com o desenvolvedor Frank Rosenblatt, o cientista responsável e que foi inspirado pelos trabalhos anteriores de Warren McCulloch e Walter Pitts. A invenção ocorreu nas décadas de 1950 e 1960 (ACADEMY, 2019).

De forma didática, alguns elementos foram suprimidos na Figura 3, porém, matematicamente, um neurônio artificial pode ser representado pela Equação 2.1, em que a saída y é o resultado da função não linear f, dada pela multiplicação de parâmetros w - que podem ser aprendidos - e são chamados de **pesos**, multiplicados pelas entradas. É somado a isso tudo o parâmetro b, chamado de **bias**, ou viés, que pode restringir ou aumentar os pesos.

$$y = f(w1x1 + w2x2 + w3x3 + b) \tag{2.1}$$

A partir do agrupamento de neurônios artificiais em rede é possível chegarmos às **redes neurais**, que, em suma, se dá pela conexão de muitos desses neurônios. Essa

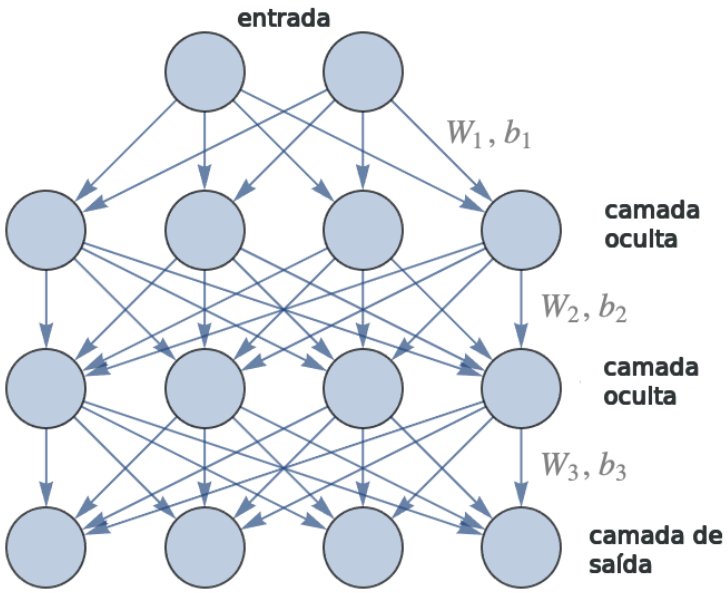
Figura 3 – Representação de um Neurônio Artificial



Fonte: Adaptada de Bernard (2021)

ligação entre eles não é feita de forma randômica e utiliza uma arquitetura clássica, que pode ser definida com a essência de um modelo de DL, conforme Bengio (2016), que é a ***feed-forward neural network*** ou rede neural de alimentação para frente, caracterizada por ser dividida em níveis, com neurônios artificiais ligados às camadas imediatamente anterior e superiores (e não às próprias ou adjacentes) e possuir N camadas intermediárias. Uma ilustração pode ser observada na Figura 4.

Figura 4 – Representação clássica de uma Rede Neural



Fonte: Adaptada de Bernard (2021)

Com a interligação de milhões de neurônios artificiais, inúmeras camadas ocultas de redes neurais podem ser formadas, dando origem às **redes neurais profundas**. Daí

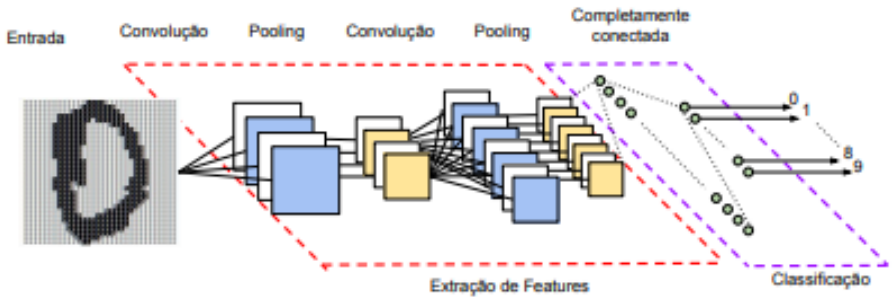
o nome *Deep Learning* (DL), em que a palavra *deep*, ou profundo, se refere ao fato da utilização de redes neurais com o mesmo nome: *deep neural networks* (BERNARD, 2021).

2.2.1.2 Redes Neurais Convolucionais

É uma das arquiteturas de redes neurais profundas mais clássicas. Também conhecida por *Convolutional Neural Networks* (CNN), é uma variação da rede neural de alimentação para frente, vista na seção 2.2.1.1. Difere-se no sentido de que, como o nome sugere, utiliza a operação matemática **convolução**. Para tanto, ao menos uma camada de suas redes utiliza essa operação no lugar de multiplicação. É especialmente utilizada no campo de Visão Computacional, podendo ser empregada em agrupamento de imagens, busca por fotos ou reconhecimento de objetos, por exemplo. (ZEILER; FERGUS, 2014).

A Figura 5 representa as etapas que compõem uma CNN, cabendo a cada uma dessas camadas desempenhar uma função na propagação do sinal. São elas: **convolução**, responsável pela definição dos atributos de entrada através de filtragens realizadas por neurônios, que resultam em um conjunto de pixels encaminhados ao nível seguinte; ***pooling*** ou agrupamento, que simplifica o que foi obtido anteriormente, reduzindo o número de amostras; já a camada **completamente conectada** tem como objetivo ligar os neurônios de *pooling* à saída, classificando o resultado de acordo com as respostas dos filtros das camadas anteriores.

Figura 5 – Arquitetura de uma Rede Neural Convolucional



Fonte: Vargas, Paes e Vasconcelos (2016)

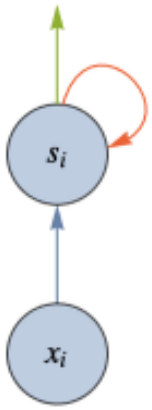
2.2.1.3 Redes Neurais Recorrentes

As *Recurrent Neural Networks* (RNN), por sua vez, são amplamente em PLN, processando informações sequenciais como, por exemplo, texto e áudio. Da mesma forma que as CNN, as RNN também aprendem através de treinamento, porém com um diferencial de utilizar informações passadas para tomar decisões, ou seja, possui memória (BENGIO, 2016).

Conforme (BERNARD, 2021), uma Rede Neural Recorrente pode ser simplificada como um módulo processando dados sequenciais enquanto mantém e atualiza um estado,

estado esse que, como citado, poderá ser reutilizado. A Figura 6 ilustra essa forma mais básica de Rede Recorrente, em que x e s, que são, respectivamente, vetores numéricos da informação sequencial de entrada e do estado, respectivamente.

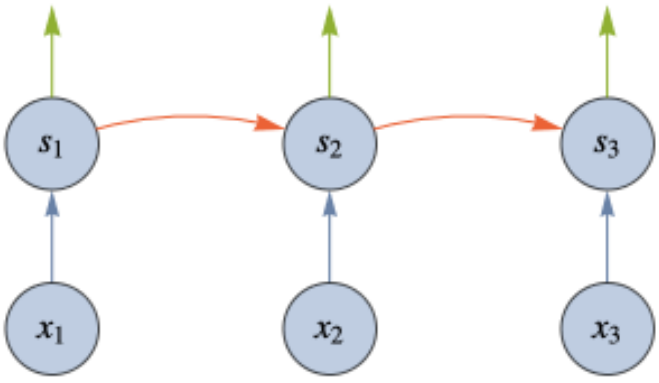
Figura 6 – Camada de recorrência



Fonte: Bernard (2021)

Já a Figura 7 exibe os desdobramentos para uma sequência exemplificada com comprimento três, deixando claro a característica de influência do estado anterior no cômputo do estado atual.

Figura 7 – Rede de recorrência



Fonte: Bernard (2021)

2.2.1.4 *Long Short-Term Memory* (LSTM)

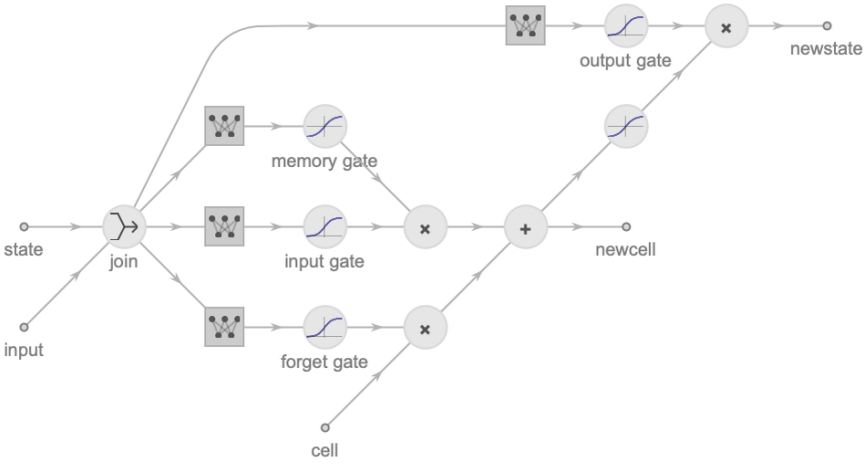
De acordo com Graves (2012), as redes do tipo *Long Short-Term Memory* (LSTM) vieram para superar as limitações das RNN, contornando as perdas de influência que uma determinada entrada sofre em grandes sequências de dados. Simplificando, são capazes de aprender conexões de longo prazo. Tudo isso é possível através de **portas**, ou *gates*,

em inglês, que possuem a capacidade de definir o que será armazenado ou eliminado das memórias internas.

Na Figura 8 temos detalhes da estrutura da rede LSTM, formadas pelas seguintes portas: ***forget gate***, ***input gate***, ***memory gate*** e ***output gate***, pelos vetores numéricos de entrada e saída, ***input*** e ***output*** e pelos vetores de estado ***state*** e ***cell state***, sendo esse último utilizado para levar informação a etapas posteriores. Seu funcionamento ocorre da seguinte forma:

- A porta *forget*, multiplicando *cell*, é capaz de definir o que deve ser levado em conta e o que deve ser esquecido da memória de longo prazo;
- *Input gate*, é multiplicada pela *memory gate*, e posteriormente somada à porta de estado *cell*. Ou seja, *memory gate* calcula o que deve ser adicionado à *cell state* e *input gate* decide se de fato essa adição deve ocorrer;
- *output gate* define quais valores de *cell state* devem ser levados ao próximo estado.

Figura 8 – Núcleo da rede da arquitetura LSTM



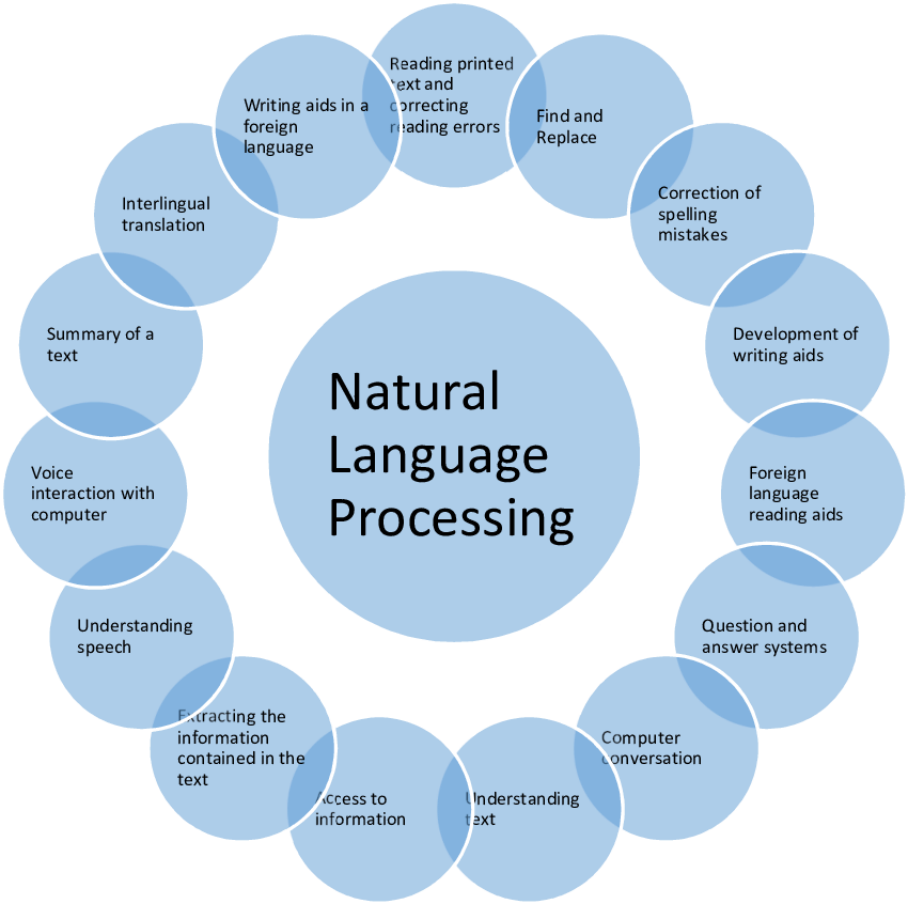
Fonte: Bernard (2021)

2.3 Processamento de Linguagem Natural (PLN)

Em inglês *Natural Language Processing* (NLP), é o ramo de IA que trata de capacitar as máquinas para compreenderem o processo natural de comunicação dos seres humanos, não apenas realizando traduções livres, mas sim observando suas nuances nos mais variados contextos (EISENSTEIN, 2019).

Aplicações como *Text to Speech* (TTS), Análise de Sentimentos, Assistentes Virtuais entre outras diversas, conforme a Figura 9, são viáveis através do PLN.

Figura 9 – Tópicos de PLN



Fonte: SÜTÇÜ e AYTEKİN (2019)

De acordo com Liddy (2001), há quatro tipos de abordagens, que exigem, naturalmente, diferentes níveis de processamento:

- **Simbólica**, que é baseada em regras de linguagem bem limitadas, ou seja, que não tenham ambiguidade. Algoritmos são construídos de modo a realizar classificações simples de palavras, uma a uma. Há uma forte dependência em conceitos de linguística, como gramática, por exemplo, que pode implicar em limitações por ambiguidade;

- A **estatística** baseia-se em sistemas supervisionados, conforme seção 2.2, e aplica modelos matemáticos, como, por exemplo, cadeias de Markov, de modo que as relações probabilísticas entre as palavras são calculadas através de *Discrete-Time Markov Chains* (DTMC). Basicamente a relação entre palavras ou elementos linguísticos é dada pela probabilidade de transição e seus estados atuais, permitindo, assim, que novas sentenças sejam geradas ou então relações entre elas sejam estabelecidas, dependendo da aplicação. Diferente da simbólica, a abordagem estatística é capaz de realizar deduções de interpretação, dispensando assim regras linguísticas;

- Já a abordagem **conexionista** é similar à estatística, com o diferencial de se valer da junção de teorias de conhecimento com aprendizado estatístico, dando maior liberdade para dedução, transformação e manipulação de textos. Isso se torna possível através do uso de Redes Neurais, conforme visto na subseção 2.2.1.1, em que as informações são paralelamente processadas por neurônios artificiais;

- Por último, a **híbrida**, como o nome sugere, combina todas as anteriores, dando um grau ainda maior de flexibilidade para resolução de problemas de PLN.

2.4 *Transformers*

Transformer é uma arquitetura de rede neural baseada no mecanismo de **atenção**. Fazendo um paralelo com a mente humana, que é capaz de focar em informações relevantes e ignorar o que não lhe convém, esse modelo concentra-se nas partes mais importantes dos dados de entrada, o que permite um melhor estabelecimento de relações entre palavras de diferentes sequências (HAN et al., 2021). Este modelo de ML surge como uma excelente alternativa às redes neurais recorrentes estudadas na subseção 2.2.1.3, que possuem limitações, como, por exemplo, falta de efetividade em situações que a entrada e saída sejam sequencialmente distantes. Como a arquitetura *Transformer* utiliza uma estrutura do tipo *encoder-decoder*, há a possibilidade de uma paralelização em seu treinamento, se tornando muito mais efetivo para as tarefas de PLN (VASWANI et al., 2017).

De acordo com Markowitz (2021), *Transformer* é sustentado por três pilares:

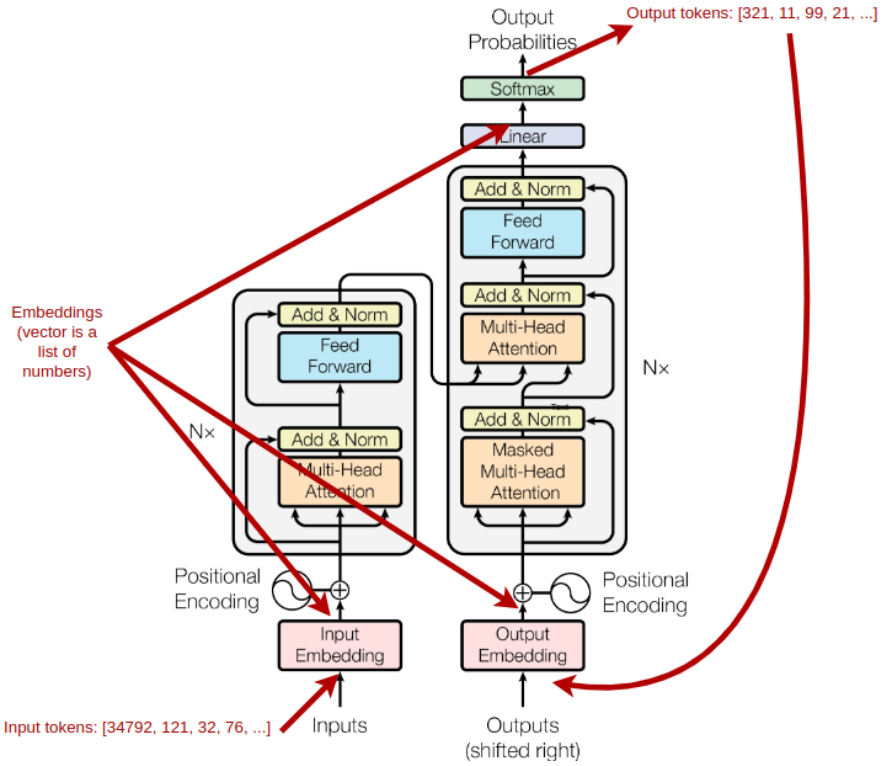
- ***Attention***, ou atenção, que permite estabelecer valores ou pesos aos elementos de diferentes entradas baseado na relevância entre eles. Isso ocorre após o modelo ser submetido a um grande volume de exemplos de frases, fazendo com que saiba quais palavras têm relação e podem ser utilizadas em conjunto. É uma ferramenta extremamente valorizada no campo de *Natural Language Processing* (NLP), principalmente na parte de tradução, pois, na prática, permite que, ao traduzir um texto, a ordem das palavras seja levada em consideração. Podemos imaginar diversas expressões que, ao serem traduzidas do português para o inglês, ilustrariam esse caso.

- O seu grande diferencial, ***self-attention***, é um tipo específico de atenção utilizado em modelos *Transformers*. É o que, na prática, possibilita ir além da tradução de textos, permitindo a realização de diversas questões de NLP, como, por exemplo, sintetizar textos, analisar sentimentos, responder a perguntas, obedecer a regras gramaticais, entre outras funções. Isso ocorre, pois, modelos que a implementam prestam atenção na própria entrada enquanto a mesma está sendo codificada.

- ***Positional encodings*** é o que torna possível, juntamente com o mecanismo de atenção, a paralelização de treinamento anteriormente mencionada. Como o nome sugere, durante a codificação, são acrescentadas às entradas vetores posicionais com a finalidade de representar a sua ordem na sentença. Conforme o modelo é treinado, essas informações o enriquecem, permitindo o estabelecimento de dependência entre palavras, até mesmo em sequências longas. Isso tudo sem a necessidade do uso de recorrência ou convolução, aumentando seu desempenho imensamente se comparado às RNNs.

A Figura 10 exibe o modelo da arquitetura *Transformer*, em que a parte esquerda da figura ilustra o **codificador** e a direita o **decodificador**. Outros elementos fundamentais, como o processo de **tokenização** e o mecanismo de **atenção** também estão presentes.

Figura 10 – Arquitetura modelo - *Transformer*



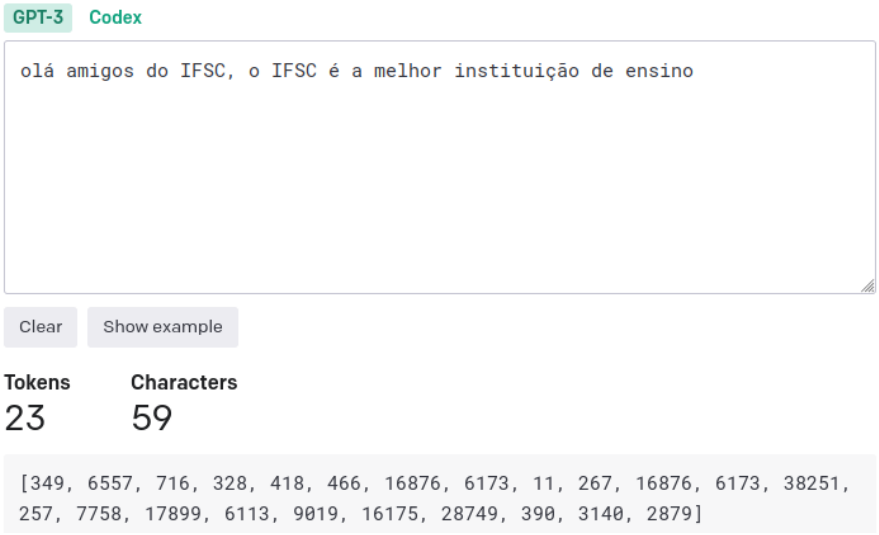
Fonte: Kosar (2022)

A tokenização, fundamental para a representação de palavras e demais caracteres através de inteiros, é o processo que torna possível que as entradas, denominadas ***Input Embeddings***, que são vetores carregando valores que representam seus vínculos com demais palavras e informações semânticas, alimentem o modelo. Ou seja, tanto entrada e saída do modelo são inteiros, que são traduzidos para texto através do processo de tokenização (VASWANI et al., 2017). Um exemplo do tokenizador[1] utilizado pelo modelo

[1] Disponível em: https://platform.openai.com/tokenizer

GPT-3, da OpenAI, pode ser observado na Figura 11.

Figura 11 – Processo de tokenização do modelo GPT-3



Fonte: O próprio autor

A respeito do codificador, trata-se de uma parte fundamental que, no modelo original, descrito por Vaswani et al. (2017), possui seis camadas. Cada uma recebe uma lista de vetores, com 512 de dimensão, em sua entrada. É aplicado o mecanismo de *self-attention* já descrito, que alimenta uma Rede Neural antes de ir para a próxima camada, sendo que a última camada de codificador aciona o decodificador, através de vetores de atenção.

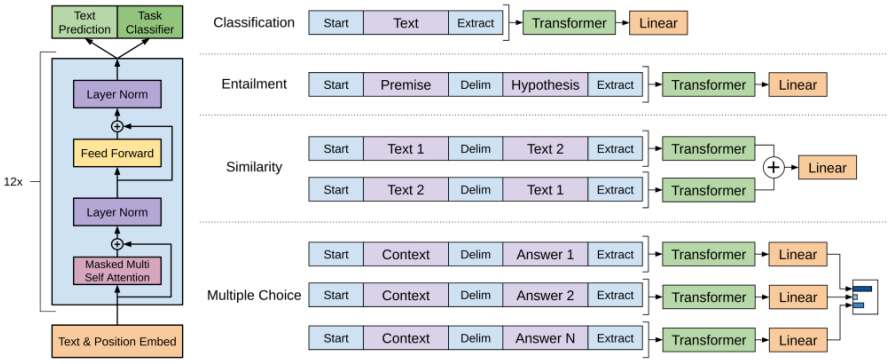
O decodificador, também composto por seis camadas, desloca os vetores recebidos para a direita, basicamente para adicionar, à esquerda, um *token* especial sinalizador de início. Além disso, há o processo de realimentação, proveniente da camada anterior. O restante é bem parecido com o processo executado pelo codificador. Por fim, o resultado é um número calculado pela função *Softmax*, que, em linhas gerais, é a probabilidade de determinados *ids*, correspondente a um ou mais *tokens*, serem a saída do modelo (ALLAMAR, 2018).

2.4.1 Modelo *Generative Pre-Trained Transformer* (GPT)

O modelo escolhido para o trabalho foi o GPT, como o nome sugere, voltado para a geração de textos, ou seja, aplica o conceito de IA generativa. É definido por Radford et al. (2018), no artigo original da técnica, como uma variante do *Transformer*, citado na seção 2.4. Mas uma variação poderosa, com a adição de seis camadas de decodificadores, conforme arquitetura ilustrada na Figura 12.

GPT difere do modelo base ao utilizar apenas decodificadores, por isso é conhecido por ser um modelo de linguagem multicamada do tipo *Transformer decoder*. Para treina-

Figura 12 – Arquitetura modelo - GPT



Fonte: Radford et al. (2018)

mento, foi utilizado um *dataset* chamado BooksCorpus, contendo mais de 7.000 livros, de variados temas. (RADFORD et al., 2018). O resultado foi um modelo de linguagem com mais de 125 milhões de parâmetros, entre *Input Embeddings* e *Positional Encodings*, com excelente desenvoltura para gerar textos com linguagem fluente e coerente (BROWN et al., 2020).

Suas versões posteriores contam com um treinamento muito mais exaustivo através de conjunto de dados maiores e mais diversificados, como o *dataset* Common Crawl[2], e com um aumento expressivo no número de parâmetros. Com isso, respostas sofisticadas para um maior número de tarefas de NLP se tornaram possíveis, como, por exemplo, criação de arte e códigos de programação, ou então seu fruto mais popular, o ChatGPT (BROWN et al., 2020; OpenAI, 2023; RADFORD et al., 2019).

A Tabela 1 traz algumas características dos modelos GPT. É importante destacar que, entre as publicações principais, há versões intermediárias, não incluídas nessa listagem.

Tabela 1 – Comparação entre versões do GPT

Modelo	Ano	Parâmetros	Camadas	Capacidade de memória
GPT	2018	125 milhões	12	1024 tokens
GPT-2	2019	até 1.5 bilhões	até 48	1024 tokens
GPT-3	2020	até 175 bilhões	até 96	até 16.384 tokens
GPT-4	2023	não informado	não informado	até 32.768 tokens

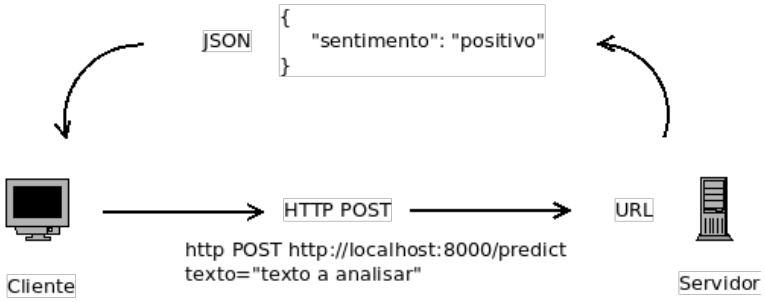
2.4.2 Interface de Programação e Aplicação

As consultas e respostas do modelo GPT ajustado para análise de sentimentos serão consumidas através de uma web *Application Programming Interface* (API), que forma uma série de regras e protocolos que permitem a comunicação de diferentes softwa-

[2] Link para o *dataset* público

res, ainda que sejam desenvolvidos em linguagens distintas (MOZILLA, 2023). A Figura 13 ilustra uma API voltada ao projeto, em que o método *Hypertext Transfer Protocol* (HTTP) utilizado é o *POST*, possibilitando ao usuário, através de uma aplicação web ou até mesmo de outro servidor, submeter textos para avaliação através do modelo. A resposta será no formato *JavaScript Object Notation* (JSON).

Figura 13 – Exemplo de uma API



Fonte: O próprio autor

A arquitetura desta API atende ao conceito *Representational State Transfer* (REST), baseado em requisições de um cliente a um servidor, em que os dados são facilmente lidos pelo solicitante. É possível de escalar, simples de utilizar e confiável, pois mantém um padrão de respostas nos casos de sucesso ou falha (FIELDING, 2000).

3 DESENVOLVIMENTO

Para o ajuste do modelo GPT à tarefa alvo deste trabalho, optou-se por trabalhar com a plataforma Google Colaboratory[1], que oferece, de forma gratuita, ambientes de notebooks Jupyter[2] para execução de códigos em Python[3], que é uma linguagem popular em atividades relacionadas à ML.

Inicialmente o modelo explorado para a tarefa-alvo deste projeto foi o GPT-2, porém, com a popularização do ChatGPT, optou-se por migrar para o GPT-3.

Para alcançar o objetivo final deste trabalho, que é desenvolver uma aplicação web que faça análise de sentimentos, alguns passos são necessários e serão detalhados neste capítulo.

3.1 Dados rotulados

Para compor a base de dados rotulados, item crucial para análise de sentimentos, foi escolhida a Play Store, mais precisamente as avaliações de usuários aos aplicativos mais populares nos últimos meses. O fato dos comentários estarem associados a um número de estrelas entre 1 e 5 facilitou bastante o tratamento da informação. A programação envolvida para obtenção e limpeza dos dados está disponível no Apêndice A, conforme Código A.1 e Código A.2, respectivamente.

Conforme a Figura 14, podemos observar que, de 12.000 comentários, restaram pouco mais de 7.400. Isso porque os comentários com avaliações 1 e 2 estrelas foram classificados como negativos, e os comentários com 4 e 5 estrelas, como positivos. Comentários neutros foram descartados, além de linhas mal formadas. Tudo isso para respeitar uma regra básica de ML, que é oferecer dados com qualidade ao modelo, para que tenhamos um bom resultado.

3.2 Modelo GPT-2

De acordo com a subseção 2.4.1, os modelos GPTs já são treinados com um grande *dataset*, sendo necessário, para a tarefa de análise de sentimento, um *fine-tuning* ou ajuste fino. Neste trabalho, foram submetidas 7.400 frases rotuladas ao modelo, que já são adequadas para se obter um resultado interessante. Através da biblioteca scikit-learn[4],

[1] Endereço oficial do Google Colaboratory
[2] Endereço oficial do Jupyter Notebook
[3] Para o desenvolvimento deste trabalho utilizou-se a versão 3.9.2 do Python
[4] Biblioteca para aprendizado de máquina disponível em: https://scikit-learn.org/stable/

Figura 14 – Trecho do Código A.2 em formato Jupyter Notebook

```
[ ] df.head()
```

	reviewId	userName	userImage	content	score
0	f124c390-90c3-4153-9ca1-8e80c0f30a2c		https://play-lh.googleusercontent.com/a-/ACB-R...	O aplicativo era ótimo, os descontos eram ótim...	1
1	ce308907-5ebe-493e-b87a-0a12e0f4c1c5		https://play-lh.googleusercontent.com/a-/ACB-R...	Pior experiência que tive de suporte com um ap...	1
2	accff4a0-f70e-485a-a85d-04fe3a2e7238		https://play-lh.googleusercontent.com/a-/ACB-R...	85% das compras chegam erradas, incompletas ou...	1
3	ec2fc975-0f38-4c2e-81c1-7b9dc7398faf		https://play-lh.googleusercontent.com/a/AGNmyx...	O aplicativo já foi melhor, tinha mais cupons ...	1
4	99cf027e-9fe9-46be-944c-220e9f114711		https://play-lh.googleusercontent.com/a/AGNmyx...	Pior aplicativo de delivery que já usei na min...	1

```
[ ] df.shape
```

```
(12000, 13)
```

(a) *Dataset* original

```
[ ] #após remoção das linhas que não contenham uma avaliação válida através do Vim, temos o dataset:
    df = pd.read_csv("/home/joao/Dropbox/Aula/TCC2/dataset/reviews_final.csv")
    df.head()
```

	4	não conheço aplicativo melhor.
0	0	Retiraram o cupom Ganhe20 de primeiro pedido?
1	4	Muito bom cheio de descontos
2	4	muito bom adorei ..só podia entregar no meu Ba...
3	0	Ótimo para pedir de tudo eficiênciente porém n...
4	4	O app tem sido essencial na pandemia. Ainda so...

```
df.shape
```

```
(7493, 2)
```

(b) Dados ajustados

Fonte: O próprio autor

95% dessas avaliações foram destinadas ao refinamento, enquanto 5% foram reservadas para testes de desempenho.

O modelo inicialmente escolhido para se trabalhar foi o GPT-2, pois, através da biblioteca Python da empresa Hugging Face[5], temos acesso ao modelo pré-treinado[6], de forma aberta e para uso totalmente off-line.

Através da plataforma Kaggle[7], foi possível obter um código para *fine-tuning* do GPT-2 que, com alguns ajustes, utilizou o *dataset* descrito na seção 3.1 e foi compatibilizado para trabalhar com o Google Colaboratory. O uso do Colab fez toda a diferença, uma vez que dispõe de *Graphics Processing Unit* (GPU) de forma gratuita, o que diminuiu, e muito, o tempo de refinamento. O código Kaggle para *fine-tuning* do modelo GPT-2, da biblioteca Hugging Face, está disponível no Apêndice B.

3.2.1 Resultados obtidos para o modelo GPT-2

A Figura 15, obtida através da execução do Código B.1, disponível no Apêndice B, sintetiza o resultado após o refinamento: aproximadamente 3 horas de execução, para um aproveitamento de 82%[8]. Tentamos repetir o mesmo processo através de *Central Processing Unit* (CPU), a partir de um desktop residencial intermediário, porém levaria dias para finalizar.

A Tabela 2 relaciona os resultados verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos através da **Matriz de Confusão**, que é um indicador que sumariza a performance de um modelo de ML.

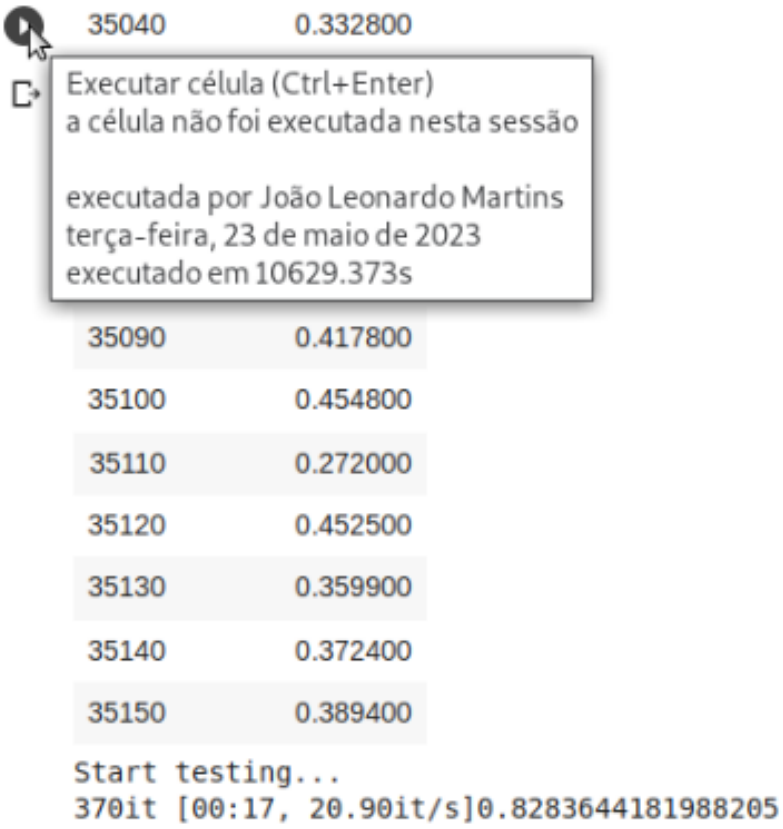
Tabela 2 – Matriz de confusão após testes com o modelo *Generative Pre-Trained Transformer-2* (GPT-2)

	Previsto como positivo	Previsto como negativo
Dataset como positivo	164	44
Dataset como negativo	21	141

O modelo treinado e um pequeno código para a realização de novas inferências estão disponíveis no repositório GitHub descrito no Apêndice A. Uma demonstração do modelo em funcionamento, ainda em ambiente de desenvolvimento, pode ser conferida na Figura 16.

[5] Empresa que fornece ferramentas para desenvolvimento na área de ML
[6] Disponível em: https://huggingface.co/gpt2
[7] Disponível em: kaggle.com/code/evilmage93/gpt-2-finetuning-on-sentiment-classification/notebook
[8] Esse resultado se refere à assertividade do modelo ao inferir os sentimentos das frases da amostra de testes retirada do *dataset*, que, neste caso, são de 370 avaliações.

Figura 15 – Tempo de execução e aproveitamento do modelo pós *fine-tuning*



Fonte: O próprio autor

Figura 16 – Utilizando o modelo GPT-2

```
input_text = "aplicativo difícil de entender, vive travando"

# predict sentiment on test data
prompt = f'<|startoftext|>Review: {input_text}\nSentiment:'
#generated = tokenizer(f"<|startoftext|> {prompt}", return_tensors="pt").input_ids
generated = tokenizer(f"<|startoftext|> {prompt}", return_tensors="pt").input_ids
sample_outputs = model.generate(generated, do_sample=False, top_k=50, max_length=512, top_p=0
                                temperature=0, num_return_sequences=0)
pred_text = tokenizer.decode(sample_outputs[0], skip_special_tokens=True)
print(pred_text)
# extract the predicted sentiment
try:
  pred_sentiment = re.findall("\nSentiment: (.*)", pred_text)[-1]
except:
  pred_sentiment = "None"

print(pred_sentiment)
```

```
Special tokens have been added in the vocabulary, make sure the associated word embeddings are
fine-tuned or trained.
The attention mask and the pad token id were not set. As a consequence, you may observe unexpec
ted behavior. Please pass your input's `attention_mask` to obtain reliable results.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
 Review: aplicativo difícil de entender, vive travando
Sentiment: negative
None
```

Fonte: O próprio autor

3.3 Modelo GPT-3

Este modelo não foi disponibilizado por completo pela empresa desenvolvedora, a OpenAI. Portanto, não há uma versão *open-source* para *fine-tuning* e utilização off-line. A solução foi utilizar a API paga[9], que possui cotas de testes e limites bem interessantes. Para o objetivo deste trabalho, não chegamos nem próximo dos limites para cobranças.

Ainda assim, vale destacar que a OpenAI permite, através de sua API, refinamentos ou *fine-tuning* em *datasets* customizados. Isso seria aplicável, por exemplo, caso o Instituto Federal de Santa Catarina (IFSC) desenvolvesse um *chatbot* para auxiliar os estudantes em procedimentos institucionais. Neste caso, o modelo poderia ser alimentado com as regras gerais e específicas dos cursos oferecidos pelo IFSC.

Através da API da OpenAI, temos acesso a diversos modelos, sendo o mais adequado para o objetivo do trabalho, no momento, o **gpt-3.5-turbo**[10]. Isso por ser elegível para o teste gratuito, ter um excelente desempenho e baixo consumo de *tokens*.

Por se tratar de um modelo comercial, treinado com uma quantidade consideravelmente maior de informações de diferentes espécies e idiomas, para a tarefa de análise de sentimento, não se fez necessário um refinamento ou *fine-tuning*. Dada a robustez do modelo, foi possível adicionar também a opção de verificação de sentimentos neutros, além de, se houverem, retornar informações como marca, se há raiva e qual o item do comentário, além de traduzi-lo, caso não esteja em português.

3.3.1 Resultados obtidos para o modelo GPT-3

A utilização da API da OpenAI é bastante simples: precisamos de um *prompt*, que define o comportamento que o modelo terá e que tipo de resposta deve retornar, além do texto em si, que será submetido ao modelo. Podemos conferir um exemplo na Figura 17. O código, na íntegra, está disponível no Apêndice C.

A mesma amostra submetida ao modelo GPT-2, com 370 avaliações de usuários da Play Store, foi aplicada ao modelo da OpenAI, através de API. O resultado foi um desempenho levemente superior, de 88%, conforme Figura 18, com trecho do código comparador, disponível no Apêndice C, Código C.2. Já a Tabela 3 sumariza o desempenho do modelo GPT-3 através da Matriz de Confusão.

Buscando compreender as divergências apresentadas, algumas frases foram listadas, conforme Figura 19. Chegou-se a conclusão de que alguns usuários simplesmente não atribuíram uma avaliação condizente com o comentário, implicando nessas diferenças observadas entre *dataset* e as previsões realizadas pelo modelo.

[9] Página oficial da OpenAI sobre preços
[10] Página oficial do modelo da OpenAI

Figura 17 – Exemplo de *prompt* e resposta do GPT-3

```
prompt = f"""
Identifique os seguintes itens a partir do texto de review:
- Sentimento (positivo ou negativo)
- O avaliador parece estar bravo? (verdadeiro ou falso)
- Item comprado pelo avaliador
- Fabricante ou vendedor do item

O texto de reviews está delimitado por tripla crase. \
Formate a resposta com as seguintes chaves: \
"Sentimento", "Bravo", "Item" and "Marca".
Se não conseguir obter alguma informação, preencha o valor\
 como "Desconhecido" Faça a resposta o mais curta possível.
O valor de "Bravo" é Verdadeiro ou Falso.

A resposta será no idioma de '''{review}'''.

Review text: '''{review}'''
"""
response = get_completion(prompt)
print(response)
```

```
{
"Sentimento": "Negativo",
"Bravo": "Falso",
"Item": "Bicicleta",
"Marca": "Desconhecido"
}
```

Fonte: O próprio autor

Figura 18 – Desempenho do GPT-3 com amostra do *dataset*

```
#Calculando a diferença entre o test_dataset e o predict_gpt3:
i = 0
for x in range(370):
  #print(x)
  if (test_dataset[1][x] == 4 and predicted2[x] != 'positive') or \
   (test_dataset[1][x] == 0 and predicted2[x] != 'negative'):
    #print(test_dataset[0][x])
    i = i + 1
    #print(x)

aproveitamento = 100 - ((100*i)/370)
print("\n\nHouve " + str(i) + " difenças..\nEquivalente a um aproveitamento\
 de " + str(aproveitamento) + "%")
```

```
Houve 44 difenças..
Equivalente a um aproveitamento de 88.10810810810811%
```

Fonte: O próprio autor

Tabela 3 – Matriz de confusão após testes com o modelo *Generative Pre-Trained Transformer-3* (GPT-3)

	Previsto como positivo	Previsto como negativo
Dataset como positivo	149	9
Dataset como negativo	36	176

Figura 19 – Exemplos de divergências entre modelo e conjunto de dados de teste

```
x = 151
print('\n' + str(test_dataset[0][x]))
print("Avaliação original / prevista: " + str(test_dataset[1][x])\
      + " / " + predicted2[x])
x = 177
print('\n' + str(test_dataset[0][x]))
print("Avaliação original / prevista: " + str(test_dataset[1][x])\
      + " / " + predicted2[x])
x = 179
print('\n' + str(test_dataset[0][x]))
print("Avaliação original / prevista: " + str(test_dataset[1][x])\
      + " / " + predicted2[x])
```

```
A taxa de serviço as vezes é maior que a taxa de entrega.
Avaliação original / prevista: 4 / negative

Vou experimentar
Avaliação original / prevista: 0 / positive

Não tem como cancelar o pedido
Avaliação original / prevista: 4 / negative
```

Fonte: O próprio autor

3.4 Comparação entre os modelos apresentados

O modelo GPT-2, por mais limitado que seja, apresentou um desempenho interessante (82%) dentro do *dataset* formado por comentários da Play Store. Outros pontos positivos são a possibilidade de trabalhar off-line, sem expor dados de refinamento na internet, e ser uma alternativa *open-source*. Como limitante, podemos destacar a complexidade de adicionar novas funções, como compreender sentimentos neutros, marcas, produtos, entre outros.

Já o GPT-3 é voltado para o público que não se importa de, eventualmente, pagar pelo uso de uma ferramenta muito poderosa, que possui uma API para consultas e também para *fine-tuning*, que consegue extrair ao máximo as intenções por trás de cada frase submetida a ela. O ônus fica por conta de ser uma solução dependente da nuvem, em que os dados de treinamento serão compartilhados com a empresa para alimentar o modelo, além de que, dependendo do nível do usuário, as requisições podem entrar em uma fila de baixa prioridade.

O modelo GPT-3 será utilizado para o desenvolvimento do trabalho, não por conta do desempenho ligeiramente superior ao GPT-2 (88% versus 82%) dentro do *dataset* da seção 3.1, mas sim tendo em vista que permitirá a elaboração de uma API muito mais completa e com tecnologia mais atual.

3.5 Desenvolvimento das interfaces

Para o desenvolvimento do *back-end* e *front-end* foram utilizadas tecnologias atuais e que possibilitam o sistema ser aplicado em um cenário real, permitindo o mesmo a trabalhar em escala, atendendo a múltiplas requisições. De modo a tornar os sistemas replicáveis, visando, principalmente, escalabilidade e portabilidade, ambos serão empacotados em contêineres Docker[11] e inicializados através de Docker Compose[12].

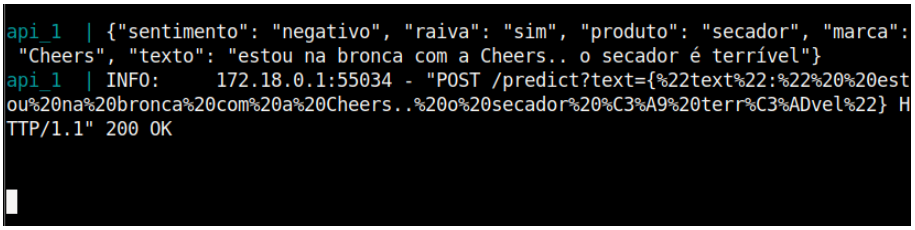
3.5.1 *Application Programming Interface* (API)

A web API foi desenvolvida através da linguagem Python, utilizando o *framework* FastAPI[13], que está de acordo com a arquitetura de API escolhida e possui um simples desenvolvimento, mas com alta performance. O Código 3.1 exibe um trecho do código da API e a Figura 20 ilustra essa requisição/resposta. O restante da programação envolvida nessa interface, bem como as instruções para o empacotamento utilizando Docker estão disponíveis no repositório apontado pelo Apêndice A.

Código 3.1 – Rota da API do analisador de sentimentos

```
@app.post("/predict",response_model = SentimentResponse)
async def getpredict(text: str):
    JSONResponse  = predict(str(text))
    print(JSONResponse)
    DicResponse  = json.loads(JSONResponse)
    return SentimentResponse(
        Sentimento = DicResponse["sentimento"].lower(),
        Raiva = DicResponse["raiva"].lower(),
        Produto = DicResponse["produto"].lower(),
        Marca = DicResponse["marca"].lower(),
        Avaliacao = DicResponse["texto"].lower()
    )
```

Figura 20 – Resposta típica da API desenvolvida



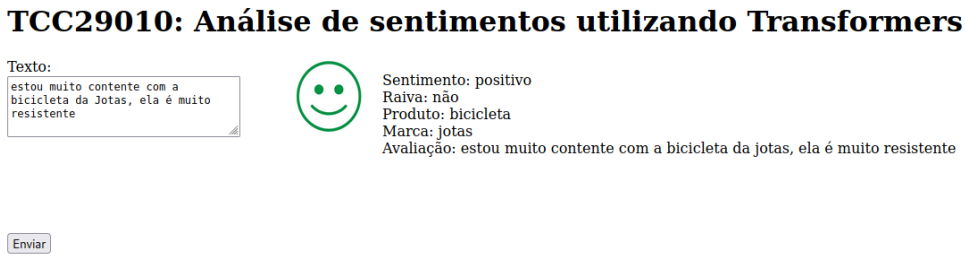
Fonte: O próprio autor

[11] Documentação oficial do Docker
[12] Docker Compose é uma ferramenta para gerenciar aplicações multicontêiner
[13] Página oficial do *framework* FastAPI

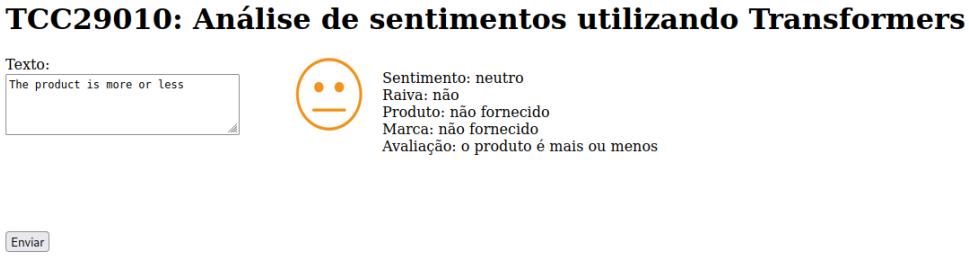
3.5.2 Interface web

De modo a simplificar a consulta e visualização dos resultados, uma simples página web foi desenvolvida, utilizando *HyperText Markup Language* (HTML), com pequenas funções de interatividade escritas em JavaScript. O resultado pode ser visto na Figura 21 e o código completo no repositório do trabalho no GitHub.

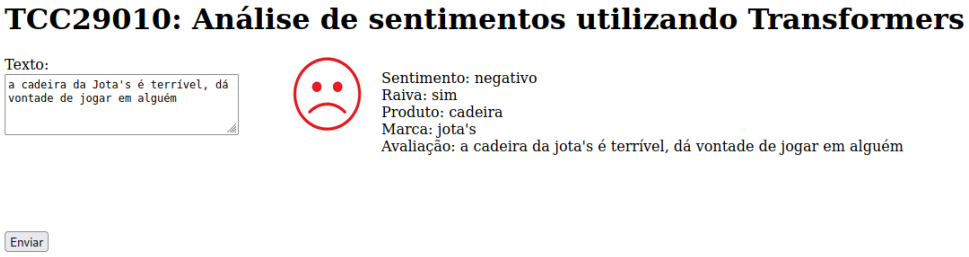
Figura 21 – Página web para interação com o modelo GPT-3



(a) Exemplo de comportamento para avaliação positiva



(b) Exemplo de comportamento para avaliação neutra



(c) Exemplo de comportamento para avaliação negativa

Fonte: O próprio autor

4 CONCLUSÕES

O objetivo principal deste trabalho foi desenvolver uma aplicação, baseada em *Transformers*, para análise de sentimentos. O modelo escolhido foi o *Generative Pre-Trained Transformer* (GPT), que, apesar de ser muito utilizado para geração de textos, pôde perfeitamente ser adaptado para atender ao objetivo proposto.

Foi possível observar, durante o desenvolvimento do projeto, o grande avanço das IAs generativas, que acabou mudando a estratégia de desenvolvimento do trabalho, tendo em vista que a popularização do ChatGPT se deu no meio da elaboração. Optou-se por migrar do modelo GPT-2 para o GPT-3, o que implicou em um resultado muito mais rico em termos de serviço.

Ainda assim, os estudos voltados ao modelo aberto GPT-2 foram engrandecedores. Coletar dados rotulados de modo a formar um *dataset*, proceder com o *fine-tuning* para somente então realizar consultas agregou bastante conhecimento em ML. Sem contar que os resultados obtidos demonstram o quão eficazes os *Transformers* são na área de NLP, especialmente para análise de sentimentos, mesmo os modelos mais simples e refinados com poucos dados.

Já com o uso do modelo GPT-3, novas funcionalidades foram adicionadas ao sistema, o que não seria possível com a tecnologia anterior, além de, na prática, representar um crescimento no teste de desempenho realizado sobre uma mesma amostra de dados. Vale ressaltar que as diferenças e implicações de se utilizar um modelo ou outro foram expostas, desde financeiras até tecnológicas.

Para uma melhor interação com os usuários, foram desenvolvidas interface web e API utilizando ferramentas *open-source* atuais de mercado, como FastAPI e Docker. Com isso, o trabalho ficou replicável e abre margem para ajustes de tecnologia no futuro, como, por exemplo, o uso de outro modelo *open-source* para realizar a tarefa de análise de sentimentos.

4.1 Trabalhos futuros

Além da possibilidade da utilização de outras versões de GPT *open-source* já mencionada, outras sugestões de projetos são:

- Adaptar o trabalho para utilização do modelo *Bidirectional Encoder Representations from Transformers* (BERT) e comparar o aproveitamento com o da técnica GPT, sobre o mesmo *dataset*;

- Através da API da OpenAI, desenvolver um *chatbot* voltado ao público do IFSC para auxiliar em procedimentos internos, em que seria obrigatório realizar um *fine-tuning* com as informações necessárias.

REFERÊNCIAS

ACADEMY, D. S. *Deep Learning Book*. 2019. Disponível em: https://www.deeplearningbook.com.br/. Acesso em: 25 de nov. de 2022. 30

ALBAYRAK, N.; ÖZDEMIR, A.; ZEYDAN, E. An overview of artificial intelligence based chatbots and an example chatbot application. In: IEEE. *2018 26th signal processing and communications applications conference (SIU)*. [S.l.], 2018. p. 1–4. 23

ALLAMAR, J. *The Illustrated Transformer. In: Jay Allamar.* ***Visualizing machine learning one concept at a time****. 27 jun.* 2018. Disponível em: https://jalammar.github.io/illustrated-transformer. Acesso em: 27 de jun. de 2023. 38

BENGIO, Y. *Deep Learning*. London, England: MIT Press, 2016. (Adaptive Computation and Machine Learning series). 31, 32

BERNARD, E. *Introduction To Machine Learning*. Champaign, IL: Wolfram Media, 2021. 28, 30, 31, 32, 33, 34

BROWN, T. et al. Language models are few-shot learners. *Advances in neural information processing systems*, v. 33, p. 1877–1901, 2020. 39

CHOWDHARY, K. R. *Fundamentals of artificial intelligence*. 1. ed. New Delhi, India: Springer, 2020. 27

DEVLIN, J. et al. BERT: Pre-training of deep bidirectional transformers for language understanding. out. 2018. 23

EISENSTEIN, J. *Introduction to natural language processing*. London, England: MIT Press, 2019. (Adaptive Computation and Machine Learning series). 34

FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. [S.l.]: University of California, Irvine, 2000. 40

FORBES. *ChatGPT tem recorde de crescimento da base de usuários*. 2023. Disponível em: https://forbes.com.br/forbes-tech/2023/02/chatgpt-tem-recorde-de-crescimento-da-base-de-usuarios/. Acesso em: 20 de jun. de 2023. 23

GRAVES, A. *Supervised sequence labelling with recurrent neural networks*. 2012. ed. Berlin, Germany: Springer, 2012. (Studies in computational intelligence). 33

HAN, K. et al. Transformer in transformer. In: RANZATO, M. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2021. v. 34, p. 15908–15919. Disponível em: https://proceedings.neurips.cc/paper/2021/file/854d9fca60b4bd07f9bb215d59ef5561-Paper.pdf. 36

JAREK, K.; MAZUREK, G. Marketing and artificial intelligence. *Central European Business Review*, v. 8, n. 2, 2019. 23

KAPLAN, J. *Artificial intelligence*. New York, NY: Oxford University Press, 2016. (What Everyone Needs To Know (R)). 28

KOSAR, V. *Transformer Embeddings and Tokenization. In: Vaclav Kosar.* ***Software Machine Learning Blog****. 5 jun.* 2022. Disponível em: https://vaclavkosar.com/ml/transformer-embeddings-and-tokenization. Acesso em: 26 de jun. de 2023. 37

LIDDY, E. Natural language processing. In: DRAKE, M. A. (Ed.). *Encyclopedia of Library and Information Science*. New York, NY: Marcel Decker, Inc, 2001. p. 2126–2136. 35

LUND, B. D.; WANG, T. Chatting about chatgpt: how may ai and gpt impact academia and libraries? *Library Hi Tech News*, Emerald Publishing Limited, v. 40, n. 3, p. 26–29, 2023. 23

MARKOWITZ, D. *Transformers, Explained: Understand the Model Behind GPT-3, BERT, and T5. In: Dale Markowitz.* ***Dale on AI****. 6 mai.* 2021. Disponível em: https://daleonai.com/transformers-explained. Acesso em: 26 de jun. de 2023. 36

MARTINEZ, P.; AL-HUSSEIN, M.; AHMAD, R. A scientometric analysis and critical review of computer vision applications for construction. *Automation in Construction*, Elsevier, v. 107, p. 102947, 2019. 24

MOZILLA. *Introduction to web APIs. In: MDN Web Docs.* ***MDN Web Docs****. 8 mai.* 2023. Disponível em: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side$_{w}eb_{A}PIs/Introduction.Acessoem$: 26$dejun.de$2023. 40

ONGSULEE, P. Artificial intelligence, machine learning and deep learning. In: *2017 15th International Conference on ICT and Knowledge Engineering (ICTKE)*. [S.l.: s.n.], 2017. p. 1–6. 29

OpenAI. *GPT-4 Technical Report*. arXiv, 2023. Disponível em: https://arxiv.org/abs/2303.08774. 39

RADFORD, A. et al. *Improving language understanding by generative pre-training. In: Alec Radford.* ***OpenAI Blog****. 11 jun.* 2018. Disponível em: https://openai.com/research/language-unsupervised. Acesso em: 26 de jun. de 2023. 38, 39

RADFORD, A. et al. *Language Models are Unsupervised Multitask Learners. In: Alec Radford.* ***OpenAI Blog****. 14 feb.* 2019. Disponível em: https://openai.com/research/better-language-models. Acesso em: 26 de jun. de 2023. 39

SCHROTENBOER, D. The impact of artificial intelligence along the customer journey: a systematic literature review. University of Twente, 2019. 23

SEARLE, J. R. Minds, brains, and programs. *Behavioral and Brain Sciences*, Cambridge University Press, v. 3, n. 3, p. 417–424, 1980. 27

SÜTÇÜ, C. S.; AYTEKİN, Ç. AN EXAMPLE OF PRAGMATIC ANALYSIS IN NATURAL LANGUAGE PROCESSING: SENTIMENTAL ANALYSIS OF MOVIE REVIEWS. *TURKISH ONLINE JOURNAL OF DESIGN ART COMMUNICATION*, Turkish Online Journal of Design, Art and Communication (TOJDAC), CTC, n. 2019,

p. 61–74, dez. 2019. Disponível em: https://doi.org/10.7456/ctc_2019_05. Acesso em: 27 jun. 2023. 35

VARGAS, A. C. G.; PAES, A.; VASCONCELOS, C. N. Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres. In: SN. *Proceedings of the xxix conference on graphics, patterns and images*. [S.l.], 2016. v. 1, n. 4. 32

VASWANI, A. et al. Attention is all you need. *Advances in neural information processing systems*, v. 30, 2017. 36, 37, 38

VILLANUEVA, M. B.; SALENGA, M. L. M. Bitter melon crop yield prediction using machine learning algorithm. *International Journal of Advanced Computer Science and Applications*, Science and Information (SAI) Organization Limited, v. 9, n. 3, 2018. 27

WAKEFIELD, K. *A guide to machine learning algorithms and their applications*. 2019. Disponível em: https://www.sas.com/en$_g$*b*/*insights*/*articles*/*analytics*/*machine* − *learning* − *algorithms*.*html*.*Acessoem* : 20*denov*.*de*2022. 28

XAVIER, F. C. *Fundamentos, pilares e aplicações da Inteligência Artificial no setor público - MIT Technology Review*. 2020. Disponível em: https://mittechreview.com.br/fundamentos-pilares-e-aplicacoes-da-inteligencia-artificial-no-setor-publico/. 27

ZAGO, B. *Panorama de Atendimento online pré e pós-pandemia E tendências para 2021*. 2021. Disponível em: https://www.ecommercebrasil.com.br/artigos/panorama-de-atendimento-online-pre-e-pos-pandemia-e-tendencias-para-2021/. Acesso em: 24 de jul. de 2022. 24

ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. In: FLEET, D. et al. (Ed.). *Computer Vision – ECCV 2014*. Cham: Springer International Publishing, 2014. p. 818–833. ISBN 978-3-319-10590-1. 32

Apêndices

APÊNDICE A – OBTENÇÃO E LIMPEZA DE DADOS PARA FORMAR UM *DATASET*

Todos os códigos utilizados neste trabalho estão disponíveis no Github, através do link https://github.com/jaoleonardo01/tcc29010.

Código A.1 – Download de comentários de alguns aplicativos através da API da Play Store

```
#!/usr/bin/env python
# coding: utf-8

#Install Google play scraper: https://github.com/JoMingyu/google-play-scraper
get_ipython().system('pip install google_play_scraper')

import json
import pandas as pd
from tqdm import tqdm

import seaborn as sns
import matplotlib.pyplot as plt

from pygments import highlight
from pygments.lexers import JsonLexer
from pygments.formatters import TerminalFormatter

from google_play_scraper import Sort, reviews, app

get_ipython().run_line_magic('matplotlib', 'inline')
get_ipython().run_line_magic('config', "InlineBackend.figure_format='retina'")

sns.set(style='whitegrid', palette='muted', font_scale=1.2)

apps_ids = ['br.com.brainweb.ifood', 'com.cerveceriamodelo.modelonow',
'com.mcdo.mcdonalds', 'habibs.alphacode.com.br',
'com.xiaojukeji.didi.brazil.customer',
'com.ubercab.eats', 'com.grability.rappi',
'burgerking.com.br.appandroid', 'com.nzn.tdg',
'com.vanuatu.aiqfome']

# Scraping data for each app

app_infos = []
```

```
35
36 for ap in tqdm(apps_ids):
37     info = app(ap, lang='en', country='us')
38     del info['comments']
39     app_infos.append(info)
40
41 app_infos_df = pd.DataFrame(app_infos)
42 app_infos_df.head()
43
44 # #### Scraping App Reviews
45 #
46 # We want:
47 # * Balanced dataset   roughly the same number of reviews for each score (15)
48 # * A representative sample of the reviews for each app
49 #
50 #
51 # We can satisfy the first requirement by using the scraping package option to
       filter the review score. For the second, well sort the reviews by their
       helpfulness, which are the reviews that Google Play thinks are most important.
52
53 app_reviews = []
54
55 for ap in tqdm(apps_ids):
56     for score in list(range(1, 6)):
57         for sort_order in [Sort.MOST_RELEVANT, Sort.NEWEST]:
58             rvs, _ = reviews(
59                 ap,
60                 lang='pt',
61                 country='br',
62                 sort=sort_order,
63                 count= 200 if score == 3 else 100,
64                 filter_score_with=score
65             )
66             for r in rvs:
67                 r['sortOrder'] = 'most_relevant' if sort_order == Sort.
       MOST_RELEVANT else 'newest'
68                 r['appId'] = ap
69             app_reviews.extend(rvs)
70
71 len(app_reviews)
72
73 # Saving reviews in a CSV file
74 app_reviews_df = pd.DataFrame(app_reviews)
75 app_reviews_df.head()
76 app_reviews_df.to_csv('reviews.csv', index=None, header=True)
```

Código A.2 – Ajuste no *dataset* para limpeza e compatibilização com o código do modelo

```
#!/usr/bin/env python
# coding: utf-8

# ### Modelo:
#
# -Disponível em: https://www.dropbox.com/sh/j3lpkp4zqxw4ppw/AACYdSsBPww1N2GNOWCRooPDa?dl=0

# In[1]:


import numpy as np
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc

df = pd.read_csv("/home/joao/Dropbox/Aula/TCC2/dataset/reviews.csv")
df.shape

def to_sentiment(rating):
  rating = int(rating)
  if rating <= 2:
    return 0
  elif rating == 3:
    return 'none'
  else:
    return 4

df['score'] = df.score.apply(to_sentiment)
df2 = df[['score','content']]
df2

#Remove possíveis valores nulos
df3 = df2['content'].notna()
#Confere se não há valores nulos
df3.isnull().values.any()
#após remoção das linhas que não contenham uma avaliação válida através do Vim, temos o dataset:
df = pd.read_csv("/home/joao/Dropbox/Aula/TCC2/dataset/reviews_final.csv")
df.head()
df.shape
```

APÊNDICE B – *FINE-TUNING* DO MODELO GPT-2

Código B.1 – *Fine-tuning* do modelo GPT-2

```
# -*- coding: utf-8 -*-
"""colab_fine-tun_gpt2

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1Oslg-qQHoNGY6L7U5ODgMVXxCBV526Mb

## GPT-2 Finetuning on Sentiment Classification

### Overview
- Compare performance of different text generation model on a sentiment detection task.
- For this, we will fine the text generation model GPT-2 on train data and report performance on the test data.
- Hence, we will also learn how to fine tune the TG models along wth how to apply these model to an example NLP task.
### Model
- Huggingface
### Dataset
- Próprio: https://www.dropbox.com/sh/oh6hbg5p9wyldfk/AAD2jw5EcP5oKaYbQ5FB55OGa?dl=0
"""

import locale
locale.getpreferredencoding = lambda: "UTF-8"

from google.colab import drive
drive.mount('/content/drive')

"""### Download and import packages"""

# uninstall
!pip uninstall -y wandb

# download
!pip install transformers

```

```
!pip install --upgrade accelerate

# import
import re
import json
import torch
import random
import pandas as pd
from tqdm import tqdm
from torch.utils.data import Dataset
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split

"""### Dataset load and prep functions"""

# Dataset class
class SentimentDataset(Dataset):
    def __init__(self, txt_list, label_list, tokenizer, max_length):
        # define variables
        self.input_ids = []
        self.attn_masks = []
        self.labels = []
        map_label = {0:'negative', 4: 'positive'}
        # iterate through the dataset
        for txt, label in zip(txt_list, label_list):
            # prepare the text
            prep_txt = f'<|startoftext|>Review: {txt}\nSentiment: {map_label[label]}<|endoftext|>'
            # tokenize
            encodings_dict = tokenizer(prep_txt, truncation=True,
                                       max_length=max_length, padding="max_length")
            # append to list
            self.input_ids.append(torch.tensor(encodings_dict['input_ids']))
            self.attn_masks.append(torch.tensor(encodings_dict['attention_mask']))
            self.labels.append(map_label[label])

    def __len__(self):
        return len(self.input_ids)

    def __getitem__(self, idx):
        return self.input_ids[idx], self.attn_masks[idx], self.labels[idx]

# Data load function
#def load_sentiment_dataset(tokenizer, random_seed = 1, file_path="/content/drive/My Drive/Colab/training.1600000.processed.noemoticon_edit.csv"):
def load_sentiment_dataset(tokenizer, random_seed = 1, file_path="/content/drive/My Drive/Colab/reviews_novo3.csv"):
```

```
    # load dataset and sample 10k reviews.
    #df = pd.read_csv(file_path, encoding='ISO-8859-1', header=None)
    #df = pd.read_csv(file_path, header=True, on_bad_lines='skip')
    df = pd.read_csv(file_path, header=None, on_bad_lines='skip')
    #df = df[[0, 5]]
    df = df[[0, 1]]
    df.columns = ['label', 'text']
    #df = df.fillna(0)
    #df['label'] = df['label'].astype(int)
    df = df.sample(7400, random_state=0)
    #df = df[df['text'].notna()]

    ##DEBUG
    #pd.set_option('display.max_rows', None)
    #print(df)

    def pick_first_n_words(string, max_words=250): # tried a few max_words, kept
    250 as max tokens was < 512
        #print(string)
        split_str = string.split()
        #print(split_str[:min(len(split_str), max_words)])
        return " ".join(split_str[:min(len(split_str), max_words)])

    df['text'] = df['text'].apply(lambda x: pick_first_n_words(x))

    print("############")

    # divide into test and train
    X_train, X_test, y_train, y_test = \
              train_test_split(df['text'].tolist(), df['label'].tolist(),
              shuffle=True, test_size=0.05, random_state=random_seed, stratify=df['
    label'])

    # get max length
    max_length_train = max([len(tokenizer.encode(text)) for text in X_train])
    max_length_test = max([len(tokenizer.encode(text)) for text in X_test])
    max_length = max([max_length_train, max_length_test]) + 10  #for special tokens
     (sos and eos) and fillers
    max_length = max(max_length, 300)
    print(f"Setting max length as {max_length}")

    # format into SentimentDataset class
    train_dataset = SentimentDataset(X_train, y_train, tokenizer, max_length=
    max_length)

    # return
    return train_dataset, (X_test, y_test)
```

```
122
123 """### Load model and tokenizer; Call data Prep"""
124
125 # import
126 from torch.utils.data import Dataset, random_split
127 from transformers import GPT2Tokenizer, TrainingArguments, Trainer, GPT2LMHeadModel
128
129 # model
130 model_name = "gpt2"
131 seed = 42
132
133 # seed
134 torch.manual_seed(seed)
135
136 # iterate for N trials
137 for trial_no in range(3):
138
139     print("Loading model...")
140     # load tokenizer and model
141     tokenizer = GPT2Tokenizer.from_pretrained(model_name, bos_token='<|startoftext
    |>',
142                                               eos_token='<|endoftext|>', pad_token=
    '<|pad|>')
143     model = GPT2LMHeadModel.from_pretrained(model_name).cuda()
144     model.resize_token_embeddings(len(tokenizer))
145
146     print("Loading dataset...")
147     train_dataset, test_dataset = load_sentiment_dataset(tokenizer, trial_no)
148
149     print("Start training...")
150     training_args = TrainingArguments(output_dir='results', num_train_epochs=10,
151                                       logging_steps=10, load_best_model_at_end=True,
152                                         save_strategy="no", evaluation_strategy="no",
     per_device_train_batch_size=2, per_device_eval_batch_size=2,
153                                       warmup_steps=100, weight_decay=0.01,
    logging_dir='logs')
154
155     Trainer(model=model, args=training_args, train_dataset=train_dataset,
156             eval_dataset=test_dataset, data_collator=lambda data: {'input_ids':
    torch.stack([f[0] for f in data]),
157                                                                    'attention_mask':
     torch.stack([f[1] for f in data]),
158                                                                    'labels': torch.
    stack([f[0] for f in data])}).train()
159
160     # test
161     print("Start testing...")
```

```
    # eval mode on model
    _ = model.eval()

    # compute prediction on test data
    original, predicted, all_text, predicted_text = [], [], [], []
    map_label = {0:'negative', 4: 'positive'}
    for text, label in tqdm(zip(test_dataset[0], test_dataset[1])):
        # predict sentiment on test data
        prompt = f'<|startoftext|>Review: {text}\nSentiment:'
        generated = tokenizer(f"<|startoftext|> {prompt}", return_tensors="pt").input_ids.cuda()
        sample_outputs = model.generate(generated, do_sample=False, top_k=50, max_length=512, top_p=0.90,
                temperature=0, num_return_sequences=0, pad_token_id=tokenizer.eos_token_id)
        pred_text = tokenizer.decode(sample_outputs[0], skip_special_tokens=True)
        # extract the predicted sentiment
        try:
            pred_sentiment = re.findall("\nSentiment: (.*)", pred_text)[-1]
        except:
            pred_sentiment = "None"
        original.append(map_label[label])
        predicted.append(pred_sentiment)
        all_text.append(text)
        predicted_text.append(pred_text)
    #transform into dataframe
    df = pd.DataFrame({'text': all_text, 'predicted': predicted, 'original': original, 'predicted_text': predicted_text})
    df.to_csv(f"result_run_{trial_no}.csv", index=False)
    # compute f1 score
    print(f1_score(original, predicted, average='macro'))

import os
os.makedirs("gpt2_10epochs", exist_ok=True)
model.save_pretrained("gpt2_10epochs")

input_text = "o aplicativo é ruim "

# predict sentiment on test data
prompt = f'<|startoftext|>Review: {input_text}\nSentiment:'
#generated = tokenizer(f"<|startoftext|> {prompt}", return_tensors="pt").input_ids
generated = tokenizer(f"<|startoftext|> {prompt}", return_tensors="pt").input_ids.cuda()
sample_outputs = model.generate(generated, do_sample=False, top_k=50, max_length=512, top_p=0.90,
                                temperature=0, num_return_sequences=0, pad_token_id=tokenizer.eos_token_id)
```

```
pred_text = tokenizer.decode(sample_outputs[0], skip_special_tokens=True)
print(pred_text)
# extract the predicted sentiment
try:
  pred_sentiment = re.findall("\nSentiment: (.*)", pred_text)[-1]
except:
  pred_sentiment = "None"

print(pred_sentiment)
```

APÊNDICE C – UTILIZAÇÃO DO MODELO GPT-3 ATRAVÉS DA API DA OPENAI

Código C.1 – Exemplo de *prompt* - GPT-3

```
# -*- coding: utf-8 -*-
"""colab_gpt3_inf

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1X0UnQ-pOMxNFzMprqIfhVHabzrMx8Jjc

## TCC29010

### João Leonardo Martins
### Orientador: Mário Noronha Neto
- Descrição.
### Modelo
- OpenAI gpt-3.5-turbo
### Dataset
- Próprio da OpenAI
### Download and import packages
"""

# uninstall
#!pip uninstall -y wandb

# download
#!pip install transformers

# install openai
!pip install openai
!pip install python-dotenv

# import
import re
import json
import torch
import random
import pandas as pd
```

```
from tqdm import tqdm
from torch.utils.data import Dataset
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split

#imports OpenAI
import openai
from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv()) # read local .env file

OPENAI_API_KEY = "INSIRA AQUI SUA API KEY"
openai.api_key = OPENAI_API_KEY

#Definição do modelo

def get_completion(prompt, model="gpt-3.5-turbo"):
    messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=0, # this is the degree of randomness of the model's output
    )
    return response.choices[0].message["content"]

review = """Precisava de uma droga de uma luminária legal para o meu quarto, e essa
     merda tinha armazenamento adicional e um preço até razoável. Chegou rápido,
    mas a corda da porra da luminária quebrou durante o transporte, e a Lumina,
    mandou uma nova, que não funcionou. """
review2 = """From the moment I received the bicycle, I encountered numerous issues
    that have made my cycling experience extremely unpleasant. The build quality of
     the bike is shockingly poor. The frame feels flimsy and lacks the durability I
     expected. It gives me constant worry that it might break or collapse while I'm
     riding, jeopardizing my safety."""
review3 = """This wheelbarrow is an absolute nightmare! It's a complete waste of
    money and time. The construction is shoddy, the handles are uncomfortable, and
    the overall quality is laughable. Save yourself the frustration and steer clear
     of this disaster."""
review4 = """Dieser Schubkarren ist ein absoluter Albtraum! Es ist eine totale
    Geldverschwendung und Zeitverschwendung. Die Konstruktion ist schlampig, die
    Griffe sind unbequem und die allgemeine Qualitt ist lcherlich. Spar dir den
    rger und halte dich von dieser Katastrophe fern."""
review5 = """

    """

prompt = f"""
Identifique os seguintes itens a partir do texto de review:
```

```
- Sentimento (positivo ou negativo)
- O avaliador parece estar bravo? (verdadeiro ou falso)
- Item comprado pelo avaliador
- Fabricante ou vendedor do item

O texto de reviews está delimitado por tripla crase. \
Formate a resposta com as seguintes chaves: \
"Sentimento", "Bravo", "Item" and "Marca".
Se não conseguir obter alguma informação, preencha o valor\
 como "Desconhecido" Faça a resposta o mais curta possível.
O valor de "Bravo" é Verdadeiro ou Falso.

A resposta será no idioma de '''{review}'''.

Review text: '''{review}'''
"""
response = get_completion(prompt)
print(response)
```

Código C.2 – Comparador para o modelo GPT-3

```
# -*- coding: utf-8 -*-
"""comparador_gpt2_gpt3.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1ltI6tF97GUiHYROXtYqYqYnysfu12Lzb
"""

import locale
locale.getpreferredencoding = lambda: "UTF-8"

from google.colab import drive
drive.mount('/content/drive')
# uninstall
!pip uninstall -y wandb
# download
!pip install transformers
!pip install --upgrade accelerate
!pip install openai
!pip install python-dotenv

# import
import re
import json
import torch
import random
```

```
import pandas as pd
from tqdm import tqdm
from torch.utils.data import Dataset
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
import re
import json
import torch
import random
import os
import time
#import pandas as pd
from tqdm import tqdm
from torch.utils.data import Dataset
#from sklearn.metrics import f1_score
#from sklearn.model_selection import train_test_split
#imports OpenAI
import openai
from dotenv import load_dotenv, find_dotenv

_ = load_dotenv(find_dotenv()) # read local .env file

#configure aqui sua apikey da openai
openai.api_key = 'SUA_API_KEY'

def get_completion(prompt, model="gpt-3.5-turbo"):
    messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=0, # this is the degree of randomness of the model's output
    )
    return response.choices[0].message["content"]

def predict(text):
    prompt_text = f"""
        return 'negative' if the sentiment is negative or return 'positive' \
        if the sentiment is positive for the following text: '''{text}'''
        """
    response = get_completion(prompt_text)

    return response

print(predict(str("Gostaria de saber o que houve cm a questão de conseguir olhar o
    caminho que o motoboy está percorrendo quando se inicia a nossa entrega!? Antes
     ficava mais fácil aguardar, pq víamos quando estava chegando, agora não
    aparece mais. É algum bug ou fizeram a retirada?").lower()))
```

```
72
73 # Dataset class
74 class SentimentDataset(Dataset):
75     def __init__(self, txt_list, label_list, tokenizer, max_length):
76         # define variables
77         self.input_ids = []
78         self.attn_masks = []
79         self.labels = []
80         map_label = {0:'negative', 4: 'positive'}
81         # iterate through the dataset
82         for txt, label in zip(txt_list, label_list):
83             # prepare the text
84             prep_txt = f'<|startoftext|>Review: {txt}\nSentiment: {map_label[label
    ]}<|endoftext|>'
85             # tokenize
86             encodings_dict = tokenizer(prep_txt, truncation=True,
87                                        max_length=max_length, padding="max_length")
88             # append to list
89             self.input_ids.append(torch.tensor(encodings_dict['input_ids']))
90             self.attn_masks.append(torch.tensor(encodings_dict['attention_mask']))
91             self.labels.append(map_label[label])
92
93     def __len__(self):
94         return len(self.input_ids)
95
96     def __getitem__(self, idx):
97         return self.input_ids[idx], self.attn_masks[idx], self.labels[idx]
98
99 # Data load function
100 def load_sentiment_dataset(tokenizer, random_seed = 1, file_path="/content/drive/My
     Drive/Colab/reviews_final.csv"):
101     # load dataset and sample 10k reviews.
102     #df = pd.read_csv(file_path, encoding='ISO-8859-1', header=None)
103     #df = pd.read_csv(file_path, header=True, on_bad_lines='skip')
104     df = pd.read_csv(file_path, header=None, on_bad_lines='skip')
105     #df = df[[0, 5]]
106     df = df[[0, 1]]
107     df.columns = ['label', 'text']
108     #df = df.fillna(0)
109     #df['label'] = df['label'].astype(int)
110     df = df.sample(7400, random_state=0)
111     #df = df[df['text'].notna()]
112
113     ##DEBUG
114     #pd.set_option('display.max_rows', None)
115     #print(df)
116
```

```
117     def pick_first_n_words(string, max_words=250): # tried a few max_words, kept
        250 as max tokens was < 512
118         #print(string)
119         split_str = string.split()
120         #print(split_str[:min(len(split_str), max_words)])
121         return " ".join(split_str[:min(len(split_str), max_words)])
122 
123     df['text'] = df['text'].apply(lambda x: pick_first_n_words(x))
124 
125     print("#############")
126 
127     # divide into test and train
128     X_train, X_test, y_train, y_test = \
129               train_test_split(df['text'].tolist(), df['label'].tolist(),
130               shuffle=True, test_size=0.05, random_state=random_seed, stratify=df['
        label'])
131 
132     # get max length
133     max_length_train = max([len(tokenizer.encode(text)) for text in X_train])
134     max_length_test = max([len(tokenizer.encode(text)) for text in X_test])
135     max_length = max([max_length_train, max_length_test]) + 10  #for special tokens
         (sos and eos) and fillers
136     max_length = max(max_length, 300)
137     print(f"Setting max length as {max_length}")
138 
139     # format into SentimentDataset class
140     train_dataset = SentimentDataset(X_train, y_train, tokenizer, max_length=
        max_length)
141 
142     # return
143     return train_dataset, (X_test, y_test)
144 
145 # import
146 from torch.utils.data import Dataset, random_split
147 from transformers import GPT2Tokenizer, TrainingArguments, Trainer, GPT2LMHeadModel
148 
149 # model
150 model_name = "gpt2"
151 seed = 42
152 
153 # seed
154 torch.manual_seed(seed)
155 
156 # iterate for N trials
157 for trial_no in range(3):
158 
159   print("Loading model...")
```

```
160   # load tokenizer and model
161   tokenizer = GPT2Tokenizer.from_pretrained(model_name, bos_token='<|startoftext|>'
        ,
162                                             eos_token='<|endoftext|>',
        pad_token='<|pad|>')
163   model = GPT2LMHeadModel.from_pretrained(model_name).cuda()
164   model.resize_token_embeddings(len(tokenizer))
165
166   print("Loading dataset...")
167   train_dataset, test_dataset = load_sentiment_dataset(tokenizer, trial_no)
168
169 #Realizando consultas ao modelo treinado da OpenAI:
170 predicted2 = []
171 for x in range(370):
172   try:
173     res = predict(str(test_dataset[0][x]).lower())
174   except:
175     print("skipping at " + x)
176   predicted2.insert(x, res)
177   #predicted2.append(res)
178   time.sleep(1)
179   print(x)
180
181 print(test_dataset[0][369])
182 print(predicted2[369])
183
184 #Calculando a diferença entre o test_dataset e o predict_gpt3:
185 i = 0
186 for x in range(370):
187   #print(x)
188   if (test_dataset[1][x] == 4 and predicted2[x] != 'positive') or \
189    (test_dataset[1][x] == 0 and predicted2[x] != 'negative'):
190     #print(test_dataset[0][x])
191     i = i + 1
192     #print(x)
193
194 aproveitamento = 100 - ((100*i)/370)
195 print("\n\nHouve " + str(i) + " difenças..\nEquivalente a um aproveitamento\
196  de " + str(aproveitamento) + "%")
197
198 # compute prediction on test data
199 original, predicted, all_text, predicted_text = [], [], [], []
200 map_label = {0:'negative', 4: 'positive'}
201 for text, label in tqdm(zip(test_dataset[0], test_dataset[1])):
202         # predict sentiment on test data
203         prompt = f'<|startoftext|>Review: {text}\nSentiment:'
204         generated = tokenizer(f"<|startoftext|> {prompt}", return_tensors="pt").
```

```
    input_ids.cuda()
205         sample_outputs = model.generate(generated, do_sample=False, top_k=50,
    max_length=512, top_p=0.90,
206                 temperature=0, num_return_sequences=0, pad_token_id=tokenizer.
    eos_token_id)
207         pred_text = tokenizer.decode(sample_outputs[0], skip_special_tokens=True)
208         # extract the predicted sentiment
209         try:
210             #pred_sentiment = re.findall("\nSentiment: (.*)", pred_text)[-1]
211             pred_sentiment = predict(str(pred_text).lower())
212             print(pred_sentiment)
213         except:
214             pred_sentiment = "None"
215         original.append(map_label[label])
216         predicted.append(pred_sentiment)
217         all_text.append(text)
218         predicted_text.append(pred_text)
219 #transform into dataframe
220 df = pd.DataFrame({'text': all_text, 'predicted': predicted, 'original': original,
    'predicted_text': predicted_text})
221 df.to_csv(f"result_run_{trial_no}.csv", index=False)
222 # compute f1 score
223 print(f1_score(original, predicted, average='macro'))
```