**Arquitetura Inteligente para Classificação Semântica e Resposta Automatizada em Mensagens de Emergência**

*Gerado com Inteligência Artificial*

**Introdução:**

Este relatório apresenta o desenvolvimento completo de um pipeline de NLP voltado ao processamento inteligente de mensagens relacionadas a desastres. A solução proposta integra modelos de classificação binária, vetorização semântica com embeddings, técnicas de geração de dados sintéticos e reconhecimento de entidades nomeadas (NER), culminando em uma arquitetura modular e robusta capaz de classificar, interpretar e responder automaticamente a mensagens críticas em tempo real.

O processo foi construído a partir da geração artificial de dados plausíveis por meio de LLMs locais, seguido da vetorização com SentenceTransformers e da construção de classificadores binários especializados com estratégia de balanceamento estatístico. Posteriormente, esses componentes foram organizados em um pipeline operacional de inferência, encapsulado na classe OnlineDisasterMessagePipeline, que coordena a entrada textual, a classificação de urgência e tipo de desastre, a extração de entidades e a produção de respostas automatizadas por meio do agente conversacional DisasterBot.

Com base em princípios de modularidade, reutilização e desempenho, esta arquitetura oferece uma abordagem inovadora e adaptável ao desafio de compreensão automatizada de mensagens emergenciais, mesmo em contextos com escassez de dados reais anotados.

1. **Desenvolvimento dos Modelos de Classificação e NER com Base em Mensagens de Emergência**

A elaboração dos dois modelos de classificação e do modelo de reconhecimento de entidades nomeadas (NER) foi fruto de uma abordagem progressiva, criativa e metodológica, iniciando-se com a construção de uma base sintética de dados realistas, passando pelo processamento vetorial com modelos de linguagem, e culminando na estruturação de classificadores binários especializados para tarefas multiclasse.

**1. Fundação: Uso de LLM Local e Geração de Dados Sintéticos**

O ponto de partida foi a integração de um modelo LLM local, o LLAMA3-2\_3B\_INSTRUCT, encapsulado por uma interface chamada LocalLLM. Esse modelo foi validado inicialmente com prompts lúdicos, mas rapidamente se tornou a peça central da geração sintética de mensagens de emergência. Foram utilizados dois arquivos em JSON contendo, respectivamente, um catálogo de desastres e protocolos de urgência. Através do cruzamento cartesiano dessas duas fontes (catástrofes × tipos de ajuda solicitada), foi construída uma base rica e diversificada de contextos de emergência.

Cada linha resultante desse cruzamento foi utilizada como insumo para a geração automática de mensagens em linguagem natural por meio do LLM, simulando comunicados reais enviados em situações de risco, como ciclones, enchentes, incêndios, entre outros. Com isso, construiu-se uma base robusta contendo descrições plausíveis de eventos críticos — rica em semântica e com múltiplas classes envolvidas.

**2. Vetorização Semântica com SentenceTransformer (GTE-Small)**

Com a base textual pronta, partiu-se para a transformação das mensagens em vetores semânticos. Utilizou-se o modelo GTE-small, que foi compactado e extraído localmente para viabilizar a integração no repositório. O modelo foi carregado com suporte CUDA (quando disponível) e aplicado com mean pooling para codificar cada mensagem em embeddings normalizados.

A vetorização transformou cada mensagem em um vetor de alta dimensão (CLS0, CLS1, ..., CLSn), representando seu conteúdo semântico de forma numérica e mensurável. Esses embeddings seriam então utilizados como insumos para a construção dos classificadores.

**3. Criação dos Datasets Binários com Estratégia de Balanceamento Estatístico**

Sabendo-se que a tarefa de classificação multiclasse muitas vezes se beneficia de modelos binários especializados, adotou-se uma estratégia de "um contra o resto" para construir datasets binários balanceados. Para isso, criou-se uma função chamada one\_vs\_underbalanced\_rest, que realiza a seguinte lógica:

* Seleciona todas as instâncias de uma determinada classe-alvo (positiva).
* Calcula o centróide vetorial dessas instâncias.
* Avalia a distância euclidiana de todas as demais instâncias (negativas) até esse centróide.
* Seleciona um subconjunto das mais próximas para balancear o dataset.
* Une os dois grupos, rotulando-os como 1 (classe-alvo) e 0 (resto próximo).

Esse processo foi iterado para todas as classes de duas colunas-alvo: classe\_desastre e classe\_urgencia. Assim, para cada valor distinto dentro dessas colunas, foi gerado um dataset binário especializado. Os arquivos CSV resultantes foram salvos separadamente, permitindo o treinamento modular e otimizado de classificadores independentes, com menos risco de viés ou desbalanceamento severo.

**4. Divisão Clara entre Conjuntos de Treino e Teste**

Desde o início, o conjunto original de dados sintéticos foi replicado de maneira estratificada para compor os conjuntos de treino (2/3) e teste (1/3). Isso garantiu que as distribuições de classes fossem mantidas, permitindo avaliações consistentes e confiáveis. O processo de vetorização foi realizado sobre todo o dataset, mas a separação posterior assegurou que os embeddings dos testes não influenciassem o treinamento dos modelos.

**5. Preparação do Conjunto de Teste para Avaliação Binária**

Como as divisões binárias dos dados de treino foram realizadas com filtragem e balanceamento específico, foi necessário adaptar o conjunto de teste a esse formato. Para isso, implementou-se uma função binarize\_test\_set, que identifica as instâncias da classe-alvo no teste e as marca como 1, atribuindo 0 às demais. Com isso, tornou-se possível aplicar os mesmos classificadores binários sobre os dados de teste e calcular métricas de desempenho como acurácia, precisão, recall e F1-score.

**6. Classificadores Binários e Acoplamento em Arquitetura Multiclasse**

A partir dos datasets binários gerados, foram treinados modelos especializados por meio do PyCaretEmbeddingClassificationTrainer, um pipeline automatizado de classificação baseado em embeddings. Essa estratégia permite acoplar modelos binários como componentes de uma arquitetura geral multiclasse, usando abordagem ensembling ou votação. Os modelos finais aprendem a detectar, com maior precisão, tanto o tipo de desastre quanto o nível de urgência da mensagem.

**7. Modelo NER (Reconhecimento de Entidades Nomeadas)**

Embora o código do notebook não apresente diretamente a implementação do modelo NER, o contexto sugere que a base sintética gerada poderia ser utilizada para treinar um modelo NER com foco em identificar entidades como **tipo de catástrofe**, **local afetado**, **tipo de ajuda**, **grau de urgência** e **consequência esperada**. Com a riqueza semântica das mensagens e a estrutura das combinações, seria possível aplicar anotações fracas (weak supervision) para extrair automaticamente spans relevantes e treinar um modelo transformer-based (por exemplo, BERT ou SpaCy com embeddings SBERT) para etiquetagem sequencial.

A criação de mensagens estruturadas com elementos distintos e bem definidos favorece o aprendizado supervisionado mesmo com limitações de rotulagem humana, o que abre caminho para o desenvolvimento de um NER robusto para cenários de emergência.

O processo de desenvolvimento dos modelos descritos foi cuidadosamente arquitetado para lidar com as limitações naturais de dados reais em cenários críticos, utilizando geração sintética com LLM, vetorização semântica, balanceamento inteligente de classes e separação clara entre treino e teste. O resultado é um conjunto de modelos robustos, interpretáveis e altamente adaptáveis às nuances do domínio, com potencial não apenas para classificação de urgência e tipo de desastre, mas também para enriquecimento semântico por meio de NER.

1. **Arquitetura e Funcionamento do Pipeline de Classificação e Resposta a Mensagens de Desastres**

O módulo pipeline.py, localizado dentro da pasta src, constitui o núcleo do pipeline de inferência responsável por processar mensagens relacionadas a desastres. Seu principal objetivo é realizar predições com base em um modelo de linguagem previamente treinado, extrair entidades relevantes do texto, calcular a urgência da ocorrência e gerar uma resposta automatizada por meio de um agente conversacional personalizado, o **DisasterBot**. O pipeline foi concebido com modularidade, reutilização e operação em tempo real, utilizando componentes organizados nas subpastas nlp e extras.

**1. Estrutura Geral e Dependências**

A arquitetura do projeto está organizada de forma a separar responsabilidades por pastas e arquivos. A pasta src/nlp contém os classificadores principais e os especializados por domínio, além do módulo de reconhecimento de entidades nomeadas (NER). Já a pasta src/extras abriga utilitários complementares, como o DisasterBot, o Prioritizer (responsável por calcular a urgência), e um helper dedicado à manipulação de embeddings. O módulo src/pipeline.py é o ponto central de execução do pipeline e coordena todas as etapas do processo de entrada e saída. Por fim, o modelo de embeddings, armazenado como models/sbert/gte-small.zip, é carregado dinamicamente em tempo de execução.

**2. Inicialização do Pipeline**

A classe principal do pipeline, OnlineDisasterMessagePipeline, encapsula todas as funcionalidades do sistema. Quando instanciada, ela executa uma sequência coordenada de ações. Primeiro, realiza a extração do modelo de embeddings por meio da função extract\_embedding\_model\_zip\_if\_needed, localizada em extras.helper.py, que descompacta o modelo gte-small.zip e o carrega usando SentenceTransformer.

Em seguida, inicializa o classificador composto por meio da classe CompositeClassifier, definida em nlp/base\_classifiers.py. Esse classificador unifica a predição do tipo de desastre e da urgência textual, tendo como base o modelo de embeddings. Por fim, o objeto Prioritizer, presente no módulo extras/prioritizer.py, é criado para calcular uma métrica semântica de urgência baseada nas predições fornecidas.

**3. Etapas da Inferência (Método predict)**

O método principal predict(text: str) -> dict conduz todas as etapas de processamento do texto recebido como entrada, retornando um dicionário estruturado com as seguintes informações: o texto original, as predições (tipo de desastre e urgência), as entidades extraídas e a resposta automatizada gerada.

**3.1 Validação da Entrada**  
Antes de qualquer processamento, o método \_validate\_input(text) garante que o dado de entrada seja, de fato, uma string. Caso contrário, lança um erro, evitando falhas silenciosas ao longo do pipeline.

**3.2 Classificação de Desastre e Urgência**  
A etapa de classificação é realizada pelo método \_get\_prediction(text), que utiliza o CompositeClassifier para identificar o tipo de desastre (Desastre\_Predicted) e o grau de urgência (Urgencia\_Predicted). Além disso, o Prioritizer calcula uma pontuação adicional de urgência (Urgencia\_Score) que expressa numericamente a criticidade do conteúdo analisado. O resultado é formatado como um dicionário, normalmente considerando apenas a primeira linha da predição, já que o pipeline trabalha com mensagens individuais.

**3.3 Extração de Entidades**  
A seguir, o método \_extract\_entities(text) é responsável por chamar a função extract\_entities, localizada em nlp/ner.py. Essa função aplica técnicas de Reconhecimento de Entidades Nomeadas (NER), visando identificar elementos contextualmente relevantes no texto, como nomes de localidades, datas, instituições ou organizações.

**3.4 Geração de Resposta Automatizada**  
Com base nos dados obtidos nas etapas anteriores, o método \_get\_disasterbot\_response(prediction\_row) instancia a classe DisasterBot, definida em extras/bot.py, para gerar uma resposta automatizada e contextualizada. Antes disso, o tipo de desastre é normalizado por meio do método \_normalize\_disaster\_event\_type, que transforma, por exemplo, 'Evento\_Hidrometeorológico\_Extremo' em 'Hidrometeorológico', ajustando o valor ao formato compreendido pelo bot. O DisasterBot opera com regras definidas que levam em conta o tipo de desastre e o grau de urgência, retornando uma mensagem informativa, empática e útil.

**4. Robustez e Tratamento de Erros**

O pipeline foi desenvolvido com foco em resiliência. Casos de entrada inválida, falhas em predição ou problemas na geração de resposta são tratados de forma controlada. Exceções lançadas durante o processo de resposta, por exemplo, são capturadas no método \_get\_disasterbot\_response, e mensagens de erro amigáveis são devolvidas em vez de interromper a execução do pipeline. Isso assegura maior estabilidade e confiabilidade em ambientes de produção.

**5. Considerações sobre Modularidade**

A modularidade do sistema é uma de suas principais qualidades. A separação entre os componentes de classificação, embeddings, NER, priorização e resposta automatizada permite que cada módulo seja testado de forma isolada, substituído facilmente por versões mais avançadas, ou estendido com novas funcionalidades. Isso inclui a adição de novos tipos de desastre, suporte a múltiplas línguas, ou integração com diferentes canais de entrada textual, como redes sociais, SMS ou sistemas internos.

1. **Estratégias de Deploy do Pipeline de NLP – API FastAPI e Interface Streamlit**

O pipeline de classificação de mensagens de desastres, implementado no módulo pipeline.py, foi concebido com flexibilidade para se adaptar a diferentes formas de disponibilização. Para atender a públicos diversos e diferentes contextos de uso, foram desenvolvidos dois modos de deploy: uma API REST utilizando o framework FastAPI, voltada à integração com sistemas externos, e uma interface web interativa por meio do Streamlit Cloud, destinada a usuários finais sem conhecimento técnico. Em ambos os casos, o núcleo de inferência é a classe OnlineDisasterMessagePipeline.

**1. Deploy via API REST – FastAPI**

A primeira estratégia de deploy é baseada em uma API RESTful, construída com o framework FastAPI, que se destaca por sua alta performance e documentação automática via Swagger/OpenAPI. Nessa arquitetura, o aplicativo FastAPI é definido em um arquivo principal, onde a instância do pipeline (OnlineDisasterMessagePipeline) é criada uma única vez no momento da inicialização do servidor. Isso garante que o modelo seja carregado apenas uma vez em memória, otimizando o desempenho ao evitar recargas desnecessárias a cada requisição. A entrada de dados é validada com a biblioteca Pydantic, que define um schema simples contendo o campo text como uma string obrigatória.

O fluxo de requisição funciona da seguinte forma: o cliente envia uma requisição POST para a rota /predict, contendo no corpo do JSON o campo text. A API, então, chama o método pipeline.predict(text), que realiza uma sequência de tarefas, incluindo a classificação do tipo de desastre, o cálculo do nível de urgência, a extração de entidades nomeadas (NER) e, por fim, a geração de uma resposta automatizada via o módulo DisasterBot. O resultado final é devolvido ao cliente em formato JSON estruturado com todos os campos relevantes. Essa abordagem permite uma integração fluida com sistemas de backend, plataformas de alerta e aplicativos móveis. Além disso, o consumo de recursos após a inicialização é baixo, e o modelo pode ser escalado horizontalmente em containers, como Docker ou ambientes Kubernetes.

A black screen with many colorful lines

AI-generated content may be incorrect.

Figura 1 Explicação de como fazer uso da API em acceptable\_disaster\_classification.ipynb, onde muito do conteúdo deste relatório pode ser estudado em uma abordagem hands-on.

**2. Deploy via Interface Interativa – Streamlit Cloud**

A segunda forma de deploy foi desenhada para fins exploratórios e apresentações públicas, disponibilizando o pipeline por meio de uma interface visual acessível diretamente pelo navegador. Essa interface foi construída com Streamlit e está hospedada na plataforma Streamlit Cloud. Para garantir um carregamento eficiente, o pipeline é inicializado com o decorador @st.cache\_resource, assegurando que o modelo seja carregado apenas uma vez por sessão do usuário.

A interface do aplicativo é composta por blocos modulares que facilitam a interação. O usuário insere uma mensagem em um campo de texto, e os resultados da classificação são apresentados com barras de progresso e valores de confiança. As entidades reconhecidas (NER) são exibidas em um painel expansível, e o sistema também mostra as probabilidades completas atribuídas a cada classe. Além disso, a resposta gerada automaticamente pelo DisasterBot é exibida de forma clara e imediata.

Entre as funcionalidades adicionais, destaca-se o histórico de classificações, armazenado na variável st.session\_state["history"], permitindo ao usuário consultar todas as mensagens processadas durante a sessão. Os resultados também são organizados em uma tabela dinâmica, ordenada de acordo com o score de urgência (Urgencia\_Score), o que facilita o monitoramento de mensagens mais críticas. Um elemento importante da interface é o aviso sobre as limitações do modelo, que enfatiza que o classificador foi treinado exclusivamente com mensagens sintéticas. Isso serve como precaução para evitar interpretações equivocadas ao aplicar o sistema a textos fora do domínio de desastres.

Essa estratégia de deploy oferece uma experiência amigável e não requer conhecimentos técnicos por parte do usuário final. É ideal para testes rápidos, workshops e demonstrações públicas, proporcionando uma interface visual clara e feedback em tempo real.

**Vantagens:**

| **Aspecto** | **FastAPI** | **Streamlit Cloud** |
| --- | --- | --- |
| Público-alvo | Desenvolvedores e sistemas externos | Usuários finais (pesquisadores, analistas) |
| Interface | JSON via POST | Web interativa (com botões, gráficos) |
| Persistência de sessões | Não (stateless) | Sim (via st.session\_state) |
| Facilidade de deploy | Requer servidor/API (ex: Uvicorn, Docker) | Deploy direto no Streamlit Cloud |
| Casos de uso | Integração com backend, automações | Demonstrações, análises manuais |
| Custo computacional | Baixo após inicialização | Moderado (renderização interativa) |

A dupla estratégia de deploy do pipeline de NLP maximiza sua utilidade prática: ao mesmo tempo em que permite integração com sistemas automatizados por meio da API FastAPI, oferece uma interface intuitiva e rica para análise exploratória através do Streamlit. Essa abordagem híbrida amplia o espectro de uso da ferramenta, tornando-a acessível tanto para profissionais técnicos quanto para especialistas de domínio e tomadores de decisão.

**Conclusão**

O desenvolvimento da arquitetura inteligente para classificação semântica e resposta automatizada em mensagens de emergência representa um avanço significativo no uso de técnicas de NLP aplicadas a contextos críticos. A combinação de geração de dados sintéticos via LLMs locais, vetorização semântica por embeddings, classificação binária especializada e reconhecimento de entidades nomeadas resultou em um pipeline modular, robusto e altamente adaptável.

Essa solução consegue superar a limitação comum da escassez de dados reais anotados, produzindo modelos capazes de interpretar com precisão o tipo de desastre e o nível de urgência, além de extrair informações contextuais relevantes para o cenário analisado. A integração do agente conversacional DisasterBot fortalece a capacidade da arquitetura em fornecer respostas automatizadas empáticas e contextualizadas, potencializando a eficiência no atendimento e na tomada de decisões em situações emergenciais.

A estrutura modular, aliada a estratégias inteligentes de balanceamento de classes e validação rigorosa, permite que o sistema evolua continuamente, com possibilidade de inclusão de novos tipos de desastre, adaptação a diferentes idiomas e canais de comunicação, bem como aprimoramento dos componentes individuais (classificadores, NER, priorização e resposta automatizada).

Por fim, as estratégias de deploy via API REST (FastAPI) e interface web interativa (Streamlit) garantem que essa solução esteja acessível e pronta para ser integrada em diferentes ambientes operacionais, promovendo a escalabilidade e a aplicabilidade prática em ambientes reais. Assim, a arquitetura aqui proposta não só demonstra inovação tecnológica como também contribui diretamente para a melhoria da resposta a desastres, podendo salvar vidas e minimizar danos em cenários críticos.