Neural Approaches for Predicting the Sentiment of Messages in a Twitch.tv Dataset

Xajavion “Jay” Seabrum, Scott Kinder
{Xajavion.Seabrum, Scott.Kinder}@colorado.edu

University of Colorado Boulder

Abstract. Semantic analysis on Twitch datasets is a growing research topic. Neural network approaches to the problem have yet to be explored in depth, despite advancements in Natural Language Processing. In this research, several neural network approaches are applied to previously studied Twitch data to compare their results to classical machine learning techniques. These results are highly impacted by the choice of word embeddings used, and therefore embeddings will be a focus of the research as well.

1 Introduction

Internet culture has been pervasive in mainstream in the modern era. With this culture, many neologisms are generated and created by a largely young audience. One place where these neologisms arise is Twitch.tv where, weekly, millions of viewers and many thousands of streamers alike log in and interact with one another. Peculiarly idiosyncratic to Twitch is the proliferation and ubiquitous use of global Twitch emotes across all channels on the platform. Viewers’ messages with these emotes seem like another language despite the fact that the emotes are embedded in and synthesize with the natural language that surrounds them. Consequentially, understanding, updating, and integrating their meaning is critical for future neural NLP approaches and applications that seek to use Twitch chat logs as a source of corpora. The overall problem of using Twitch chat as a corpus is challenging not only because of Twitch emotes but also because: a single emote can constitute an entire utterance meaning, readily available libraries that specifically pre-process Twitch text do not yet exist, and the interactivity between chatters and streamers cause inherent noisiness in data collection.

To the best of our literature search and understanding, prior work with regards to predicting the sentiment analysis of Twitch have only used traditional machine learning (ML) approaches for the task directly [1, 2]. The earlier approach [1] performed ML and NLP techniques on a dataset developed by [3], whereas [2], using the same dataset, figured out how to incorporate different embedding techniques to increase performance while still relying on traditional ML schema. The present paper instead seeks to address the dataset from [3] from a deep learning perspective.

Specifically, this paper seeks to try Long Short-Term Memory (LSTMs) and Gated Recurrent Units (GRUs) neural architectures on this dataset as they have the promise to provide better performance over the ML approaches employed in [2,3], as evidenced by recent advancements in Natural Language Processing. Furthermore, another goal is to capture different data modalities in a fuller model, such as metadata about the chat, which includes the channel name where the comment was posted and whether the viewer is a moderator or a subscriber. This metadata could help leverage any insights gained from the output of the neural model to more accurately predict the sentiment of a message.

2 Related Work

Emoji Studies
Studies have focused on gathering data and analyzing data from Instagram in a multimodal image-comment neural perspective [4] or from Twitter from an ML NLP paradigm [5,6] to understand emojis and their sentiment and to predict when such emojis will be used. The limitation here is that it has been shown that Twitch emotes and chat structure differ drastically from typical emoji usage and other social media websites [7]. This present paper therefore differs from [4–6] in that while Twitch emotes and emojis seemingly should share similar ground, their linguistic uses and contexts are too disparate.

Twitch Chat Structure
One study details how the structure and language used in Twitch messages departs from other types of language use online because it is solely livestreamed and astronomically interactive [7]. Another study builds upon this idea to detect how Twitch emotes are linguistically used to replace certain letters in a chatter's message to obfuscate their toxic speech acts [8]. This paper's work differs in the sense that, while the task is to take into account the uniqueness that comes with Twitch streams' chats, determining the exact context in which emotes are used in toxic chat messages is not of interest; rather, interest lies in performing an overall Twitch sentiment classification for chat messages.

Prior Semantic Analyses on Twitch Data
In terms of model architecture, prior approaches to semantic analysis for Twitch datasets have generally relied on traditional ML techniques, such as logistic regression and random forests [1]. Additionally, word- and emoji-based sentiment lexica have been found to complement each other nicely in this task [3]. Researchers in [2] also synthesized a framework from various sources to determine a Twitch emote's sentiment. The first distinction that this paper will have compared to others is to experiment with varying deep learning architectures such as LSTMs and GRUS and with multimodal approaches. In addition, another distinction that this paper will employ is to experiment with different embedding techniques. The synthesis of these approaches will yield the building up of the final model to maximize sentiment classification.

3 Methods

The Kobs annotated semantic analysis dataset [3] will be the focus of analysis. This dataset contains 1,922 samples of 6 relevant features and a target label, "sentiment", containing three classes, "negative", "neutral", and "positive". The 6 features are: Date, Channel, Game, Mod, Subscriber, and Message. The "Message" feature contains a string of words which represent a message sent by a given user on Twitch.

The other relevant features within the data that this paper will explore are the "Channel", "Mod", and "Subscriber" columns as it seems likely that these features will influence the sentiment of a message. A key obstacle will be determining how to extract information from the messages in the dataset so as to perform prediction. This requires investigating various preprocessing and embedding techniques.

Prior research will guide the preprocessing approach, but other preprocessing techniques will be used as well. The authors of the paper containing the dataset utilize lowercasing and tokenizing both punctuation and words. The only exception made in [3] is the case of emotes, emojis, and emoticons which are preserved as-is. URLs are all represented as the same token, and words with added letters (often for emphasis) are reduced to being just one letter over their original English spelling in the case of additional added letters. This exact approach will be implemented as well as other preprocessing techniques, such as those described in [2].

For embedding, both techniques employed in [2,3] and the corpora used will be considered for analysis. Both pretrained and untrained models will be utilized. The pretrained model that will be used is GloVe [9] and the untrained model is Word2Vec [10]. For training the Word2Vec model, selecting an appropriate corpus for it to train on is a target of interest as Twitch neologisms and emotes can tend to shift over time, as well as change from channel to channel [2]. For this reason, 2020 and 2022 Twitch data from related streamers to the target dataset will be experimented with to determine the impact on both the embedding vectors and model performance. Sources for the various corpus data can be found in [3,11,12].

Once embedding strategies have been created for the messages, they will be fed through various model architectures. These architectures explored will be include LSTMs and GRUs. For the LSTM and GRU, several hyperparameters in terms of number of units/cells, bidirectionality, dropout, and activation functions will be experimented with. Additionally, the other features present in the data from [3] will be concatenated to the LSTM/GRU's output for the embeddings. This concatenated representation will then be passed forward for sentiment classification, hopefully improving prediction. For clarity, both approaches will use both types of embeddings techniques, as they are described below in Section 4.

4 Experimental Design

4.1 Current Baseline

The current state-of-the-art machine learning approach on the dataset found by [2] has a baseline accuracy of 71.2%. The objective metric for assessing the models described below will be accuracy of predicting the correct classification for the sentiment of a message.

4.2 Subsetting the Data

The suggested stratified train/test split of 80%/20% in [2] was implemented in this paper. The dataset was split in a stratified manner to preserve the prior distribution of 40.1%/38.9%/21.0% (positive, neutral, negative respectively) of the sentiment class. From there, the training data were split into a 87.5/12.5 split to create a validation dataset. This ultimately results in a 70/10/20 split with 1,344 observations in the training set, 193 observations in the validation set, and 385 observations in the test set.

The data splits were each further separated into three distinct frames. The first frame, X_1, contains the messages. The second frame, X_2, contains metadata information such as channel name, if the user is subscribed, and if the user is a moderator. The third frame, Y, contains the sentiment classification. These values correspond to 0 being a negative message, 1 being a neutral message, and 2 being a positive message.

This paper will not use K-fold validation as [2] found that “a 5-fold cross validation evaluation is consistent with the results obtained with a fixed split”. However, unlike in [2], *all* 1,922 messages will be considered. The end goal is to produce a matrix of results with the various embedding techniques as rows, architectures (with optimum hyperparameters) as columns, and the values for accuracy, precision, recall, and F1 as the cells. This will allow researchers to better understand how different embedding methods and techniques work on Twitch data using common neural model architectures.

4.3 Formulating the Embedding Techniques

Three different embedding choices are experimented with using a combination of Word2Vec and GloVe embeddings. The Word2Vec embeddings were generated on Twitch corpora from 2020 and 2022. This choice was deliberate as the data in the dataset comes from 2018 and the authors of this paper did not want the embeddings to possibly come from the streams in the dataset (Scott - explain this to me).

The first corpus, which this paper will refer to as the *short* corpus, was collected by the authors using Twitch’s API from six popular Twitch streamers over the week of April 12, 2022, to April 19, 2022. The chat messages from this corpus underwent preprocessing. First, they were tokenized with the TweetTokenizer in the NLTK Python library [13]. This tokenizer was selected as it would keep certain text-based emojis, such as ":)", and URL links as a single token. Next, the unique ratio of each message was calculated by dividing the number of unique tokens in the message by the total number of tokens in the message. Finally, the corpus's messages were filtered and appended to a list so that each message meets all the following requirements: has at least 3 tokens, has no more than 20 tokens, and (3) has a unique ratio of at least 65%. These filtering decisions were made so as to minimize the number of spammy or repetitious messages passed into the Word2Vec algorithm as not doing so could cause learned embeddings to be suboptimal. This created a list containing 244,380 messages. This message list was sent into the Word2Vec algorithm with the parameter minimum token count set to 6 and an embedding dimension of 200. This resulted in a Word2Vec vocabulary size of 9,524. This 9,524-dimension Word2Vec embedding is the *short vector*.

The second corpus, which this paper will refer to as the *long* corpus, is attributed entirely to [12]. The authors of [12] provide 1,951 CSV files with anonymized 2020 Twitch streamer data. These files were read in and the messages therein were preprocessed in the same way as described for the *short* corpus, resulting in a list containing 5,841,389 messages. This message list is appended to the end of the *short* message list. The concatenated list is then passed into Word2Vec, with the parameter minimum token count set to 15 and an embedding dimension of 200. This resulted in a Word2Vec vocabulary size of 54,451. This 54,451-dimension Word2Vec embedding is the *long vector*.

Continuing on, 200-dimensional GloVe embeddings were also loaded in. To generate the ***short embedding***, all X_1 from the training set are checked first against the *short vector*. If the word exists in the vector, then that word is assigned to the *short vector*'s value. If not, then it is checked against the GloVe embedding vector. If it exists there, then it is assigned to the GloVe vector's value. If it does not exist in either, then it is set to be a 200-dimensional vector of all zeroes. Doing this results in 25.0% of the vocabulary in the X_1 training set being set to 0.

To generate the ***long embedding***, the same process as described above is performed, except with respect to the *long vector*. This technique results in 18.9% of the vocabulary in the X_1 training set being set to 0. Finally, as a baseline for model performance, the ***zero embedding*** was also created. This embedding is all zeroes for all words in the vocabulary of the training set's messages.

Short embedding, ***long embedding***, and ***zero embedding*** are the embedding techniques used in this analysis. This objective here is to see how the different embeddings affect model performance when predicting Y.

4.4 Model Architectures

As a brief overview, LSTMs and GRUs are the architectures of choice. Further details, such as the number of nodes used in each layer and hyperparameters, can be found in Section 5. One of three aforementioned embedding techniques are

Table 1: This table shows the results from the optimal model runs when evaluated against the test set. Our best results are in bold-italics, whereas the current state of the art is just in bold. The main metric of interest is accuracy, as that is the only metric that the current state of the art reports. Other metrics such as precision, recall, and F1 are included here for completeness

		Model	*Dolin et al.*	**LSTM**		**GRU**		**+Metadata**	
Emb. Choice		**Train Emb?**	—	True	False	True	False	True	False
Dolin et al.		Accuracy	**71.2**	—	—	—	—	—	—
Short		Accuracy	—	61.6	57.9	61.3	58.7	62.1	57.1
	Precision	Macro	—	60.3	56.3	59.2	57.8	60.5	55.9
		Weighted	—	61.0	58.5	60.8	59.4	62.3	57.7
	Recall	Macro	—	57.1	54.3	58.2	56.9	60.3	52.1
		Weighted	—	61.6	57.9	61.3	58.7	62.1	57.1
	F1	Macro	—	57.6	54.7	58.5	57.1	60.4	52.2
		Weighted	—	60.5	57.6	60.9	58.8	62.2	55.9
Long		Accuracy	—	61.6	61.8	61.3	58.7	***64.2***	61.8
	Precision	Macro	—	60.3	60.0	59.2	57.8	***62.3***	59.8
		Weighted	—	61.0	62.1	60.8	59.4	***64.9***	62.2
	Recall	Macro	—	57.1	57.8	58.2	56.9	***62.8***	59.1
		Weighted	—	61.6	61.8	61.3	58.7	***64.2***	61.8
	F1	Macro	—	57.6	58.3	58.5	57.1	***62.4***	59.3
		Weighted	—	60.5	61.4	60.9	58.8	***64.4***	61.9
Zero		Accuracy	—	58.7	39.9	59.7	39.9	43.1	43.1
	Precision	Macro	—	56.7	13.3	58.2	13.3	30.1	30.1
		Weighted	—	58.3	15.9	60.0	15.9	35.6	35.6
	Recall	Macro	—	55.9	33.3	58.5	33.3	36.2	36.2
		Weighted	—	58.7	39.9	59.7	39.9	43.1	43.1
	F1	Macro	—	56.2	19.0	58.3	19.0	29.8	29.8
		Weighted	—	58.4	22.8	59.9	22.8	35.3	35.3

passed into either a bidirectional LSTM or GRU. The bidirectional LSTM/GRU is then followed by another bidirectional LSTM/GRU before finally being sent to a dense layer of 3 nodes with a softmax activation function to predict which sentiment class the message belongs to. The difference between each model run is if the neural network is allowed to train the embedding layer. Additionally, another model architecture, +metadata, concatenates X_2 data after X_1 data have been passed through the LSTMs/GRUs. The X_2 data are passed through a dense 3-node layer before being concatenated.

5 Experimental Results

Two-layer bidirectional LSTMs and GRUs were the optimal model architectures. In general, optimal batch sizes and epochs were 16 and 32 respectively, though early stopping often meant that most of the models ran for less than 32 epochs. For consistency and comparison between architectures, a fixed number of nodes were used in the LSTM and GRU layers, which generally worked well. Some hyperparameter configurations completely blocked any learning of the data even on the training data. When the number of nodes was set too low (generally less than 8), the model would fail to improve past predicting the majority class.

5.1 Effect of Model Architecture

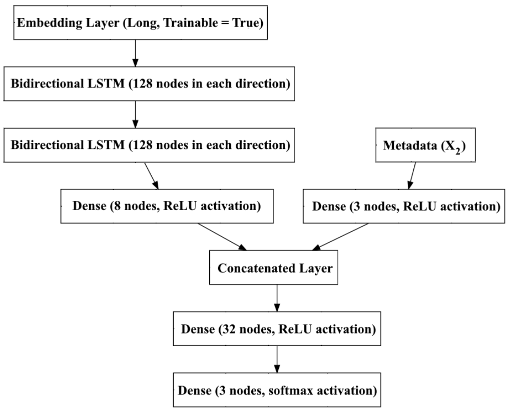
Model architecture did not have much impact on overall accuracy. LSTMs and GRUs performed nearly identically when just looking at the message embeddings (that is, using just X_1). This is to be expected for the most part, as they are in the same family of RNN variants. The biggest impact was from the added metadata of the metadata features (X_2), specifically whether the message came from a moderator or a subscriber. This makes sense, as generally moderators don't post negative comments, and subs don't either to a lesser extent.

5.2 Effect of Embedding Choice

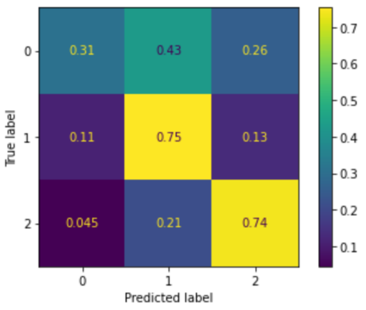
Embeddings were clearly very important. Even so, both the short embedding and long embedding techniques performed incredibly similarly. For example, word embedding vectors for known semantic entries like "lol" between the different embedding techniques generally yielded similar vectors. This is demonstrated in Table 1 where in the test accuracy results between the short and long embeddings were virtually indistinguishable. The biggest change between embedding choices for LSTMs and GRUs was actually based on whether or not we allowed the embedding layer to update the embedding vectors during training, which generally improved model performance. This is likely due to weight updates happening to the 18.9% (long embedding) or 25% (short embedding) words that had zeroes as their embedding.

The baseline of using the zero embeddings while not allowing the embedding layer to be trained and updated behaved as expected. The 39.9% accuracy in Table 1 for both the LSTM and GRU is from the model solely predicting the majority class. However, interestingly, when we let the models train the embeddings, the performance almost matches that of using either the short or long embeddings. Learning embeddings in this short of time was quite impressive, although upon further examination, the embedding weights were relatively smaller than their short embedding or long embedding counterparts. While the models using this embedding technique did not perform the best, to be able to reach greater than 58% accuracy was not expected. However, this ***zero embedding*** technique did fail in the +metadata model, most likely due to some implementation or Keras library issue. In all, we see that starting from a predefined model trained from a Twitch corpus provides an advantage over just starting and training the embeddings from 0, all other variables held equal. Additionally, allowing the models to train on the predefined embeddings still improves performance across the board on the test set.

Fig. 1: The architecture and confusion matrix of the best model


Embedding Layer (Long, Trainable = True)
Bidirectional LSTM (128 nodes in each direction)
Bidirectional LSTM (128 nodes in each direction)
Metadata (X_2)
Dense (8 nodes, ReLU activation)
Dense (3 nodes, ReLU activation)
Concatenated Layer
Dense (32 nodes, ReLU activation)
Dense (3 nodes, softmax activation)

(a) The architecture for the best performing model



True label \ Predicted label	0	1	2
0	0.31	0.43	0.26
1	0.11	0.75	0.13
2	0.045	0.21	0.74

(b) The confusion matrix of the best model. For example, 75% of neutral messages (1) in the test set were classified correctly

5.3 Best Performing Model

Finally, the best model's results are shown in bold-italics in Table 1. The architecture and confusion matrix for this model can be found in Figure 1. The best model was the +metadata model that allowed the ***long embedding*** scheme to be updated and trainable.

For the most part, it does a decent job in categorizing positive and neutral messages. Most misclassified positive messages were misclassified as being neutral messages. However, the misclassified neutral messages were equally likely to be classified as negative or positive. This is likely due to the ambivalence present in messages labelled as neutral where the embeddings for the words in the neutral messages aren't providing a strong enough signal for the model to correctly classify them.

The model performed the worst on the negative sentiment messages. In fact, it predicted a plurality of them as neutral! The problem is two-fold. First, negative messages were the minority class across all the datasets. This lends the model to have the proclivity of not predicting them correctly. Second, some of the messages within the original dataset that are labelled as negative *aren't* negative. For example, while "cata dumbass" is a negative sentiment message, it is enigmatic as to why messages such as "go out" and "???" are labelled as negative when they are more than likely neutral. Even with some data mislabelling, these examples are much less than the 43% implicated in the confusion matrix.

5.4 Future Direction

Going forward, there are several avenues for future research that could be explored. One potentially fruitful approach could include investigating modalities for the audiovisual content of the stream. This would have several obstacles such as the amount of data in video streams, or being able to retroactively collect the data itself. It would add much needed context to ambiguous emotes or phrases, which is lacking in unimodal, message only, models. However, both of these would not necessarily be guaranteed to impact the output, since the Mechanical Turk workers that evaluated the semantics did not have access to that data when they performed their evaluation [3]. For future work, obtaining a more recent and more carefully annotated semantic analysis dataset and audiovisual data from the livestream video and its transcript could be beneficial to prediction accuracy.

Another useful approach could be to gather and include other potentially useful information such as if streamers belong to similar communities (such as if they collaborate together) or if certain games belong to the same genre. Communities might say a phrase that appears to be negative but the actual intent behind them is positive. Similarly with groups of games in similar genres, similar game terminology and phraseology might be indicative of the sentiment behind a message. However, like with including any amount of new variables, it is important to check to see if the effect actually matters in predicting the outcome variable as adding superfluous variables and metadata will only add noise to the model and slow it down. Even so, the metadata variables that we did include did garner additional improvements in model performance.

Lastly, other embedding techniques on the message data could be the most important of all improvements. As past research has shown, embedding techniques are crucial to model success. Researchers in [2] created a network called LOOVE, short for Learning Out Of Vocabulary Emotes, to generate sentiment across many Twitch emotes using their own data collected using Twitch's API. A similar approach could be created or constructed. Additionally, using a transformer based approach could allow for greater semantic understanding of the words in context, compared to Word2Vec or GloVe embeddings. While the word embeddings for canonical phrases like "lol" did show good relation to other similar words, transformers have the advantage of self-attention.

6 Conclusions and Limitations

Semantic analysis and classification of Twitch messages has many challenges. Due to the subjectivity of semantics, rapidly changing neologisms, and low amount of labelled data, any machine learning architecture will have to be able pick up on a lot of nuance from limited amounts of data. This can be problematic for neural network approaches, which generally struggle with small amounts of data, due to the nature of having to learn many parameter weights while trying to generalize to unseen data. Although the results did not outperform traditional machine learning techniques, many lessons were learned, and a neural network approach could still potentially be viable given more data, better word embeddings, better architecture design, or some combination of these factors. A focus on embedding techniques could be a key aspect for research going forward, as that is one of the more complex aspects to the Twitch dataset. Added metadata and variables can improve accuracy to a certain extent, as there is most likely useful signal coming from the metadata from which neural models can interpret to improve predictions.

The analysis done on Twitch semantics was intended to be done for research purposes only. All user data was completely anonymized prior to analysis [3]. Any future research or commercial application should take into account the following biases. First, data used in this analysis comes exclusively from English streams, indicating the results will have a Western bias. Additionally, Twitch has a mostly male user base, and this additional bias should be considered especially for comparison between other mediums.

References

1. Chouhan, A., Halgekar, A., Rao, A., Khankhoje, D., Narvekar, M.: Sentiment analysis of twitch. tv livestream messages using machine learning methods. In: 2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT), IEEE (2021) 1–5
2. Dolin, P., d'Hauthuille, L., Vattani, A.: Feelsgoodman: Inferring semantics of twitch neologisms. arXiv preprint arXiv:2108.08411 (2021)
3. Kobs, K., Zehe, A., Bernstetter, A., Chibane, J., Pfister, J., Tritscher, J., Hotho, A.: Emote-controlled: obtaining implicit viewer feedback through emote-based sentiment analysis on comments of popular twitch. tv channels. ACM transactions on social computing **3**(2) (2020) 1–34
4. Barbieri, F., Ballesteros, M., Ronzano, F., Saggion, H.: Multimodal emoji prediction. arXiv preprint arXiv:1803.02392 (2018)
5. Venkit, P., Karishma, Z., Hsu, C.Y., Katiki, R., Huang, K., Wilson, S., Dudas, P.: A 'sourceful' twist: Emoji prediction based on sentiment, hashtags and application source. arXiv preprint arXiv:2103.07833 (2021)
6. Felbo, B., Mislove, A., Søgaard, A., Rahwan, I., Lehmann, S.: Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. arXiv preprint arXiv:1708.00524 (2017)
7. Olejniczak, J.: A linguistic study of language variety used on twitch. Tv: Desriptive And Corpus-based Approaches (2012) 21–23
8. Kim, J., Wohn, D.Y., Cha, M.: Understanding and identifying the use of emotes in toxic chat on twitch. Online Social Networks and Media **27** (2022) 100180
9. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). (2014) 1532–1543
10. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
11. Hutto, C., Gilbert, E.: Vader: A parsimonious rule-based model for sentiment analysis of social media text. In: Proceedings of the international AAAI conference on web and social media. Volume 8. (2014) 216–225
12. Ringer, C., Nicolaou, M., Walker, J.: Twitchchat: A dataset for exploring livestream chat. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. Volume 16. (2020) 259–265
13. Bird, S., Klein, E., Loper, E.: Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc." (2009)

Appendix

The models described in this paper were run in Python 3.7.13 in Google Colab, using at most 10 GB of RAM. The code that the authors used can be found here: https://github.com/xjseabrum/nndl_final_project/