Relatório de Pré-Processamento

Inteligência Artificial

Marcello Victor Ferreira e Silva
Maria Eduarda Schimidt

Introdução

Este relatório apresenta o passo a passo seguido na realização do trabalho, que tem como objetivo desenvolver e comparar estratégias de deteção de emoções em textos com técnicas de I.A.

Como base de dados, foram utilizados dois arquivos .csv obtidos do repositório SemEval2025-task11 [1], contendo textos e suas respectivas classificações para seis emoções distintas: raiva (anger), nojo (disgust), medo (fear), alegria (joy), tristeza (sadness) e surpresa (surprise).

A tarefa principal é um problema de classificação multi-label, onde cada texto pode expressar múltiplas emoções simultaneamente. Para cada emoção, a classificação é binária (0 ou 1), indicando a ausência ou presença da emoção no texto analisado.

O trabalho compara o desempenho da detecção de emoções atráves de duas técnicas, uma clássica e outra moderna. A clássica utiliza dois algoritmos de aprendizado de máquina (Regressão Logística e Redes Neurais MLP) treinando-os com representações vetoriais dos textos geradas através da técnica Word2Vec, enquanto a moderna consiste em fornecer os dados de treino à LLMs e verificar, com os dados de teste, o desempenho. Ao final do relatório, é feita a avaliação dos resultados e comparação entre as técnicas.

Para não poluir o relatório, os blocos de código serão referenciados pelo número do seu tópico no notebook emotion_detection.ipynb.

Técnica Clássica

A detecção de sentimentos com uma abordagem clássica, como já mencionado na introdução, utiliza dois algoritmos de aprendizado de máquina. Entretanto, para utilizar nossos dados como entrada para os algoritmos, precisamos que eles estejam vetorizados e, para obter os vetores, foi feito um pré-processamento nos dados como descrito nos capítulos abaixo. Cada capítulo dessa seção da técnica clássica tem exatamente o mesmo número de seção do notebook *emotion_detection.ipynb*.

1. Exploração Inicial - Tópico **1** no notebook

Após carregar os arquivos, cada um em um dataframe pandas e, antes de aplicar qualquer transformação ou tratamento, foram utilizadas algumas funções para visualizar a estrutura dos dados e compreender quais modificações seriam necessárias. Além disso, para facilitar o tratamento dos dados, foi feita a união dos dois dataframes em um só, já que foi observado que o formato dos dois é o mesmo.

2. Limpeza dos textos - Tópico **2** no notebook

A etapa de limpeza de textos, que utilizou da estrutura encontrada em sentiment.py [2], que por sua vez é uma adaptação de Twitter Sentiment Analysis [3], é a etapa onde os elementos de ruído e incosistências presentes nos textos são eliminados, criando um formato padronizado e simplificado.

A função *clean_text()* recebe texto como parâmetro e é responsável por aplicar expressões regulares para tratar diversos tipos de elementos textuais. Alguns exemplos de transformações são tornar o texto minúsculo, substituir números pelo token “0”, substituição de nomes de URL, username e e-mail pelos tokens "URL", "USERNAME" e "EMAIL" respectivamente, substituição de cada emoji por um token que o represente, entre outras retiradas e substituições de string. Com isso, os textos são transormados para um formato padronizado que permitirá a continuidade do processamento.

3. Tokenização - Tópico **3** no notebook

Na etapa de tokenização, os textos já em formato padronizado se tornam uma lista de termos individuais, que são os tokens. Como os textos já haviam sido tratados e

passaram a possuir uma estrutura “comportada”, bastou que fosse utilizado o método “split()” para converter cada frase em um array de tokens.
Essa etapa é necessária para preparar os dados para a próxima fase de vetorização, onde cada palavra individual precisa ser mapeada para uma representação numérica. O Word2Vec, técnica escolhida para a vetorização, requer que o texto seja fornecido como uma sequência de tokens discretos, não como uma string contínua.

4. Vetorização - Tópico **4** no notebook

Agora, já com as listas de tokens referentes a cada um dos textos iniciais, é feita a conversão dos tokens em representações numéricas que possam ser processadas pelos algoritmos de aprendizado de máquina, ou seja, em vetores.

Essa conversão é feita com o treinamento de um modelo Word2Vec, assim como em sentiment.py [2], utilizando as listas de tokens como entrada. Nesse treinamento, o Word2Vec cria embeddings vetoriais capazes de capturar relações semânticas entre palavras, fazendo com que palavras com significados similares ou que aparecem em contextos parecidos possuam representações vetoriais próximas no espaço multidimensional, permitindo ao modelo interpretar algo próximo a semânticas do texto.

Como nosso objetivo é classificar o sentimento das frases, foi implementada a função word_vector() que tem como objetivo calcular a representação vetorial de um texto completo a partir dos vetores individuais de suas palavras. Esta função recebe como entrada a lista de tokens de um texto e o tamanho do vetor desejado, inicializando um vetor zero com as dimensões apropriadas. Em seguida, ela itera sobre cada palavra do texto, buscando seu vetor correspondente no modelo Word2Vec treinado e somando-o ao vetor acumulador. Para cada palavra encontrada no vocabulário, um contador é incrementado para fazer o cálculo da média.

5. Treinamento dos Modelos de Classificação - Tópico **5** no notebook

Com os vetores equivalentes a cada texto já calculados, é possível utilizar algoritmos de aprendizado de máquina para treinar os modelos. Os dados foram divididos em conjuntos de treino e teste utilizando a função train_test_split() da biblioteca scikit-learn, com uma proporção de 80% para treinamento e 20% para teste. Os mesmo conjuntos de treinamento e teste foram também salvos em

arquivos csv, para serem utilizados na técnica moderna e ser possível fazer uma comparação entre os resultados.

Para lidar com a natureza multi-label do problema, foi utilizado o MultiOutputClassifier do scikit-learn, que adapta classificadores binários para problemas multi-label treinando um modelo independente para cada emoção. Foram implementados dois algoritmos distintos: Regressão Logística, que representa uma abordagem linear, configurada com 1000 iterações máximas para garantir convergência; e Rede Neural MLP, que oferece maior capacidade de capturar relações não-lineares nos dados, configurada com duas camadas ocultas de 100 e 50 neurônios respectivamente, e 500 iterações máximas.

Ao avaliar os modelos, cada emoção teve sua avaliação individual, utilizando as métricas acurácia, calculada com a função *accuracy_score* e a precisão, que utilizamos a função classification_report para calcular por não ter encontrado outra função que retorna somente a precisão.

Em relação à acurácia, nos dois modelos foi possível obter valores acima de 80% para as emoções *disgust, fear, sadness* e *surprise*, sendo que *anger* e *joy* ficaram um pouco abaixo mas num patamar ainda aceitável.

Entretanto, quando se trata da precisão, observamos que as emoções *disgust, fear* e *surprise* tiveram valor 0 de precisão, o que nos levou a fazer mais uma exploração nos dados para entender o possível motivo. Ao analisar a distribuição das emoções no dataset, vimos que as exatas emoções que tiveram precisão 0 também são as três que menos aparecem, tendo uma diferença considerável de aparições em relação às três mais presentes. Com isso, inferimos que, por não aparecer muitas vezes nos dados de treino, o modelo tende a achar difícil um exemplo de texto conter *disgust, fear* ou *surprise* e, por isso, deve classificar todos os casos de teste como não contendo essas emoções. Isso não afeta tanto a acurácia, já que na maior parte dos casos essa classificação de fato estará correta. Entretanto, quando se trata de precisão, que consiste na porcentagem de verdadeiros positivos classificados, o modelo sempre erra.

Para verificar se esta teoria era uma verdade, foi feita a contagem de classificações 0 e 1 para cada emoção, resultando no que já era esperado para as emoções de precisão 0.

Técnica Moderna

Para a execução da técnica moderna, utilizamos LLMs pré-treinados, que já possuem grande conhecimento linguístico adquirido durante o treinamento em grandes volumes de texto. Nessa etapa, foram inicialmente enviados os dados de treinamento (gerados após o pré processamento da técnica clássica), para que o modelo entenda a tarefa, junto com o comando do que deve ser feito. Em seguida, enviamos os dados de teste sem os labels, pedindo como resposta a classificação das emoções presentes naqueles textos e, assim, calculamos as métricas de avaliação.

Prompt utilizado:

```
Vou te enviar um dataset com textos e as emoções contidas em cada texto. As possibilidades de emoções são: anger (raiva), disgust (nojo), fear (medo),  joy (alegria), sadness (tristeza), surprise (surpresa), sendo que as opções de resposta para cada emoção, por texto, são 0 ou 1 (não contém ou contém). Cada texto pode conter múltiplas emoções, apenas uma, ou nenhuma. Entenda quais os padrões de detecção de cada emoção para esses textos e, em seguida, te enviarei um dataset sem os labels para testar sua capacidade de detecção de emoções.
```

1. OpenAI GPT-4.1

Ao enviar o dataset de treinamento para o ChatGPT, ele fez uma análise exploratória nos dados, compreendeu que os textos já foram tratados e processados para e embeddings e fez algumas observações, como o desbalanceamento na quantidade de vezes que cada emoção aparece e algumas correlações como *disgust* e *sadness* aparecerem juntas em uma quantidade considerável de amostras, não sendo muito alto, mas sendo a maior correlação encontrada.

No ChatGPT foi observado um comportamento mais “preguiçoso”, já que só fazia a classificação de algumas amostras e, para que continuasse fazendo, foi necessário solicitar (quase implorar) para que continuasse.

Após algumas tentativas de que o GPT continuasse a predição, reparamos que ele já não estava mais utilizando os valores reais de teste, mas apenas ignorando os embeddings e escolhendo um valor aleatório para os labels. Sendo assim, seguimos apenas com o Claude.

2. Anthropic Claude Sonnet 4

Assim como no modelo anterior, ao enviar o dataset para o Claude, ele identificou um desequilibrio no número de aparições entre as emoções e algumas correlações mais evidentes. Entretanto, Ao enviar os de teste, o Claude foi mais eficiente para classificar e, como eu pedi, enviou um arquivo .csv com sua classificação de labels, permitindo que eu pudesse comparar com os valores reais e avaliar com as métricas de acurácia e precisão.

Emoção	Acurácia	Precisão
anger	0.440	0.351
disgust	0.971	0.000
fear	0.932	0.000
joy	0.286	0.266
sadness	0.855	0.000
surprise	0.773	0.046

Comparação

Após a execução das técnicas clássica e moderna, pudemos observar que a ténica clássica, utilizando Word2Vec combinado com Regressão Logística e MLP, é mais consistente em relaçao aos resultados, sendo mais previsível e interpretável. Apesar de ter apresentado limitações na classificação de emoções menos frequentes no dataset, resultando em precisão zero para *disgust, fear* e *surprise* devido ao desbalanceamento dos dados - os modelos clássicos mantiveram permitiram análise detalhada das falhas, possibilitando identificar que o problema estava relacionado à distribuição desigual das classes no conjunto de treinamento.

Em comparação com a técnica clássica, a moderna com LLMs apresentou alguns desafios na utilização dos modelos comprometeram em parte a avaliação. O GPT demonstrou comportamento inconsistente, realizando classificações parciais e, posteriormente, gerando predições aparentemente aleatórias que ignoravam os embeddings fornecidos. Embora o Claude Sonnet 4 tenha se mostrado mais “disposto” na execução da tarefa, a falta de controle sobre os processos internos de treinamento e classificação dos LLMs dificultou a interpretação dos resultados. Mesmo assim, foi possível ver que, nos dois métodos, a quantidade baixa de ocorrências de algumas emoções foi um fator crucial para baixas acurácias e precisões 0.

Referências:

1. SemEval2025-task11: https://github.com/emotion-analysis-project/SemEval2025-task11
2. sentiment.py: https://colab.research.google.com/drive/1sY3y2SgecN9h6Tdw5mZd3-L0QTMOF3wK
3. Twitter Sentiment Analysis - word2vec, doc2vec: https://www.kaggle.com/code/nitin194/twitter-sentiment-analysis-word2vec-doc2vec/notebook