Data and Resources (folder containing Google colab code, datasets, and the video recording. Also contains our trained model RoBERTa and our tokenizer which is NOT in our code zip file due to large size.)

Video Submission (direct link to the video submission)

Slang Detector

Project by: Jasmine Dong (jqd3), Derek Huang (djh328), Eileen Liang (el596)

AI Keywords: Natural Language Processing (NLP), Machine Learning, Assistive Technology, Sentiment Analysis

Application Setting: Analyzing words or phrases for the potential of slang

Part 1: Project Description

Project Goals and Evolution

Initially, our team wanted to make an assistive tool for people to interpret text for underlying tone. We decided that doing sentiment analysis on words or phrases that were input would be good for individuals with autism and others who need help decoding emotions in texts. We foresaw this to be especially useful since mostly everyone would benefit from receiving real-time assistance in decoding nuanced expressions during the Digital Age.

In brief, we wanted to use NLP techniques and libraries to handle recognizing patterns and expressions that could be commonly misunderstood with a focus on handling informal language, which includes abbreviations, slang, and emojis. We planned to utilize datasets of words/phrases labeled with positive and negative connotations with various machine learning algorithms to see how useful they would be on sentiment analysis. None of the members of our team had previously taken CS 4740 Natural Language Processing (NLP) or any other courses that provided instruction handling large language models. Despite this, we were motivated by the challenge and the opportunity to learn about an unfamiliar area of AI.

Our motivation for this was to build on existing AI models' inability to accurately handle slang language. When testing the version of ChatGPT freely available at the time of this proposal, we found that it would frequently mistranslate new slang words. For older informal words (such as established curse words or words that were new slang many years ago), it seemed to work decently well. However, since language is always evolving and the rise of TikTok and other social media platforms started rapidly popularizing "new" slang words, ChatGPT was not as good at identifying or defining them.

A model specifically trained for slang detection presents a viable solution to many challenges. For instance, individuals using AI-driven customer service bots might face misunderstandings if they use contemporary slang. Similarly, content moderation systems could misinterpret the tone or meaning of user-generated content, leading to inappropriate actions or responses. Slang usage patterns can also identify age demographics on social media platforms, as younger generations are the primary users of contemporary slang. AI models can help researchers and marketers better understand and segment their target audiences. Additionally, this can help gauge the political sentiment of young voters, which can compel campaigners to be more resonant and relatable to younger demographics.

Given our limited background knowledge of NLP, we first did some research to familiarize ourselves with the fundamental concepts of NLP. This research included understanding the processes of data gathering and cleaning, tokenization, the various stages involved in model training, and different evaluation metrics necessary for testing the efficacy of our model, such as accuracy, precision, recall, and F1 score. We planned to explore NLP libraries and frameworks like spacCY and NLTK and use existing datasets like Sentiment140.

Upon receiving feedback on this project proposal, we decided to expand on this idea and make it more intricate than training existing models on existing datasets. Based on this, our group fleshed out another proposal to create an intergenerational slang translator that would take in pieces of text containing informal or slang language and translating it into clear, plain English in the hopes of facilitating more cross-generational communication.

In order to enhance the novelty and functionality of our project, we decided to integrate Urban Dictionary (a crowdsourced online dictionary for slang words and phrases), Merriam-Webster, and ChatGPT's API into our system. We chose Urban Dictionary because it is the most up-to-date source of slang usage. The integration aims to first analyze the presence of all slang words in a piece of text by fine tuning a pre-trained AI model, searching for the nuanced definition on Urban Dictionary through its API and then using ChatGPT to rewrite the original text into a more understandable version.

Core AI Aspects of the Project

Our approach includes creating and refining our own datasets, using a pretrained model (RoBERTa), designing and implementing custom preprocessing, tokenization, and fine-tuning strategies tailored specifically for slang detection, using gradient descent algorithms, and cross-validation evaluation methods.

Our dataset consists of example sentences containing a wide variety of slang term usage (Urban Dictionary's example sentences) as well as formal sentences from Merriam-Webster. To build a robust and diverse dataset, we compiled a list of 16,581 words from three different text files: one containing potentially slang words, one containing potentially obscene words, and one containing the most commonly used English words. This ensured our dataset covered a broad spectrum of language use. We then used these words to scrape example sentence entries from Urban Dictionary and Merriam-Webster using the BeautifulSoup library. Each example sentence was labeled with a 1 if it contained slang and a 0 if it did not. Any Merriam-Webster example sentence containing a slang badge was labeled as containing slang, while all Urban Dictionary entries were labeled as 1 for slang. Everything was saved into ".txt" files in a format that could be easily digested by our model. We also refined the data before training to get rid of characters that caused errors during the preprocessing of data, such as extra tab characters.

We then used RoBERTa's tokenizer to tokenize the sentences and padded them to maximum length so that they can be batch processed. The tokenized data was then converted into tensors for compatibility with PyTorch, a deep learning framework.To manage the tokenized data, we created a custom dataset class and utilized a data loader for efficient batching during training. We modified RoBERTa's classification head to predict whether a given text sample contains slang. Fine-tuning involved training the model on our labeled dataset and adjusting the model's parameters for optimal performance. We used the AdamW optimization algorithm, a variant of the Adam optimizer. We chose AdamW because it adjusts learning rates dynamically, making the training process more efficient and stable and minimizes cross-entropy loss. Additionally, the weight decay term helps prevent overfitting. We utilized the cross-entropy loss function, suitable for classification tasks, to measure the difference between the predicted probabilities and the true labels. This guided the optimizer in minimizing this difference, thereby improving the model's accuracy in detecting slang.

We then set up a training loop to iterate over the data. We trained the model on six datasets of example sentences and evaluated its performance after 1, 3, and 6 training sessions for accuracy, precision, recall, and F1 score using the 'sklearn.metrics' library.

For a UI, we considered implementing a Google Chrome Extension that would take in highlighted text and run it through our project and output the translated version to replace the highlighted text. However, it proved to be difficult since it didn't have easy access to our product. Therefore, we decided to implement a basic UI using Jupyter widgets. First, we needed to get user input and preprocess the sentence similarly to how we preprocessed our dataset. This involved tokenizing the user's input and restricting the maximum length of user input. Finally, we passed the user's input into our predict_slang method, which returned the predicted classification of the user's input. We then used an ipywidget to create a button to open the user input box, and they also had the option to close the input dialog box as well with a close button.

From there, the user could type into the input box and then click the submit button. Our model would predict the classification of the user's input and then print it onto the user's screen right below the submit button. Additionally, one of our goals was to then convert any sentences identified as slang into plain English for the user to more easily understand. However, due to limitations with our model, we were only able to classify whether a sentence contained slang. To satisfy this feature requirement given our time constraints, we called the OpenAI API and gave it a prompt to convert the user's sentence into plain English. The returned result was also printed to the user's screen, right below our model's output. As of May 2024, the recent release of a new free version of ChatGPT (GPT 4-o) has made the model more accurate in recognizing slang and translating the examples into plain English. This is unfortunate for our project because it's essentially what we created, but at least that means the plain English translations from the API are more likely to be accurate.

If given more time, we would likely implement this feature ourselves by training a separate model to identify what words make a sentence contain slang. This would also involve acquiring a new dataset where each word is tagged with a 0 or 1 to represent if that word is a slang word. We could accomplish this by checking to see if each word in the sentence has a definition in Urban Dictionary and then tagging it with 1 if such a definition exists (binary classification). Along with fine-tuning a pre-trained model for token classification, we would use a suitable loss function, such as categorical cross-entropy, for classification tasks. Finally, we would get the definition of any tokens tagged as slang and then call the Urban Dictionary API for the definition and substitute the definition into the user's input sentence for a translation into plain English.

Part 2: Evaluation

How accurate can our model detect slang and how precise and reliable are its predictions?

The success of our slang interpretation project was evaluated based on the accuracy of the identification of slang sentences (based on the definitions of slang from Urban Dictionary). We partitioned our dataset gathered from Urban Dictionary into a training set, validation set, and test set. About 80% of our total dataset was the training set, 10% the validation set, and 10% the test set. The training set and validation set was used to adjust the parameters of the models.

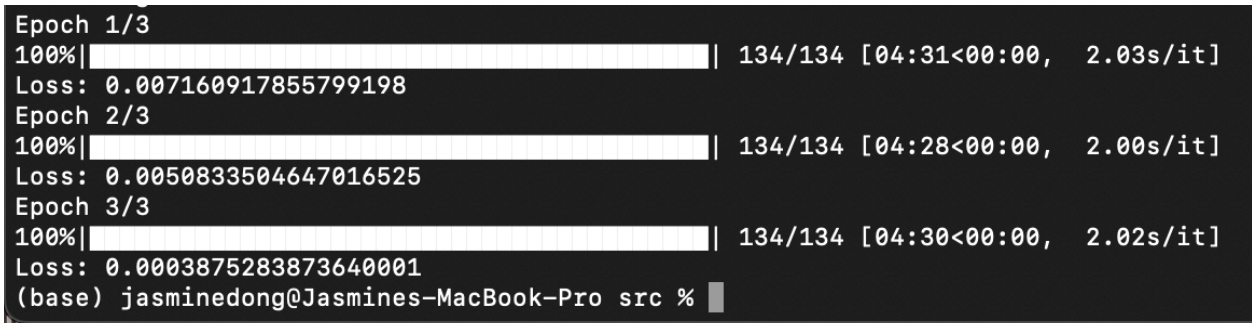
	Accuracy	Precision	Recall	F1
1 dataset	0.9378	0.9718	0.9143	0.9422
3 datasets	0.9552	0.9701	0.9484	0.9591
6 datasets	0.9587	0.9746	0.9502	0.9623

The four metrics – accuracy, precision, recall, and F1 scores – from the sklearn.metrics library were used for their following traits:

- **accuracy:** functionally the same as the Jaccard score for binary classification: measures how closely the predicted labels match the correct labels. It is calculated by comparing the intersection of the predicted and actual slang-or-not labels to their union.
- **precision:** a calculation that portrays “the ability of the classifier not to label as positive a sample that is negative”; For our model, precision indicated how often we correctly predicted slang sentences out of all sentences marked as slang.
- **recall:** a calculation that portrays “the ability of the classifier to find all the positive samples”; mathematically, it is the ratio of true positives to the sum of true positives and false negatives. For our model, recall indicated how well it identified all actual slang sentences.

- **F1 score:** this score can be "interpreted as a harmonic mean of the precision and recall." It is particularly useful when we need to balance the trade-off between precision and recall. A higher F1 score indicates a well-rounded performance. (scikit-learn).

Looking at the different values of accuracy, precision, recall, and F1 scores as we trained on more datasets, we can see that all scores increased. We think this means our model was successful because an increasing accuracy score indicates that prediction-labeled and correctly-labeled sets became more similar, an increasing precision score indicates that the proportion of relevant results to irrelevant results increased, an increasing recall score indicates that relevant results were increasingly returned, and an increasing F1 score indicates that the the performance between precision and recall were balanced.



```
Epoch 1/3
100%|██████████████████████████████████████████████| 134/134 [04:31<00:00,  2.03s/it]
Loss: 0.007160917855799198
Epoch 2/3
100%|██████████████████████████████████████████████| 134/134 [04:28<00:00,  2.00s/it]
Loss: 0.0050833504647016525
Epoch 3/3
100%|██████████████████████████████████████████████| 134/134 [04:30<00:00,  2.02s/it]
Loss: 0.0003875283873640001
(base) jasminedong@Jasmines-MacBook-Pro src %
```

The screenshot above captures the progress of our model training process on the sixth dataset. The significant reduction in loss from ~0.00716 to ~0.00038 over three epochs indicates that the model is effectively learning and significantly improving its performance on the training data. The lower loss values mean that the predicted probabilities are increasingly aligning with true labels, showcasing the model's improved performance.

How do different elements of our system design contribute to overall success?

We evaluated the impact of various design choices, such as the choices in data collection and preprocessing techniques, the use of the AdamW optimizer, and the choice of loss functions.

We conducted ablation studies by modifying one element at a time and observing changes in the evaluation metrics.

Initially, the data scraping process was projected to take hours due to the server request wait times. To address this, we employed the asyncio, aiohttp, and aiofiles libraries to make our server requests and local file I/O operations asynchronous. This approach significantly sped up the process. However another problem arose: both Urban Dictionary and Merriam-Webster began disconnecting our IP address due to the high volume of requests. We overcame this issue by splitting our dataset into chunks of 3,000 words each, resulting in six smaller datasets of example sentences. Additionally, since Merriam-Webster includes many slang words and phrases, we checked the word badges for labels such as "slang," "vulgar," or "obscene" and tagged those entries accordingly. When we didn’t account for the fact that Merriam-Webster also included some slang words, there were more errors and lower accuracy.

In our project, we experimented with various optimizers and found the AdamW optimizer to be the best option. AdamW offers dynamic learning rate adjustment and incorporates weight decay, which helps prevent overfitting. Initially, we tried using Stochastic Gradient Descent (SGD), but this approach did not yield satisfactory results due to the lack of adaptive learning rate adjustments. Without the benefits of AdamW’s features, our model’s performance dropped by approximately 5% across all key metrics.

Furthermore, for our classification task, the cross-entropy loss function proved to be highly effective. We tested other loss functions, such as mean squared error, but observed a notable decrease in model performance. Cross-entropy loss is well-suited for classification tasks as it measures the performance of a classification model whose output is a probability value between 0 and 1, making it a better fit for our task compared to mean squared error, which is more appropriate for regression tasks.

References

Parsing example sentences from each word/phrase entry from these sources:

- Merriam-Webster English dictionary (words or phrases used in non slang sentences): https://www.merriam-webster.com/
- Urban Dictionary: https://www.urbandictionary.com/

Text sources:

- words-slang.txt by Christopher Wellons
- bad-words.txt by Luis von Ahn
- google-10000-english-no-swears.txt by Josh Kaufman

Software:

- RoBERTa: https://arxiv.org/abs/1907.11692

API's:

- ChatGPT API: https://platform.openai.com/docs/guides/text-generation

Other resources:

- scikit documentation: https://scikit-learn.org/stable/index.html
- AdamW documentation: https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html