Data Analysis and Machine Learning with Python
Final Report

# A Merged System of Bot Detection and Sentiments Analysis for Social Media Comments

Group C

{PHOU, Valentin. Dept. of Computer Science (NTUST);
GOH, Grace. Dept. of Electrical Engineering and of Computer Science;
CHANG, Hao-Yun. Dept. of Atmospheric Sciences and Dept. of Law;
SHIAU, Ren-Wei. Dept. of Atmospheric Sciences;
Roman Vertichikh, International Advanced Technology Program}

Abstract

*In this project, we developed a python-based system that merged bot comment detection and sentiment analysis. In particular, we used multiple unsupervised approaches to separate real comments and bot comments, before feeding them into a sentiment analyzing RNN (recursive neural network) model that labeled the sentiments of all comments as "positive", "negative", or "neutral". This generalized model could possibly be used on analyzing the impact on "public"[1] attitude brought by bot comments.*

[1] A quote is used here since we found out that there does exist bot comments on public forums.

## Table of contents

## I. INTRODUCTION

Sentiment analysis has been a widely applied technique to understand the opinion of the public toward specific issues. However, with the number of bot-generated comments growing on social media, we have to find a way to distinguish them in order to ensure that the result of sentiment analysis is genuine. In this project, we aimed to build a system that could *detect* and *analyze* the sentiments of bot comments (comments generated by AI instead of humans) on real social medias (in our project, comments from Reddit were used for example) centered at a specific topic (in our project, the launching of GTA6). The workflow of our system could be visualized as the following figure:

We expected this system to have a wide variety of usages on commercial, political, public opinion analyzing fields, etc. Although we used Reddit comments and GTA6 for functions-demonstration in this report, the model was actually *generalized* and could be applied to comments from *any* forum about *any* topic.

## II. Related Work

With the rapid development of social media, bot accounts have exerted a significant influence on online community ecosystems, driving advancements in bot detection technologies. Among these, methods that combine machine learning with sequential models have gradually become a focal point of research. In a project published on GitHub, Reddit Bot Detector, Tourond (2021) explored the problem of bot account identification on the Reddit platform using supervised learning methods. This project employed feature engineering to collect multidimensional data (such as account activity patterns and linguistic features) and trained traditional classification models for prediction. Although the study did not employ deep learning techniques, it nonetheless provided empirical foundations and feature design references for subsequent applications of sequence modeling in bot detection on social platforms [1].

On the other hand, the development of Long Short-Term Memory (LSTM) models has marked a significant breakthrough in the field of sequential data processing. Hochreiter and Schmidhuber (1997) first proposed the LSTM model to address the vanishing gradient problem encountered by traditional Recurrent Neural Networks (RNNs) when learning long sequences [2]. By introducing mechanisms such as the forget gate, input gate, and output gate, LSTM networks are able to effectively retain crucial information while suppressing irrelevant signals, making them particularly suitable for tasks involving long-term dependencies, such as language modeling and speech recognition. Subsequently, Gers, Schmidhuber, and Cummins (2000) proposed an enhanced version of the LSTM architecture, emphasizing the critical role of the forget gate. This modification enables the model to automatically reset parts of its internal state when processing continuous sequences, thereby reducing the impact of cumulative errors and enhancing the stability and generalizability of the model [3].

## III. Data Description

We used a self-established web scraper (please refer to the contents below) to search reddit discussions titled about “GTA6”. The scraper automatically generated a dataset that treated one comment as an independent data, alongside with their username and main thread (i.e. title of the discussion). The whole dataset was used for bot detection since this part involved unsupervised learning only and didn’t require labels for sentiments; on the other hand, we labeled the sentiments for 421 comments manually. That is, we read through the comments and decided what type of sentiment it contained. We trained the sentiment analyzing model with these parts of the dataset, and let the model label all of the dataset after the model is successfully trained. The datasets, unlabeled, labeled manually, labeled by the model, were attached to the file submitted.

## IV. Methodology

### i/ Web Scrapping

In this project, Reddit web scraping was performed to collect user comments related to the "GTA6 trailer" topic. A list of Reddit post URLs in JSON format was used as input. For each URL, a web request was sent using the requests library while including appropriate headers to simulate a browser and avoid being blocked. The JSON response contains structured data about the original post and its comments. The program extracted the post title and looped through all top-level comments, collecting the username, timestamp, and content of each valid comment (excluding deleted or system comments). The collected data was then organized into a table using the pandas library and appended to an existing Excel file using openpyxl, ensuring no data was overwritten. This approach allowed efficient extraction and storage of Reddit discussion data for further analysis.

### ii/ Bots Detection

Following the initial web scraping phase, which yielded a dataset of Reddit comments centered around the official GTA6 trailer, our objective was to identify potential bot activity within the dataset. We took inspiration from the bot detection structure of a previous project published on GitHub [1], although their approach was supervised. Since no labels were available to indicate which users or comments originated from bots, supervised learning was not feasible, with our dataset.

Moreover, even if labeled data were available, supervised learning often suffers from a lack of generalizability: models trained on a specific dataset may not adapt well to new or evolving patterns of bot behavior, particularly in dynamic environments like social media. Bot strategies change frequently, and a supervised model might become quickly outdated without continuous retraining on new labeled data.

Given these constraints, we decided to adopt an unsupervised learning approach, focusing on anomaly detection techniques that do not require labels but instead rely on identifying patterns that deviate significantly from the norm.

## ii.1/ Initial Approach: User-Level Detection

### ii1.1/ Feature Extraction

Our initial detection strategy focused on identifying bot-like behavior at the **user level**. The intuition behind this approach was that bots often exhibit repetitive or unnatural posting patterns across multiple comments. To capture such abnormal activity, we engineered a diverse and multi-faceted set of features by grouping the dataset by user.

We designed the following feature categories:

- Text-based features: **average comment length** and its **standard deviation**, **keyword repetition ratio** (reflecting how repetitive the user's messages are), **ratio of unique words** (measuring lexical diversity), **number of comments containing URLs** (since bots often include external links).

- Temporal features: **comment rate** (average number of posts per hour), **variance in posting intervals** and **max burst activity** (i.e., max number of comments posted in a 1-hour window), **most active posting hour/day** and their **respective entropy**, **hourly frequency dominance** (measuring if one hour concentrates a large proportion of activity).

- Thread interaction features: **number of distinct threads** a user participates in, **thread entropy** (to assess dispersion across discussions), **reply frequency**: (proportion of comments in the most active thread).

- Similarity measures: **inter-comment similarity** (computed via TF-IDF cosine similarity between the user's comments), **context similarity** (which compares comment content to its associated thread title).

This extensive set of features was designed to capture behavioral, temporal, emotional, and contextual cues, all of which could signal unnatural or automated activity.

### ii.1.2/ Modeling

Given the absence of ground truth labels for bot detection, we opted for unsupervised anomaly detection models. We used ChatGPT to guide the selection of effective models known to perform well in anomaly detection, particularly for text and behavioral data. Based on this, we selected three complementary models:

- Isolation Forest: an ensemble method that isolates outliers by randomly selecting features and splitting values. It performs well with high-dimensional and sparse data.

- One-Class SVM: a boundary-based method that identifies deviations from a learned normal region in feature space.

- DBSCAN (Density-Based Spatial Clustering of Applications with Noise): a density-based clustering algorithm that flags points in low-density regions as noise. The eps parameter (neighborhood radius) was selected automatically using the k-distance graph and KneeLocator.

Each model independently flagged users as anomalies. To consolidate predictions, we used a majority voting rule: a user was classified as a bot (is_bot_u = 1) if flagged by at least two of the three models.

## ii.2/ Expanding the Scope: Comment-Level Detection

While analyzing the user-level bot predictions, we observed some limitations. In particular, many users posted only one or two comments, which made it difficult to confidently assess their behavior based on aggregate statistics. Moreover, some users classified as non-bots occasionally posted messages that visibly resembled automated content — overly enthusiastic, repetitive, or spam-like.

To address this, we decided to complement our approach with a comment-level anomaly detection pipeline. The goal was to identify individual suspicious messages, regardless of the overall behavior of the user behind them.

### ii.2.1/ Feature Extraction

For each comment, we computed a set of lightweight yet informative features. These included:

- Textual characteristics: **length** (character count of the comment), **word_count** (number of words), **unique_word_ratio** (lexical diversity within the comment).

- Punctuation & special symbols:
  **has_url** (binary flag indicating presence of a URL for spam or promotions), **has_question** (contains a question mark), **has_exclamation** (contains an exclamation mark), **punctuation_count** (total punctuation marks (.,!?) used), **uppercase_ratio** (proportion of uppercase letters, excessive use may reflect shouting or promotional tone).

- Contextual features: **hour** (used to find unusual activity patterns), **thread_similarity** (semantic similarity between the comment and the title of the thread, low similarity may signal irrelevant or automated messages).

This design aimed to capture abnormal or non-human writing style, content structure, and posting context, with minimal computational overhead.

#### ii.2.2/ Modeling and Classification

Using these features, we applied the same three unsupervised models as in the user-level approach: Isolation Forest, One-Class SVM and DBSCAN.

Each model independently predicted whether a comment was an outlier. Final classification was based on majority voting: a comment was labeled as bot-generated (is_bot_c = 1) if it was flagged by at least two out of the three models.

## ii.3/ Final Decision Rule and Fusion

To unify both detection perspectives, we considered that a comment is ultimately classified as bot-generated (is_bot = 1) if either the user-level model (is_bot_u) or the comment-level model (is_bot_c) flagged it as suspicious.

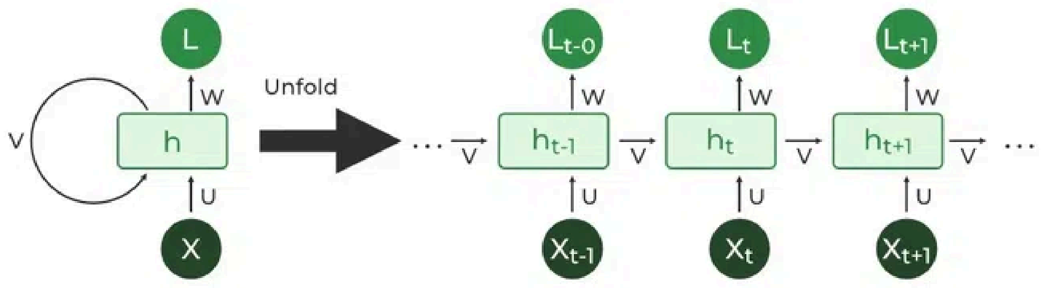
This fusion prioritizes high recall over precision, reflecting our objective to avoid missing any potentially harmful bot activity, even if it means tolerating a few false positives. For instance, a legitimate user who typically posts normal content may occasionally publish a promotional or suspicious comment, and our fusion rule allows us to still capture that single anomaly.

## iii/ Sentiment Analysis

## iii.1/ Framework of The Model: Recursive Neural Network (RNN)

As stated previously, the objective of this part was to build a model that distinguished the sentiment of a comment. In other words, we had to “teach” the model how to read the comments and make its decision about the sentiment of the comments. Upon building the model, traditional machine learning algorithms such as decision tree, random forest, K-nearest neighbor, etc. However, to use these algorithms, we must create features for the comments first (for example, the total length of the comment would be a feasible feature in that case), but what features should we use exactly? Moreover, would the choice of features become a source of bias itself? To avoid these concerns, we turned to deep learning algorithms, in specific RNN-based NLP (natural language processing), instead.

RNN is a deep-learning algorithm widely used on NLP tasks. Long term short, this algorithm “reads” a text sequentially, trains a model with the part it has read already, then keeps “reading”, training a new model with the old one merged in. Hence, the newer model will always contain information from the older ones, enabling the final model to capture the sequential structure in natural language text. The figure below is a schematic picture of how RNN models work:



(https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/)

In particular, we used the Long-Short Term Memory algorithm (Hochreiter et. al, 1997.), which is a type of RNN architecture, to build our model. The key component of this algorithm is the inclusion of forget gate, input gate, and output gate, which decides whether to retain, discard, or use a specific piece of memory (information) as output, which enables the model to learn when to use and when to forget a piece of information (Gers et. al, 2000).

### iii.2/ Model Improvement

We used a 0.8/0.2 train/test split to test the accuracy of the model. In order to improve the performance, we established a semi-automatic hyper parameter tuning program that saved the best model to date throughout the whole testing and used it to predict the unlabeled dataset directly. This avoided the impact of randomness of deep-learning models. That is to say, we wouldn't have to worry about the performance of the model being unproducible, since the particular model with *that* performance is stored in the computer's memory and used directly to predict. The tuning program we used went through three hyper parameters (unit, dropout, recurrence dropout) with 5, 4, and 3 options respectively, thus 60 combinations in total.

In addition, since we had a relatively small dataset, we introduced GloVe (Global Vectors for Word Representation) pre-trained embeddings to help the model learn more about the relationship between different wordings of English.

## V. Results and Analysis

### i/ WEB SCRAPPING

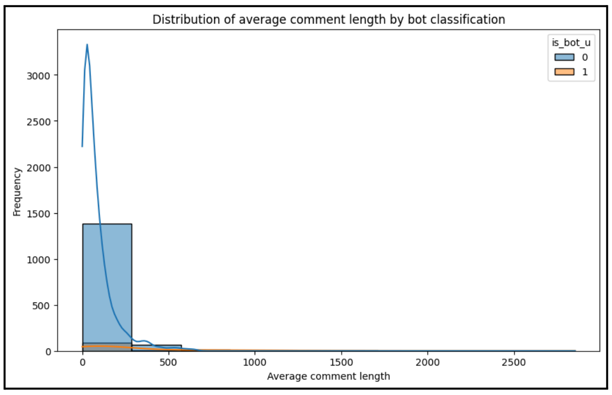
...

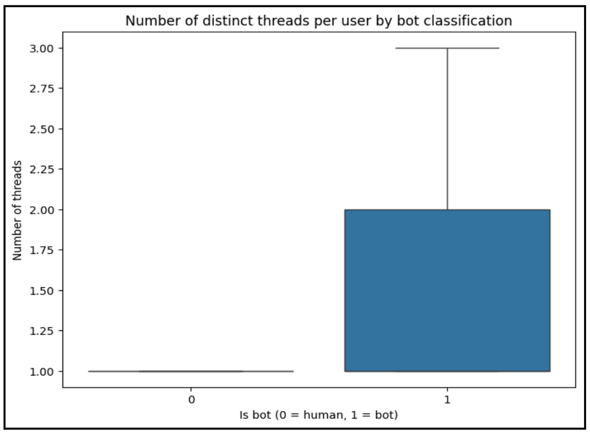
## ii/ BOTS DETECTION

### ii.1/ User-Level detection results

After applying our anomaly detection pipeline at the user level, 120 users were flagged as bots out of 1,575 users, representing approximately 7.6% of the population.

To better understand the behavioral characteristics that led to these classifications, we analyzed key features. One of the most discriminative was average comment length. Bots tend to post much longer comments on average (332.98 characters) compared to humans (83.34 characters). This large gap suggests that many bots produce verbose or promotional messages, often with structured formatting or repeated blocks of text.





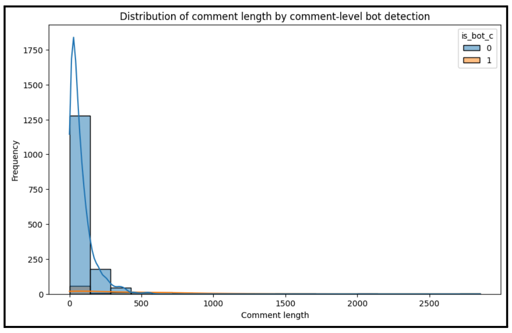
Another distinguishing factor was the number of distinct threads in which users participated. Bot users tended to post in slightly more threads on average, hinting at efforts to spread content across multiple discussions, though the difference remained modest.

Finally, most human users posted only one comment, whereas a few bots posted multiple times.

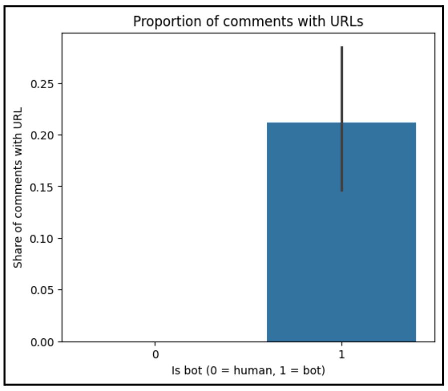
### ii.2/ Comment-level detection results

The comment-level analysis identified 123 comments as bots out of 1,631 total comments (approx. 7.5%). This fine-grained approach captured individual messages that deviated from typical user behavior, even for users not classified as bots globally.

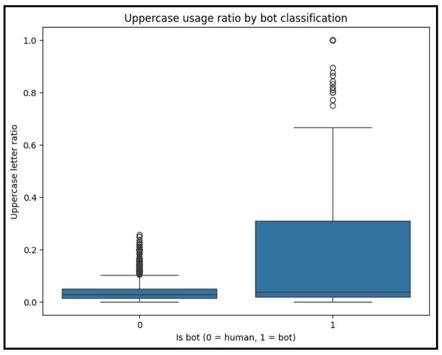
Comment-level bots stood out in several ways. Their average comment length was five times higher than human-written comments (397 vs 78 characters). These messages were often more elaborate, repetitive, or templated.



Bots also used more punctuation, especially symbols like “!”, “?”, and excessive periods, which may be associated with attempts to sound emphatic or trigger attention.



Additionally, 21.1% of bot comments included URLs, while none of the human comments did. This further supports the hypothesis that bot content is often promotional or spam-related.

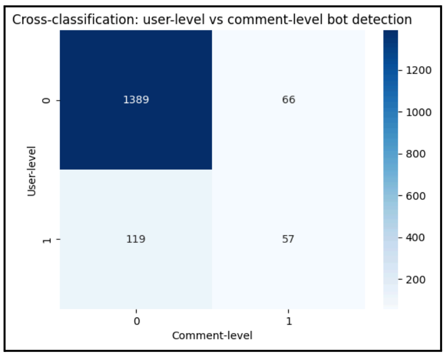


We also found that bot comments used a higher uppercase ratio, a stylistic marker often linked to shouting or emphasis, although this is only a secondary indicator.

## ii.3/ Fusion and final bot classification

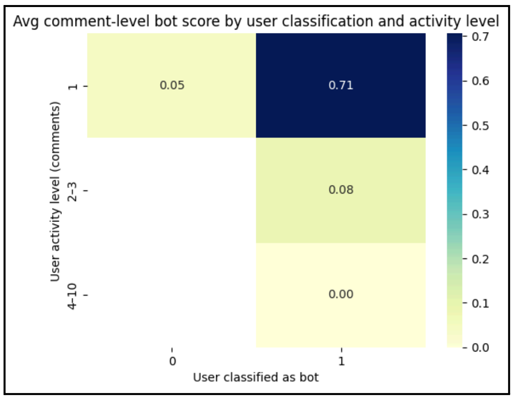
Combining user-level and comment-level outputs via a logical OR rule, we flagged a total of 242 comments (14.8% of the dataset) as bot-generated. Looking at each strategy individually, the user-level model flagged 180, the comment-level model 123, and the fusion resulted in 242 detections. This illustrates the added value of combining both signals.

When comparing the detection overlap between methods, we observed that some users flagged as bots only had normal-looking comments, and vice versa, highlighting the complementary nature of both strategies.



To further understand consistency, we looked at the average proportion of bot-like comments per user. As expected, users classified as bots had a near-maximum bot-score per comment, whereas human users had nearly zero.

Finally, we explored how the average bot-score per comment varied across users depending on both their classification and their activity level. Interestingly, bot users with only one comment had significantly higher bot scores, suggesting that even sparse activity can be revealing.



### ii.4/ Validation Strategy Using Artificially Generated Bot Comments

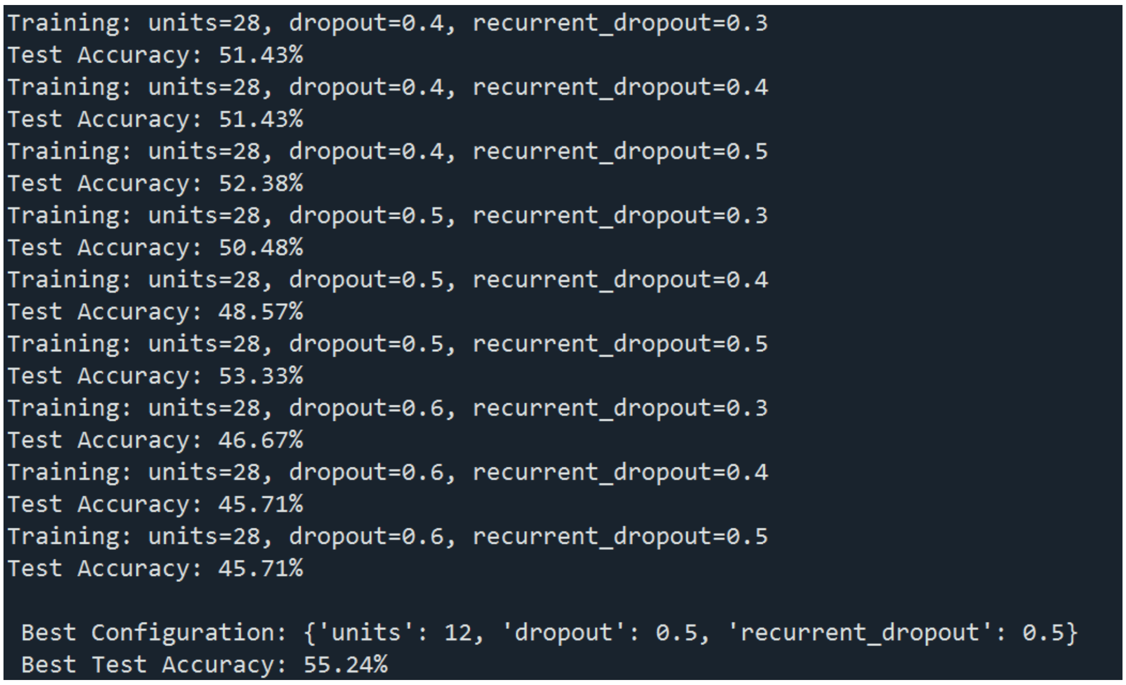
A key challenge in unsupervised learning is the lack of ground truth labels, making it difficult to verify the accuracy of detection. To address this, we manually injected five AI-generated comments crafted to mimic typical bot behavior (e.g., excessive enthusiasm, promotional content, unnatural phrasing), each under a new user ("FakeBot_1" to "FakeBot_5").

After re-running our full pipeline : feature extraction, anomaly detection models, and final fusion, all five synthetic comments were successfully flagged as bots (is_bot = 1).

This qualitative validation confirms that our system detects realistic bot patterns, not just random outliers. While we cannot compute standard metrics like precision or recall, this test strengthens confidence in the model's practical relevance and robustness, even if some false positives or negatives may remain.

## iii/ Sentiment Analysis

The screenshot below was part of the output of the tuning program:


```
Training: units=28, dropout=0.4, recurrent_dropout=0.3
Test Accuracy: 51.43%
Training: units=28, dropout=0.4, recurrent_dropout=0.4
Test Accuracy: 51.43%
Training: units=28, dropout=0.4, recurrent_dropout=0.5
Test Accuracy: 52.38%
Training: units=28, dropout=0.5, recurrent_dropout=0.3
Test Accuracy: 50.48%
Training: units=28, dropout=0.5, recurrent_dropout=0.4
Test Accuracy: 48.57%
Training: units=28, dropout=0.5, recurrent_dropout=0.5
Test Accuracy: 53.33%
Training: units=28, dropout=0.6, recurrent_dropout=0.3
Test Accuracy: 46.67%
Training: units=28, dropout=0.6, recurrent_dropout=0.4
Test Accuracy: 45.71%
Training: units=28, dropout=0.6, recurrent_dropout=0.5
Test Accuracy: 45.71%

 Best Configuration: {'units': 12, 'dropout': 0.5, 'recurrent_dropout': 0.5}
 Best Test Accuracy: 55.24%
```

As we could see, the best performing model after tuning had a test accuracy of 55.24%, considering that our data is divided into three labels, we could say that the model had indeed learned something (the accuracy was obviously higher than guessing the values randomly). However, it could only get slightly more than half of the sentiments right, it was apparent that there was still room for improvement, especially for the purpose of our system.

In particular, when we looked at the training log (pasted below), we could tell that the model suffered from overfitting, which is a common symptom caused by a small, relatively noisy dataset. Thus, we could probably try to simplify the model's structure or just expand the training dataset with an aim for a better performance.

```
Epoch 25/30
26/26 [==============================] - 4s 155ms/step - loss: 0.0720 -
accuracy: 0.9902 - val_loss: 1.6443 - val_accuracy: 0.5676
Epoch 26/30
26/26 [==============================] - 4s 155ms/step - loss: 0.0714 -
accuracy: 0.9902 - val_loss: 1.7056 - val_accuracy: 0.5766
Epoch 27/30
26/26 [==============================] - 4s 154ms/step - loss: 0.0617 -
accuracy: 1.0000 - val_loss: 1.7813 - val_accuracy: 0.5856
Epoch 28/30
26/26 [==============================] - 4s 155ms/step - loss: 0.0730 -
accuracy: 0.9853 - val_loss: 1.7942 - val_accuracy: 0.5586
Epoch 29/30
26/26 [==============================] - 4s 154ms/step - loss: 0.0492 -
accuracy: 1.0000 - val_loss: 1.8091 - val_accuracy: 0.5766
Epoch 30/30
26/26 [==============================] - 4s 156ms/step - loss: 0.0444 -
accuracy: 1.0000 - val_loss: 1.9209 - val_accuracy: 0.5586
```

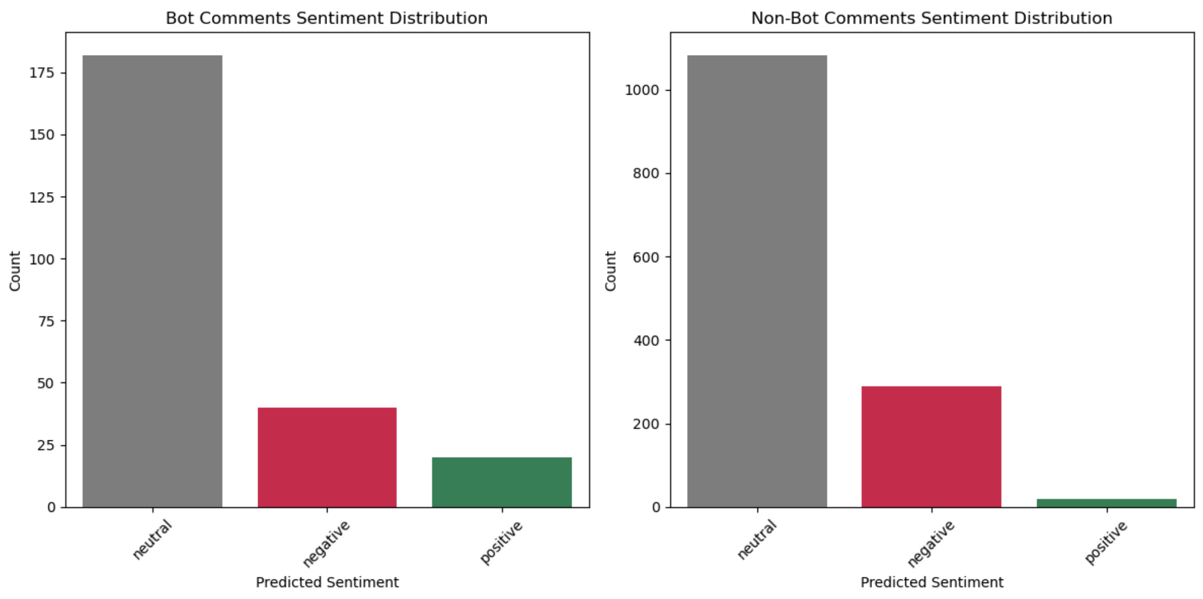
```
4/4 [==============================] - 0s 30ms/step - loss: 2.0959 - accuracy: 0.5333
Test Accuracy: 53.33%
```

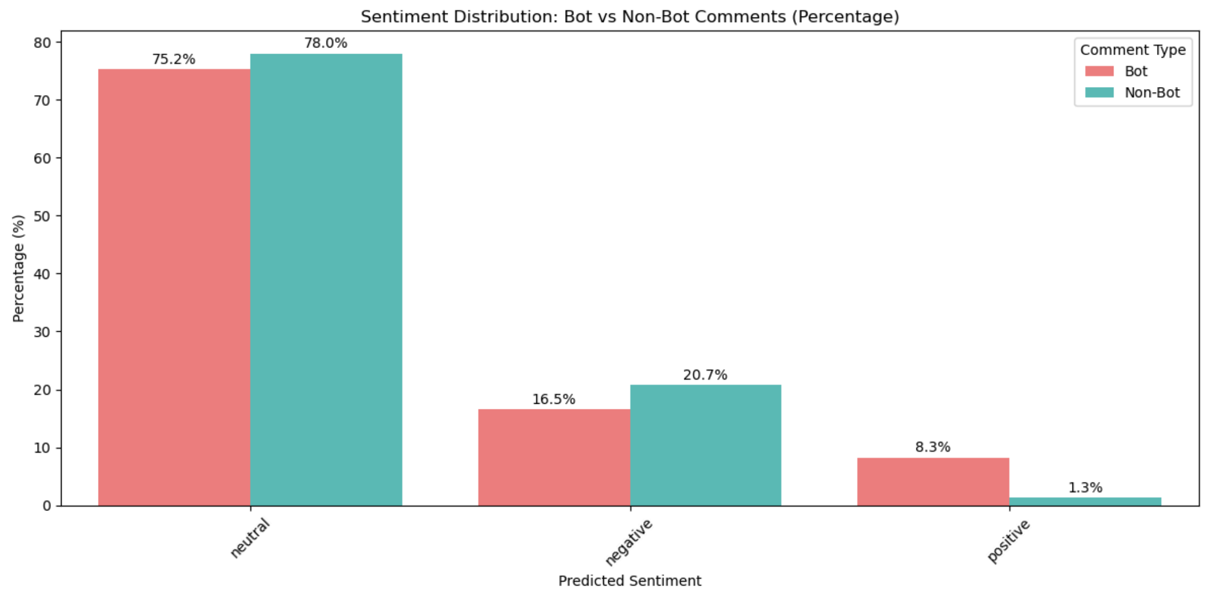
## VI. Discussion

After performing bot detection and RNN analysis, we obtained the results, as shown in the attached figures. There are two figures in total: one compares the number of comments by sentiment for both bots and non-bots, while the other shows the same comparison but in terms of proportions.

There is no significant difference between bots and non-bots in the Neutral and Negative categories, with differences of 2.8% and 4.2%, respectively. However, a notable gap of approximately 7%  and relatively low emerges in the Positive category. This gap warrants further investigation: why is the proportion of Positive sentiment significantly higher among bots?

Our hypothesis is that the official GTA6 team may be using bot comments to shape public opinion, aiming to maintain positive expectations among the audience. This could be interpreted as a form of marketing strategy.





Even though we have this hypothesis, it cannot be considered fully valid without clear and rigorous verification. Rather, it is worth reflecting on what aspects of our research process could have been improved. Given more time, there are two areas where we believe significant improvements could have been made.

First, regarding manual labeling calibration and sample size: while it was a good practice to have at least three people review and calibrate each sentiment label, the total number of samples used for sentiment analysis was relatively limited. This may have resulted in certain patterns or features being overlooked.

Second, during RNN auto-tuning, the model often exhibited overfitting. If new labeled data were to be introduced, the results could vary significantly, indicating the model's instability and limited generalizability.

Last but not least, the task itself had a certain degree of difficulty for the RNN model. Since we would like to analyze the public's sentiment *toward the product*, comments that shows strong sentiments but not toward the target product were to be labeled as "neutral", e.g. complaints about other products. As a result, the model not only needed to understand human's phrasing for different sentiments, but also had to understand the full context of the discussion, which caused extra challenges for training.

# VII. Conclusion

As mentioned above, although further improvements are required to our models, it has already revealed a possible strategy of using bots comments for marketing purposes. Furthermore, our work has established a fluent and applicable workflow on detecting the "extra" sentiments brought by the bots comments. As long as we keep on improving the model (e.g. expanding the size of the training set or tuning the model more thoroughly), we believe that the system can be used as a powerful tool to deal with the ever-increasing bot comments on the internet.

## ACKNOWLEDGEMENTS

We would like to express our gratitude to the professors and teaching assistants of the course for their guidance throughout this project. We are also thankful for the open-source tools and resources that made this work possible. In particular, we acknowledge the contributions of the online community for sharing web scraping techniques that helped us build our dataset, as well as prior research on bot detection strategies that inspired our modeling choices. The implementation of recent anomaly detection models such as Isolation Forest, One-Class SVM, and DBSCAN played a key role in our unsupervised approach to identifying suspicious online activity.

Workload Distribution:

1. Web scraper: Grace;
2. Bot Detection: Roman and Valentin
3. Sentiment Analysis: Hao-Yun
4. Visualization: Ren-Wei

## REFERENCES

[1] M. Tourond, *Reddit Bot Detector: An exploratory analysis using supervised learning*, GitHub, 2021. [Online]. Available: https://github.com/MatthewTourond/Reddit-Bot-Detector
[2] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.
[3] Felix A. Gers; Jürgen Schmidhuber; Fred Cummins (2000). "Learning to Forget: Continual Prediction with LSTM". Neural Computation. 12 (10): 2451–2471.
[4] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.