Twitter: A Commentary

Ishan Shah Michael Chen Rishi Ponnekanti Udai Jain

The University of Texas at Austin

Problem

Twitter has grown rapidly in the last decade as a platform for sharing short-form written content. It's used by individuals to share thoughts and jokes, academics to discuss research, and even governments as a primary form of communication. This project seeks to predict response sentiment on a tweet's replies using the tweet's content.

This problem is relevant today since many creators want to gauge how their audience will respond to their content before publishing it. This can also be beneficial to organizations to help rephrase tweets that may be interpreted negatively to protect reputation. Finally, while this project will be focused on Twitter, the techniques can easily be applied to other major platforms like YouTube and Facebook to help content creators gather insights.

Data Source

Building a model to accurately predict reponse sentiment requires a large amount of data. Manually collecting enough tweets and replies with the Twitter API is not feasible due to the rate limit, so we used an existing dataset [1] to train our model. The data was provided as a CSV file of 34,521 tweet IDs, each of which had a list of reply sentiments (positive, neutral, negative) totalling in 1,519,504 unique replies.

Using this, we collected the tweet text for each tweet using the `GET statuses/show/:id` route of the Twitter API. This took around 11 hours since we had to comply with the rate limit of 900 requests / 15 minutes. We also computed sentiment scores by converting the reply sentiments to numerical values (positive=1, neutral=0, negative=-1) and taking the average. Lastly, we added a column with the number of replies to each tweet.

Pre-Processing

Once we had each tweet's text, we performed the following pre-processing steps:

- **Remove whitespace:** Remove any excess whitespace using regular expressions.
- **Normalize case:** Convert all letters to lowercase.
- **Tokenize string:** Split the text into tokens by whitespace.
- **Remove stopwords**: Use NLTK's stopwords dataset [2] to remove common words like `the`, `and`, `to`, etc.
- **Lemmatize words**: Use NLTK's lemmatizer to convert words to their base form.
- **Remove punctuation:** Remove any words containing punctuation like `@` or `#`, which show up frequently in tweets.

Finally, we're left with the following features:

Cleaned Tweet	Reply Sentiment	Reply Count
fox corporation owner fox news trying bully ro...	-0.520000	150
folk hear cornovirus deal news heck doctor rec...	-0.200000	45
news finally excited journey potential finding...	0.500000	6
good news person investigation novel coronavir...	-0.250000	12
two avid golfer promised whoever died first wo...	0.166667	36

Table 1. Dataset containing the cleaned words in each tweet, the sentiment of the tweet's replies, and the number of replies.

Vectorization

Performing text vectorization is common in natural language processing as it allows for text to be represented as vectors of numbers, letting us perform linear algebra operations on them. We used Scikit-learn's `TfidfVectorizer`, which uses the term frequency–inverse document frequency (TF-IDF) model to determine which words are most relevant to each document.

Let t be some term in a document d. The TF-IDF score is computed with:

$$\text{TFIDF(t, d)} = \frac{\text{Frequency of } t \text{ in } d}{\text{Total words in } d} \cdot \log\left(\frac{\text{Total documents}}{\text{Documents containing } t}\right)$$

This captures the importance of each term in a document (TF) relative to how frequently it occurs in other documents (IDF). This is critical because it helps us find words that are unique to documents, rather than words that are common to all documents like `the`, `and`, etc. We can now train machine learning models on the vector representations of these tweets.

Machine Learning

We experimented with a few different models (linear regression, decision trees, support vector machines) before settling on `XGBoost`, a gradient boosted decision tree model, as it yielded the lowest error. Specifically, we used `XGBRegressor` as we were trying to predict the sentiment score of a tweet's replies which can have a continuous value. We split the data into training and testing sets, where each training example consisted of a tweet and the sentiment score we computed earlier. We used `RandomizedSearchCV` to tune the model's hyperparameters as it randomly selects parameters, performs cross-validation, and returns the parameters that yielded the lowest error.

We can visualize the model's performance by treating the model's results as a classification problem. If we take the sign of the average of the sentiment scores, we can reduce the problem to binary classification. We can then plot an ROC curve to see how often the model correctly predicts a positive or negative sentiment:

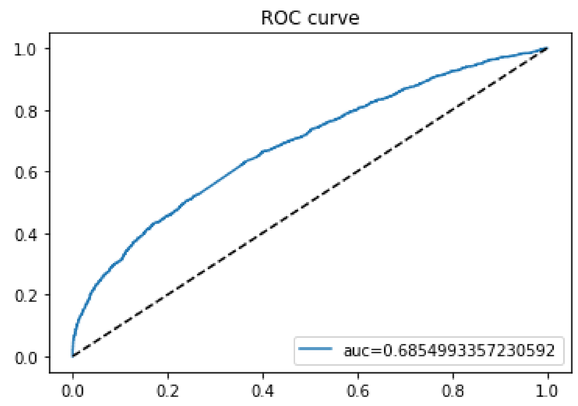

Figure 1. ROC curve for the model on the training set.

This, however, is not a perfect metric since many tweets have a neutral sentiment that hovers around 0, so taking the sign of these tweets is not a good indicator of its sentiment.

Results

We tested our model on a subset of the data and found that the mean squared error on the best model was **0.1971**. Here are some takeaways this led us to:

- It is hard to judge the model's quality due to the nature of the data.
- While the model generally predicts the sentiment of the replies correctly, it is often swayed by certain words in the input with abnormally high prior probabilities.
- Sarcasm and humor are interpreted subjectively but machine learning models make objective predictions.

Product Development

After training the model, we decided to build a simple webapp that would let users predict response sentiment on tweets of their own. We created a lightweight backend with FastAPI and a frontend with React.js. Upon receiving a tweet, the backend cleans, vectorizes, and runs a saved version of the model on the tweet. The frontend then displays the predicted response sentiment on a sliding scale. Below is a screenshot of the frontend tested with one of Elon Musk's tweets:

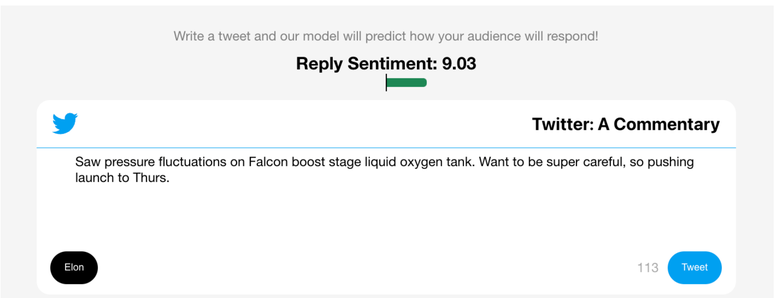

Figure 2. Sample tweet and predicted sentiment from https://twittercommentary.netlify.app/

Future Research

In the future, we would like to build a more robust model. We noticed that the model could be easily skewed by the presence of certain words in the tweet with high prior probabilities like `love`. This could be combatted by training on a wider dataset or by using an n-gram model, which splits the data into n word chunks to partially preserve the relationship between words that are next to each other. We would also like to make the model more easily interpretable by users since it is currently unclear which words affect the predicted sentiment the most.

References

[1] S. Tayebi Arasteh, M. Monajem, V. Christlein, P. Heinrich, A. Nicolaou, H. Naderi Boldaji, M. Lotfinia, and S. Evert, "How will your tweet be received? predicting the sentiment polarity of tweet replies," in *Proceedings of the 2021 IEEE 15th International Conference on Semantic Computing (ICSC)*, (Laguna Hills, CA, USA), pp. 370–373, 01 2021.

[2] E. L. Steven Bird, "Natural language toolkit."

[3] Y. Matalon, O. Magdaci, A. Almozlino, and D. Yamin, "Using sentiment analysis to predict opinion inversion in tweets of political communication," *Scientific Reports*, vol. 11, no. 1, 2021.

[4] V. A. Kharde and S. S. Sonawane, "Sentiment analysis of twitter data : A survey of techniques," *CoRR*, vol. abs/1601.06971, 2016.