CS918 - Sentiment Analysis Coursework

University id: 2283791

Abstract

The analysis of tone and sentiment in short texts remains a pivotal challenge within Natural Language Processing (NLP), offering widespread applications from enhancing chatbot responsiveness to dissecting user feedback across various platforms. This study zeroes in on the task of sentiment classification within Twitter feeds, utilizing the SemEval 2017 Task 4 dataset as a testbed. Our methodology adopts a hierarchical approach to model exploration, initiating with baseline classifiers like Naive Bayes and Logistic Regression to establish foundational performance metrics. Progressively, we delve into more sophisticated architectures, including Recurrent Neural Networks (RNNs), with a specific focus on Long Short-Term Memory (LSTM) networks and an enhanced LSTM variant incorporating attention mechanisms. The culmination of our investigation involves fine-tuning the Bidirectional Encoder Representations from Transformers (BERT) model to our dataset, aiming to juxtapose its efficacy against traditional and neural network-based approaches. This sequential exploration not only underscores the evolution of sentiment analysis techniques but also provides a comprehensive performance analysis across a spectrum of models in capturing Twitter sentiment.

1 Introduction

Sentiment analysis represents a quintessential classification challenge within the realm of natural language processing (NLP), wherein texts are algorithmically assigned to predefined categories based on their conveyed sentiment. This process can be binary, distinguishing between positive and negative sentiments, or multi-class, identifying a range of emotional states or opinions. A notable application of sentiment analysis is within the domain of Twitter data, where the objective is to discern the emotional undertones of user-generated tweets. The complexity of this task is augmented by the unique characteristics of social media discourse, which often diverges from the standard conventions of language.

The brevity enforced by Twitter's character limit compels users to employ abbreviations and a plethora of emojis and emoticons, which serve as compact but expressive means of conveying emotions. Furthermore, the frequent use of hashtags, slang, and occasionally vulgar language contributes to the complexity of sentiment analysis in this context. These elements not only deviate from conventional English usage but also introduce ambiguity, making the computational interpretation of sentiment a more daunting task. Consequently, the peculiarities of social media text challenge the robustness of sentiment analysis algorithms, necessitating the development of sophisticated techniques that can adeptly navigate the nuanced linguistic landscape of platforms like Twitter.

2 Datasets

The dataset we are working with consists of 5 files, one for training, one for development, and three for testing.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Throughout the model development and assessment, we will use only the training and development datasets. We use the training data for training the models and the development data for evaluating its performance.

Figure 1 shows that the five datasets have similar sentiment distribution, and that the labels are imbalanced with the majority of tweets having the label neutral.

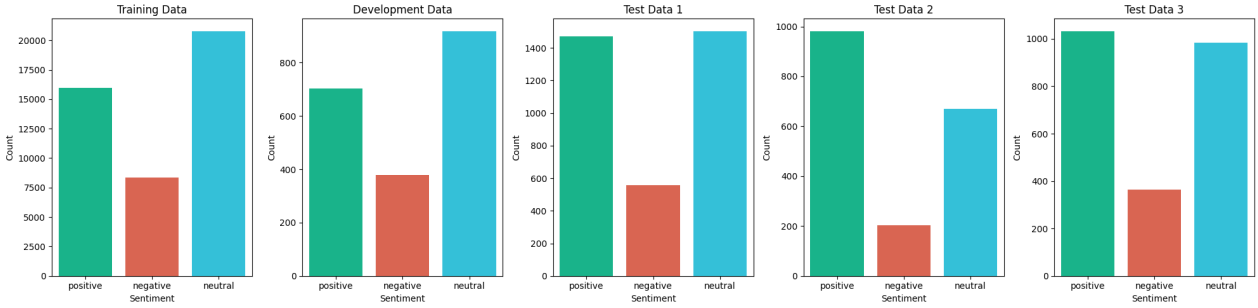


Figure 1: Sentiment distribution for the five different datasets

This shows us that we are dealing with an imbalanced data and this could affect the model's ability to distinguish between **neutral** and **negative** classes.

3 Data Pre-processing

Given that our data contains raw social media text, it requires extensive pre-processing and cleaning before it can be fed into a classifier.

In this section, we will detail each preprocessing step along the additional Python packages used for performing it.

Step 1: Initial Preprocessing

1. **Lowercasing:** Converts the tweet text to lowercase to ensure consistency, as the same word with different cases (e.g., "Hello" vs. "hello") would be treated as different tokens otherwise.
2. **Normalization:** Abbreviations such as 'w/' and 'w/o' are standardized to 'with' and 'without' respectively, making the text more formal and uniform.
3. **Removing User Mentions:** User mentions in the format `@username` are removed because such information is irrelevant to the sentiment classification.
4. **Removing URLs:** URLs are stripped from the tweet using a regular expression, as they typically do not contribute to the content's textual analysis.
5. **Removing Digits:** Digits and ordinal representations (e.g., 1st, 2nd) are removed, focusing the analysis on textual content.
6. **Whitespace Cleanup:** Extra spaces are reduced to a single space, and leading/trailing spaces are trimmed.
7. **Removing Retweets:** The abbreviation "RT" indicating a retweet is removed.
8. **Special Case Substitution:** Specific terms like "ac/dc" are substituted (e.g., with "acdc") to ensure they are treated as single tokens.

Step 2: Handling Contractions

We then expand common English contractions (e.g., transforming "isn't" to "is not") using a contractions fixer. This standardizes the text for analysis. We use the contractions Python package for this step.

Step 3: Tokenization

The tweet is tokenized into individual words or symbols, which are then processed individually. We use NLTK's **TweetTokenizer** because it's specifically designed to work with tweets (handle hashtags, emoticons, etc) unlike regular tokenizers which are designed to work with regular text.

Step 4: Token-Level Cleaning and Transformation

Each token undergoes several checks and transformations:

- Emoticons and emojis are translated into text descriptions. Instead of removing the emojis and emoticons altogether, we replace them with their corresponding text. We use Ekphrasis and emoji packages for this.
- Hashtags are segmented into constituent words, which are then cleaned. While we could consider hashtags as noise and should be removed, in many cases they are the most important words in a tweet, and in many cases they can be a single word that the user decided to highlight or emphasize by using the hashtag symbol. Therefore, we keep them and we segment them. Segmenting means replacing hashtags like **#messiisthegoat** with **messi is the goat**.
- Tokens found in a predefined slang dictionary are replaced with their full forms. Common slang terms like **LOL** and **afaik** are replaced with their corresponding English sentences.
- General cleaning removes special characters and single characters from tokens. This is done to ensure our data consists of only English words.

Step 5: Removal of Stopwords and Final Cleaning

Cleaned tokens are further filtered to remove stopwords and single-character tokens. This step ensures that only meaningful content is retained for analysis. Stopwords are very common words like **the**, **and**, **is**, etc. And they are not distinctive features for classifying the text.

Step 6: Reconstruction

The final step involves joining the cleaned and filtered tokens back into a single string, representing the preprocessed tweet ready for NLP tasks.

4 Exploratory Analysis and Correlation study

Before building classification models, we briefly explore the data to understand and collect any important insights.

Figure 5 shows the raw tweet length by sentiment, and figure 6 shows the tweet length after applying the pre-processing pipeline.

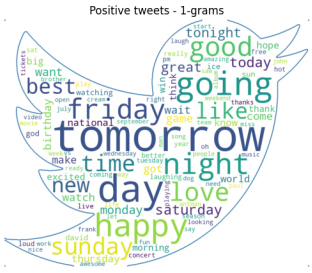
There is no apparent relation between the length of the tweet and its sentiment, and on general the majority of tweets have length in range 100-125.

However, after applying the pre-processing pipeline, tweets become shorter and more clustered in the range 50-100.

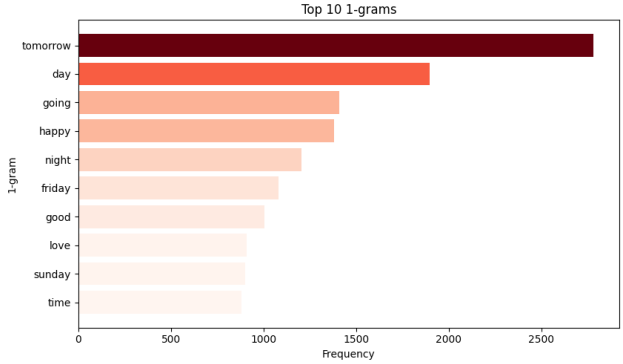
What we can conclude from these two figures is that our tweets are generally short (this is an important observation for RNN-based models) and that applying pre-processing shorten the tweets.

We try to understand the **content** of the tweets by their sentiment aiming at the question: **What makes a tweet positive, negative, or neutral?** Essentially, the words that a user write along with emojis and hashtags are what convey what the user want to communicate.

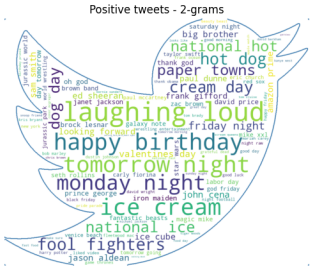
We create unigrams, bigrams, and trigrams for each sentiment class. We visualize the top 10 using barchart, and the top 100 using wordclouds. We utilize the WordCloud for generating wordclouds.



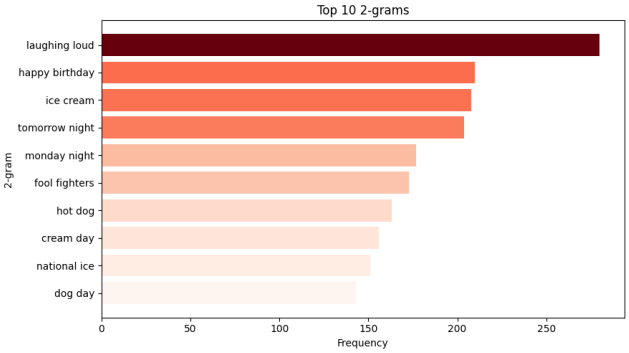
(a) Word cloud for positive unigrams



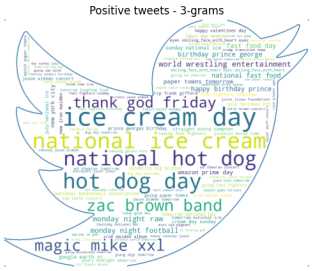
(b) Bar chart for positive unigrams



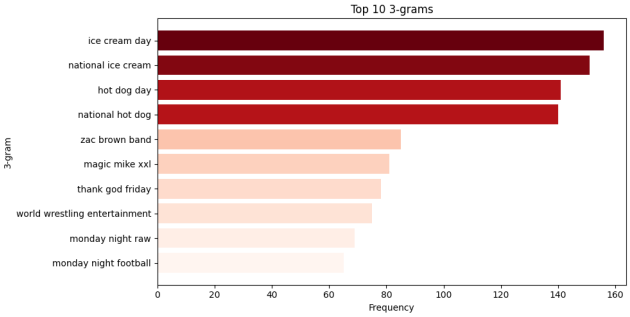
(c) Word cloud for positive bigrams



(d) Bar chart for positive bigrams

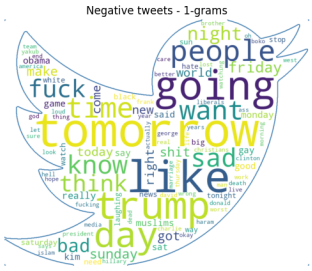


(e) Word cloud for positive trigrams

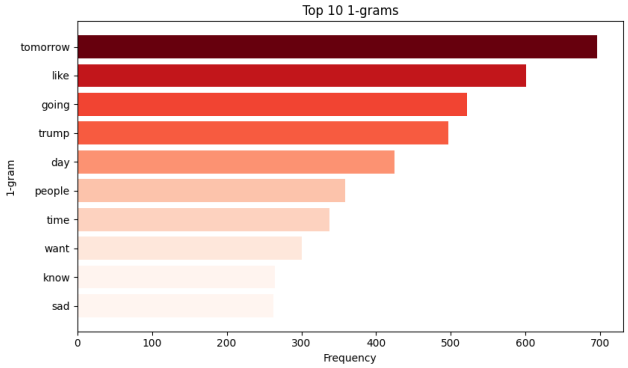


(f) Bar chart for positive trigrams

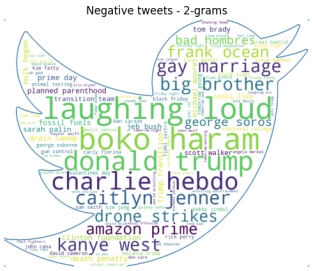
Figure 2: Visualizations for positive sentiment analysis



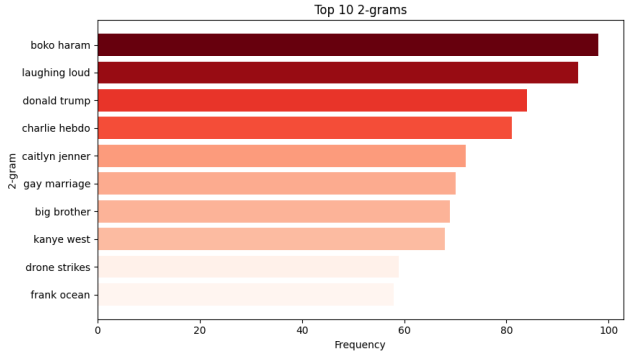
(a) Word cloud for negative unigrams



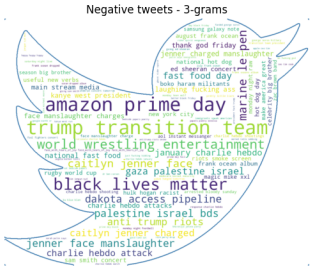
(b) Bar chart for negative unigrams



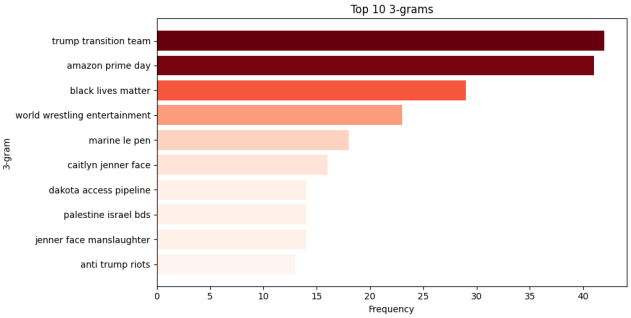
(c) Word cloud for negative bigrams



(d) Bar chart for negative bigrams

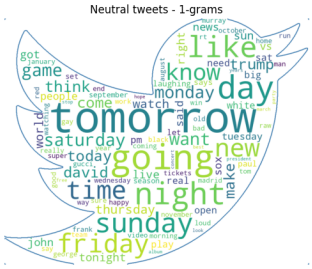


(e) Word cloud for negative trigrams

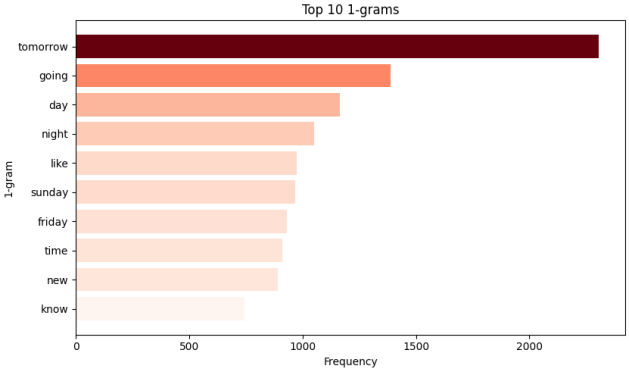


(f) Bar chart for negative trigrams

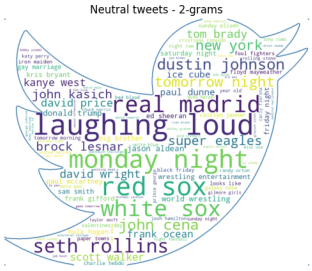
Figure 3: Visualizations for negative sentiment analysis



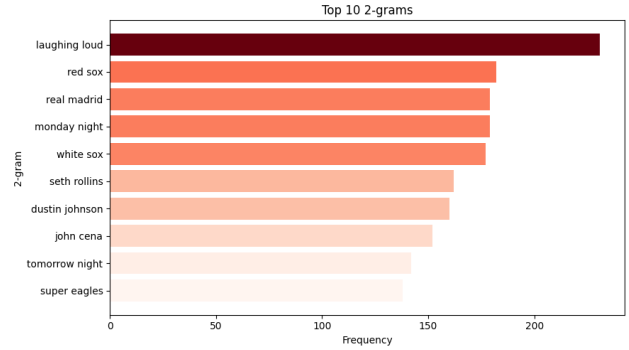
(a) Word cloud for neutral unigrams



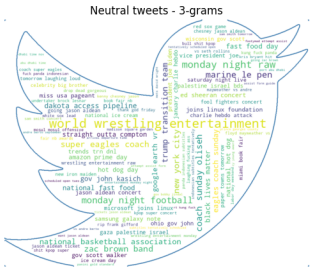
(b) Bar chart for neutral unigrams



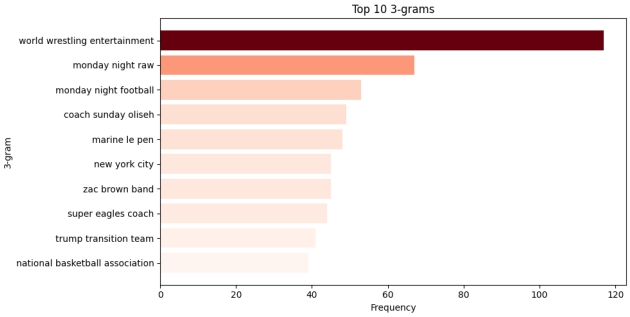
(c) Word cloud for neutral bigrams



(d) Bar chart for neutral bigrams



(e) Word cloud for neutral trigrams



(f) Bar chart for neutral trigrams

Figure 4: Visualizations for neutral sentiment analysis

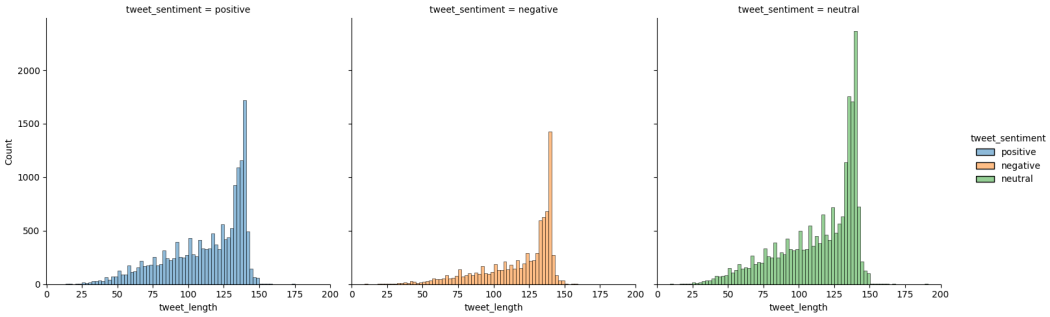


Figure 5: Tweet length distribution by sentiment

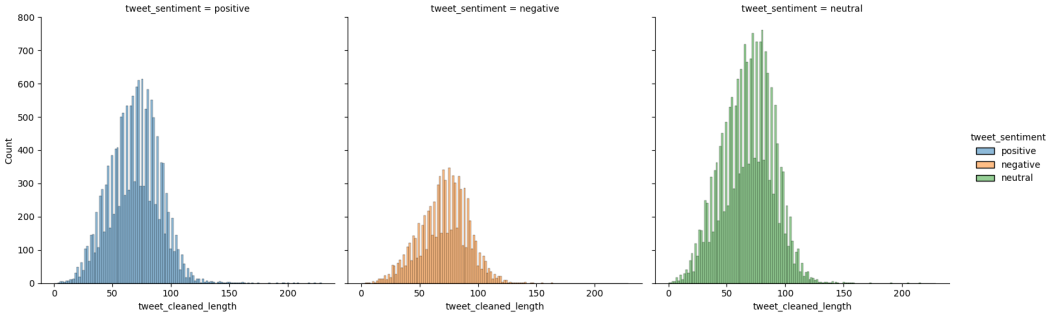


Figure 6: Tweet length after pre-processing distribution by sentiment

5 Performance evaluation

model_name	test1	test2	test3	model_size	epoch_train_time
Naïve Bayes	0.48	0.48	0.45	NaN	NaN
Logistic Regression	0.57	0.55	0.53	NaN	NaN
SVM	0.56	0.55	0.52	NaN	NaN
LSTM	0.60	0.59	0.57	3131403	3 seconds
LSTM + Attention	0.61	0.61	0.59	3132004	3 seconds
BERT (raw tweets)	**0.72**	**0.72**	**0.70**	109484547	6 minutes, 15 seconds
BERT (cleaned tweets)	0.67	0.68	0.64	109484547	6 minute, 10 seconds